Exact Complexity Certification of a Standard Primal Active-Set Method for Quadratic Programming

Daniel Arnström and Daniel Axehill

Exact Complexity Certification of a Standard Primal Active-Set Method for Quadratic Programming

Daniel Arnström and Daniel Axehill

Abstract— **Model Predictive Control (MPC) requires an optimization problem to be solved at each time step. For real-time MPC, it is important to solve these problems efficiently and to have good upper bounds on how long time the solver needs to solve them. Often for linear MPC problems, the optimization problem in question is a quadratic program (QP) that depends on parameters such as system states and reference signals. A popular class of methods for solving QPs is primal active-set methods, where a sequence of equality constrained QP subproblems are solved. This paper presents a method for computing which sequence of subproblems a primal active-set method will solve, for every parameter of interest in the parameter space. Knowledge about exactly which sequence of subproblems that will be solved can be used to compute a worst-case bound on how many iterations, and ultimately the maximum time, the active-set solver needs to converge to the solution. Furthermore, this information can be used to tailor the solver for the specific control task. The usefulness of the proposed method is illustrated on a set of MPC problems, where the exact worst-case number of iterations a primal active-set method requires to reach optimality is computed.**

I. INTRODUCTION

Model Predictive Control (MPC) requires an optimization problem to be solved at each time step, which for linear MPC often is a quadratic program (QP) which depends on parameters such as system states and reference signals, making it a multi-parametric QP (mpQP). Often, these mpQPs are solved offline parametrically for a set of parameters and the pre-computed solution is then used online [1]. However, the complexity of the pre-computed solution grows exponentially with the dimensions of the problem and for high-dimensional problems, limited memory can restrict the use of a pre-computed solution online. For such problems, the QP has to be solved online and the limited time and computational resources often at hand in real-time MPC require the employed QP solver to be efficient and to have guarantees on the time needed to solve the QPs within a given tolerance.

Popular methods for solving QPs encountered in MPC are active-set methods [2][3][4][5], interior-point methods [6][7] and gradient projection methods [8][9][10]. Active-set methods easily integrate warm starting of the solver, i.e., the use of a previous solution to start the solver in the next iteration, which often reduces the amount of iterations needed by the solver. In the worst case the complexity of active-set methods can be exponential, in contrast to interior-point methods and some gradient projection methods for which there exist theoretical polynomial bounds on the computational complexity [6][8][11].

Recently, methods for exact complexity certification of the dual active-set QP method presented in [3] and the primal-dual active-set QP method presented in [4] have been proposed in [12] and [13], respectively. However, methods for exact complexity certification of primal active-set QP methods, such as the popular method presented in [2], have not yet been presented. The different active-set methods have pros and cons known from the literature and this work does not argue for or against any of these alternatives, but adds the possibility for real-time certification also to *primal* active-set QP methods. Still, an important advantage of a primal active-set method in the context of real-time optimization, is that it, in contrast to a dual or a primal-dual active-set method, produces iterates that are feasible with respect to the primal constraints in the QP formulation, allowing the solver to be terminated early while still ensuring a primal feasible solution.

The main contribution of this paper is a method for analyzing *exactly* which subproblems the primal active-set method presented in [2] will solve in order to compute an optimal solution for any set of parameters in an mpQP. This can ultimately be used to determine the exact computational complexity of the active-set method for a given mpQP. For a given mpQP the proposed method is used offline, giving a priori knowledge about how the active-set method will act when employed online such as a worst-case bound on the number of iterations. Furthermore, exact knowledge about the subproblems that can be encountered can be used to tailor the solver for the specific mpQP at hand. Our method is similar to the ones presented in [12], [13] and [14], where the parameter space is iteratively partitioned depending on the decision made by an active-set method in each iteration. A challenging aspect of the analysis of the primal active-set QP method considered in this work is that it turns out that all iterates are not necessarily affine in the parameter, in contrast to the methods studied in [12], [13] and [14]. Non-affine iterates are shown to lead to a partition of the parameter space consisting of both linear and quadratic inequalities, in contrast to only linear inequalites which is the case in [12], [13] and [14].

The rest of the paper is outlined as follows: Section II introduces notation and some background theory that are used in Section III to describe the primal active-set solver from [2] and some properties of the method that will be used for the complexity certification. Section IV describes the proposed method for analyzing the primal active-set

This work was supported by the Swedish Research Council (VR) under contract number 2017-04710.

D. Arnström and D. Axehill are with the Division of Automatic Control, Linköping University, Sweden {daniel.arnstrom,daniel.axehill}@liu.se

solver. The method is then illustrated on a set of examples, including MPC problems that are representative for problems encountered in real-time MPC, in Section V.

II. Preliminaries

Consider an mpQP in the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}x^T H x + (f^T + \theta^T f_\theta^T)x \\ \text{subject to} \quad & Ax \leq b + W\theta, \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$ and the parameter $\theta \in \Theta_0 \subseteq \mathbb{R}^p$, with Θ_0 being a polyhedron. The mpQP is given by $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $W \in \mathbb{R}^{m\times p}$, $f \in \mathbb{R}^n$, $f_\theta \in \mathbb{R}^{n\times p}$, and $H \in \mathbb{S}^n_{++}$. For convenience, we also introduce the notation $b(\theta) = b + W\theta$ and $f(\theta) = f + f_\theta\theta$. It is well-known that a linear MPC problem can be cast in the form in (1), where the parameter θ contains the measured/estimated state [1].

The feasible set can also be expressed in terms of each constraint as $[A]_i x \leq [b]_i + [W]_i\theta, i \in \mathcal{K}$, where the notation $[.]_i$ means the i:th row of the corresponding matrix and $\mathcal{K} = \{1, 2, ..., m\}$. If a constraint holds with equality it is said to be active.

The primal active-set method to be studied is an iterative algorithm which searches for the active constraints at the optimum, motivating the following notation. x_k is the iterate at iteration k and $\mathcal{W}_k$ is a subset of the constraints that are active at x_k called the working-set. Moreover, we define A_k, b_k and W_k to denote the rows of the matrices indexed by $\mathcal{W}_k$ and we denote the complement of $\mathcal{W}_k$ as $\mathcal{C}_k \triangleq \mathcal{K} \setminus \mathcal{W}_k$. The constrained set $P_k \triangleq \{x \in \mathbb{R}^n | A_k x = b_k(\theta)\}$ denotes the manifold defined by the working-set at iteration k and $\tilde{P}_k \triangleq \{x \in \mathbb{R}^n | A_k x = 0\}$ denotes the corresponding null space.

Next, we define two projections π_k and $\tilde{\pi}_k$ that project elements in $\mathbb{R}^n$ onto P_k and $\tilde{P}_k$, respectively, in the norm defined by the Hessian.

$$\pi_k z \triangleq \underset{x\in P_k}{\text{argmin}} ||x - z||_H^2, \quad \tilde{\pi}_k z \triangleq \underset{x\in \tilde{P}_k}{\text{argmin}} ||x - z||_H^2 \tag{2}$$

where $||.||_H$ denotes the norm given by the Hessian and is defined for $z \in \mathbb{R}^n$ as $||z||_H^2 = z^T H z$. Explicitly, the projections are given by

$$\begin{aligned} \tilde{\pi}_k z &= (I - A_k^* A_k)z \\ \pi_k z &= \tilde{\pi}_k z + A_k^* b(\theta) \end{aligned} \tag{3}$$

with $A_k^* \triangleq H^{-1}A_k^T(A_k H^{-1} A_k^T)^{-1}$. We also introduce notation for $\tilde{H}_k \triangleq A_k H^{-1} A_k^T$.

III. PRIMAL ACTIVE-SET METHOD

One important class of methods for solving QPs are active-set methods. These methods solve a sequence of subproblems with equality instead of inequality constraints to solve the original QP. In this paper we consider the primal active-set method presented in [2], summarized in Algorithm 1.

Algorithm 1 Primal Active-Set Method for QP [2]

Input: $x_0, \mathcal{W}_0, k = 1$
Output: $x_k^*, \lambda_k, \mathcal{W}_k$

while true **do**
  Compute x_k^* and λ_k by solving (5)
  $p_k \leftarrow x_k^* - x_k$
  if $p_k = 0$ **then**
    if $\lambda_k \geq 0$ **then return** $x_k^*, \lambda_k, \mathcal{W}_k$
    else $l \leftarrow \underset{i\in\mathcal{W}_k}{\text{argmin}} [\lambda_k]_i$, $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \setminus \{l\}$
  else
    $m \leftarrow \underset{i\in\mathcal{C}_k^+}{\text{argmin}} \alpha_k^i$, $\alpha_k \leftarrow \min\{1, \alpha_k^m\}$
    if $\alpha_k < 1$ **then** $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \cup \{m\}$
    $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $k \leftarrow k + 1$

Algorithm 1 starts with a feasible point x_0 and a corresponding working-set $\mathcal{W}_0$, containing indices of a subset of the constraints that are active at x_0.

Remark 1: We allow x_0 to be affine in the parameter θ, i.e., $x_0 = F_0\theta + G_0$.

In an iteration of the method, constraints are added to or removed from the working-set while maintaining primal feasibility and updating the iterate. The iterate is updated in a line search fashion [2], i.e., $x_{k+1} = x_k + \alpha_k p_k$, with the search direction p_k and the step length α_k, defined below.

The search direction p_k is the Newton step direction given by $p_k \triangleq x_k^* - x_k$, where x_k^* is the solution to the equality constrained QP problem

$$\begin{aligned} x_k^* = &\underset{x}{\text{argmin}} \ \frac{1}{2}x^T H x + f^T(\theta)x \\ &\text{subject to} \ \ A_k x = b_k(\theta) \end{aligned} \tag{4}$$

x_k^* is called a constrained stationary point, or a local optimum, and can be obtained by solving the following Karush-Kuhn-Tucker (KKT) system containing the dual variable λ_k associated with the equality constraints in (4)

$$\begin{pmatrix} H & A_k^T \\ A_k & 0 \end{pmatrix} \begin{pmatrix} x_k^* \\ \lambda_k \end{pmatrix} = \begin{pmatrix} -f(\theta) \\ b_k(\theta) \end{pmatrix} \tag{5}$$

The solution to (5) can be explicitly expressed as

$$x_k^* = -\pi_k H^{-1} f(\theta) = -\tilde{\pi}_k H^{-1} f(\theta) - A_k^* b_k(\theta) \tag{6a}$$

$$\lambda_k = -\tilde{H}_k{}^{-1}(b_k(\theta) + A_k H^{-1} f(\theta)) \tag{6b}$$

Importantly, the solution to the KKT-system in (4) is affine in θ, i.e.,

$$x_k^* = F_k^*\theta + G_k^*, \quad \lambda_k = F_k^\lambda\theta + G_k^\lambda \tag{7}$$

with $F_k^*, G_k^*, F_k^\lambda, G_k^\lambda$ defined by

$$F_k^* \triangleq \tilde{\pi}_k H^{-1} f_\theta + A_k^* W_k, \quad G_k^* \triangleq \tilde{\pi}_k H^{-1} f + A_k^* b_k \tag{8a}$$

$$F_k^\lambda \triangleq \text{-}\tilde{H}_k^{-1} W_k + A_k^{*T} f_\theta, \quad G_k^\lambda \triangleq \text{-}\tilde{H}_k^{-1} b_k + A_k^{*T} f \tag{8b}$$

The step length $\alpha_k \in [0, 1]$ is picked as the maximal step length that maintains primal feasibility. Two situations are possible. If x_k^* is feasible, a full step is taken ($\alpha_k = 1$). Otherwise, there will be at least one blocking constraint, and

α_k will then be the distance from x_k to the first blocking constraint $[A]_m x \leq [b(\theta)]_m$, $m \in \mathcal{C}_k$ in the direction of p_k. The step length selection can formally be stated as

$$\alpha_k = \min \left\{ 1, \min_{i \in \mathcal{C}_k^+} \alpha_k^i(\theta) \right\} \tag{9}$$

where we define

$$\alpha_k^j(\theta) \triangleq \frac{[b(\theta)]_j - [A]_j x_k(\theta)}{[A]_j p_k(\theta)} \tag{10}$$

$$\mathcal{C}_k^+ \triangleq \{ i \in \mathcal{C}_k |\, [A]_i p_k(\theta) > 0 \}. \tag{11}$$

α_k^j is a measure of the distance from the current iterate x_k to the hyperplane $[A]_j x = [b(\theta)]_j$ in the search direction p_k. $\mathcal{C}_k^+$ contains all constraints that are not in the working-set which can possibly be a blocking constraint.

Moreover, m is the minimizing index of the nested minimization in (9) and is added to the working-set, i.e., $\mathcal{W}_{k+1} = \mathcal{W}_k \cup \{m\}$ and $\mathcal{C}_{k+1} = \mathcal{C}_k \setminus \{m\}$, if $\alpha_k < 1$.

If $x_{k+1} = x_k^*$, global optimality is checked, which is done by checking dual feasibility, i.e, if $[\lambda_k]_i \geq 0, \forall i \in \mathcal{W}_k$. If the dual iterate is not dual feasible, a constraint corresponding to the most negative dual variable $[\lambda_k]_l$ is removed from the working-set, resulting in $\mathcal{W}_{k+1} = \mathcal{W}_k \setminus \{l\}$ and $\mathcal{C}_{k+1} = \mathcal{C}_k \cup \{l\}$.

A. Properties of Algorithm 1

The main operations of Algorithm 1 are removing and adding constraints to the working-set. We now consider properties of subsequent search directions and iterates after constraints are added to $\mathcal{W}$, discussed in III-A.1, and after constraints are removed from $\mathcal{W}$, discussed in III-A.2. These insights will later be used to certify the algorithm.

1) Addition of a constraint to $\mathcal{W}$: When a constraint is added to $\mathcal{W}$ there will be a relationship between the subsequent and previous search direction in terms of a projection as is shown in the following lemma

Lemma 1: If a constraint is added to $\mathcal{W}$ in iteration k, $p_{k+1} = (1 - \alpha_k)\tilde{\pi}_{k+1} p_k$

Proof: From the KKT-conditions the following holds

$$H x_{k+1}^* + A_{k+1}^T \lambda_{k+1} = -f(\theta) \tag{12a}$$

$$A_{k+1} x_{k+1}^* = b_{k+1}(\theta) \tag{12b}$$

Substracting $H x_{k+1}$ from (12a) gives

$$\begin{aligned} H p_{k+1} + A_{k+1}^T \lambda_{k+1} &= \text{-}f(\theta) - H x_{k+1} \\ &= \text{-}f(\theta) - H x_k^* + (1\text{-}\alpha_k) H p_k \\ &= A_k^T \lambda_k + (1 - \alpha_k) H p_k \end{aligned} \tag{13}$$

where $x_{k+1} = x_k^* - (1 - \alpha_k) p_k$ has been used in the second equality and $H x_k^* + A_k^T \lambda_k = -f(\theta)$ has been used in the third equality. Furthermore, subtracting $A_{k+1} x_{k+1}$ from (12b) gives

$$A_{k+1} p_{k+1} = b_{k+1}(\theta) - A_{k+1} x_{k+1} = 0 \tag{14}$$

where the last equality follows since $x_{k+1} \in P_{k+1}$. (13) and (14) form the KKT-system

$$\begin{pmatrix} H & A_{k+1}^T \\ A_{k+1} & 0 \end{pmatrix} \begin{pmatrix} p_{k+1} \\ \tilde{\lambda} \end{pmatrix} = \begin{pmatrix} (1 - \alpha_k) H p_k \\ 0 \end{pmatrix} \tag{15}$$

with $\tilde{\lambda} = \lambda_{k+1} - \begin{pmatrix} \lambda_k^T & 0 \end{pmatrix}^T$. (15) is in the form of (5) by identifying $f(\theta) = \text{-}(1 - \alpha_k) H p_k, b_k(\theta) = 0$, and $x_k^* = p_{k+1}$. Inserting this in (6a) gives $p_{k+1} = (1 - \alpha_k)\tilde{\pi}_{k+1} p_k$ ■

When constraints are added in consecutive iterations, Lemma 1 can be used recursively to obtain the following relationship between search directions

Corollary 1: If constraints are added to $\mathcal{W}$ from iteration k until iteration $k + N$, $p_{k+N} = (1 - \tau)\tilde{\pi}_{k+N} p_k$ for some $\tau \in [0, 1]$.

Proof: By recursively applying Lemma 1

$$p_{k+N} = \prod_{i=k+N-1}^{k} \Big((1 - \alpha_i)\tilde{\pi}_i \Big) p_k = (1 - \tau)\tilde{\pi}_{k+N} p_k$$

with $(1 - \tau) \triangleq \prod_{i=k}^{k+N-1} (1 - \alpha_i)$. The last equality follows from the intrinsic property of projections that $P_{i+1} \subseteq P_i \Rightarrow \tilde{\pi}_{i+1}\tilde{\pi}_i = \tilde{\pi}_{i+1}$. ■

After consecutive additions of constraints there will also be a relationship between the iterates in terms of projections

Corollary 2: If constraints are added to $\mathcal{W}$ from iteration 0 until iteration k, $x_k = \pi_k x_0 + \tau\tilde{\pi}_k p_0$

Proof: Using Corollary 1 gives

$$\begin{aligned} \tau\tilde{\pi}_k p_0 = \tilde{\pi}_k p_0 - p_k &= (\pi_k x_0^* - \pi_k x_0) - (x_k^* - x_k) \\ &= \text{-}\pi_k x_0 + x_k \iff x_k = \pi_k x_0 + \tau\tilde{\pi}_k p_0 \end{aligned} \tag{16}$$

where $\pi_k x_0^* = x_k^*$ has been used in the third equality and follows from (6a) and $\pi_k \pi_0 = \pi_k$ since $P_k \subseteq P_0$. ■

2) Removal of a constraint from $\mathcal{W}$: When a constraint is removed there will be a relationship between the subsequent search direction and the normal of the removed half-plane, as described by the following lemma

Lemma 2: If constraint l is removed from $\mathcal{W}_k$ in iteration k, $p_{k+1} = [\lambda_k]_l \tilde{\pi}_{k+1} H^{-1} [A]_l^T$

Proof: A constraint is removed from $\mathcal{W}_k$ when a constrained stationary point has been reached. Thus, $x_{k+1} = x_k^*$ and the search direction is given by

$$p_{k+1} = x_{k+1}^* - x_{k+1} = x_{k+1}^* - x_k^* \tag{17}$$

Since x_{k+1}^* and x_k^* are optimal, subject to $\mathcal{W}_{k+1}$ and $\mathcal{W}_k$, the following equations hold from the KKT-conditions

$$H x_k^* + A_{k+1}^T [\lambda_k]_{\bar{l}} + [A]_l^T [\lambda_k]_l = -f \tag{18a}$$

$$A_{k+1} x_k^* = b_{k+1} \tag{18b}$$

$$H x_{k+1}^* + A_{k+1}^T ([\lambda_k]_{\bar{l}} + \Delta_\lambda) = -f \tag{18c}$$

$$A_{k+1} x_{k+1}^* = b_{k+1} \tag{18d}$$

where $[.]_{\bar{l}}$ denotes all rows except the l:th row. By subtracting (18a) from (18c) and (18b) from (18d) the following KKT-system is obtained

$$\begin{pmatrix} H & A_{k+1}^T \\ A_{k+1} & 0 \end{pmatrix} \begin{pmatrix} p_{k+1} \\ \Delta_\lambda \end{pmatrix} = \begin{pmatrix} [A]_l^T [\lambda_k]_l \\ 0 \end{pmatrix} \tag{19}$$

which is in the form of (5) by identifying $f(\theta) = -[A]_l^T [\lambda_k]_l$, $b(\theta) = 0$ and $x_k^* = p_{k+1}$. Inserting this in (6a) gives

$$p_{k+1} = \tilde{\pi}_{k+1} H^{-1} [A]_l^T [\lambda_k]_l = [\lambda_k]_l \tilde{\pi}_{k+1} H^{-1} [A]_l^T \tag{20}$$

which is the stated relation. ■

Lemma 2 together with Corollary 1 gives the following fundamental property of the search directions computed by Algorithm 1

Corollary 3: At iteration $k+N$, let l be the index of the latest removed constraint from $\mathcal{W}$, removed in iteration k. Then $p_{k+N} = (1-\tau)[\lambda_k]_l \tilde{\pi}_{k+N} H^{-1}[A]_l^T$ for some $\tau \in [0,1]$.

Proof: The corollary follows by combining Corollary 1 and Lemma 2 and using $P_{i+1} \subseteq P_i \Rightarrow \tilde{\pi}_{i+1}\tilde{\pi}_i = \tilde{\pi}_{i+1}$. ■

Hence, the search directions will be *completely* determined by a projection of the normal of the latest constraint removed from $\mathcal{W}$. This property will be important in the certification of Algorithm 1, presented in the upcoming section.

IV. CERTIFICATION OF ACTIVE-SET METHOD

This section describes a method to exactly identify which sequence of working-set changes different parameters will give rise to when Algorithm 1 is applied to (1). The method is similar to the ones presented in [12], [13], and [14], in the sense that the parameter space is iteratively partitioned depending on how the working-set changes in each iteration.

There are two sources leading to a change in the active-set: either a constraint is added or removed. The second case only happens after a constrained stationary point has been reached. Moreover, if this point is a global optimum, i.e., if all the dual variables are non-negative, Algorithm 1 terminates with the global solution. Thus, Algorithm 1 can be split into two modes

a) Checking for global optimality and removing constraints, performed at lines 5-6.
b) Taking a step and adding a constraint, performed at lines 8-10.

Which mode is entered depends on whether the current iterate is a stationary point or not, i.e. if $p_k = 0$, which is checked for at line 4. Mode a) is entered if the iterate is a stationary point and mode b) is entered if it is not.

This characterization of Algorithm 1 is used to create a partition of Θ_0 reflecting which sequence of working-set changes that is performed to reach optimality. Regions are partitioned in the following way: If the region is locally optimal, it will be partitioned based on which constraint that is removed or if all dual variables are non-negative. If all dual variables are non-negative, the region is marked as globally optimal and does not have to be partitioned further. This corresponds to mode a).

Otherwise, if the region is not locally optimal, it will be partitioned based on which constraint is added or if a full step is taken. If a full step is taken the region is marked as locally optimal. This corresponds to mode b).

The procedure will be repeated for all new regions until all are marked as globally optimal. In the final partition, parameters in the same region signify that they produce the same sequence of working-set changes to reach optimality. The method is summarized in Algorithm 2.

Each region of the partition is represented by a tuple $(\Theta, \mathcal{W}, F, G, s, k, \hat{n})$ containing the following data

- $\Theta \subseteq \Theta_0 \subseteq \mathbb{R}^p$ - The subset of the parameter space that defines the region.
- $\mathcal{W}$ - The working-set in the region.
- $F \in \mathbb{R}^{n\times p}$ and $G \in \mathbb{R}^{n\times 1}$ - Matrices that define the affine mapping $x_k = F\theta + G$ for $\theta \in \Theta$.
- s - A status flag that marks if the region is locally optimal 1, globally optimal 2, or neither 0.
- k - Number of iterations performed by Algorithm 1 to reach the current state.
- $\hat{n}$ - The normal of the latest constraining half-plane that has been removed from the working-set.

S is a stack containing tuples corresponding to regions of Θ_0 that are yet to reach global optimality.

Remark 2: Algorithm 2 is well suited for parallelization by distributing the stack S over multiple processors.

Algorithm 2 Partition Θ_0 based on working-set changes

Input: $\Theta_0, \mathcal{W}_0, F_0, G_0$, mpQP
Output: FinalPartition
1: Push $(\Theta_0, \mathcal{W}_0, F_0, G_0, 0, 0, \text{NaN})$ to S
2: **while** S is not empty **do**
3: Pop p_c from S
4: **if** ISLOCALLYOPTIMAL(p_c, mpQP) **then**
5: Partition = REMOVECONSTRAINT(p_c,mpQP)
6: **else**
7: Partition = ADDCONSTRAINT(p_c, mpQP)
8: **for** p in Partition **do**
9: **if** p is global optimum **then**
10: Append p to FinalPartition
11: **else**
12: Push p to S
13: **return** FinalPartition

The procedure REMOVECONSTRAINT partitions the parameter space depending on what happens in mode a), i.e., whether global optimality is reached or if a constraint is removed. The procedure is described in detail in Section IV-B and is summarized in Algorithm 4 in the end of that section. Likewise, the procedure ADDCONSTRAINTS partitions the parameter space depending on what happens in mode b), i.e., whether a full step is taken or if a constraint is added. The procedure is described in detail in Section IV-A and is summarized in Algorithm 3 in the end of that section.

How to determine if a region is locally optimal or not, leading to the different modes, is done by the procedure ISLOCALLYOPTIMAL and is described in Section IV-C and summarized in Algorithm 5.

A. Adding constraints and taking a step

Whether a full step can be taken or if there is a blocking constraint is determined by (9). If there is an $\alpha_k^j < 1, j \in \mathcal{C}_k^+$ j is a blocking constraint. Additionally, if j is the minimizing index of the nested minimization of (9), it will be added to $\mathcal{W}_{k+1}$ and $\alpha_k = \alpha_k^j$. Hence, the set Θ_k^j of all parameters in iteration k leading to constraint j being added to $\mathcal{W}_{k+1}$ is given by

$$\Theta_k^j \triangleq \{\theta \in \Theta_k | \alpha_k^j(\theta) < \alpha_k^i(\theta), \quad \forall i \in \mathcal{C}_k^+ \setminus \{j\}, \\ [A]_j p_k(\theta) > 0, \quad \alpha_k^j(\theta) < 1\} \tag{21}$$

j being a blocking constraint is ensured by $[A]_j p_k(\theta) > 0$ and $a_k^j(\theta) < 1$, while $\alpha_k^j(\theta) < \alpha_k^i(\theta)$, $\forall i \in \mathcal{C}_k^+ \setminus \{j\}$ ensures that j is the minimizing index in (9).

Furthermore, a full step will be taken if there are no blocking constraints. This is the case when $\alpha_k^i \geq 1, \forall i \in \mathcal{C}_k^+$, leading to $\alpha_k = 1$ in (9). Hence, the set Θ_k^* of all parameters in iteration k leading to a local optimum is given by

$$\Theta_k^* \triangleq \{\theta \in \Theta_k | \alpha_k^i(\theta) \geq 1, \quad \forall i \in \mathcal{C}_k^+\} \tag{22}$$

As was discussed in the end of Section III-A, the behaviour of Algorithm 1 depends on the latest constraint that was removed from $\mathcal{W}$. Therefore, two different cases are considered when describing Θ_k^* and Θ_k^j explicitly in terms of θ. Case 1 considers the case when a constraint has not been removed since the start of Algorithm 1, i.e., if a constrained stationary point has not yet been reached, whereas Case 2 considers the case when at least one constraint has been removed from $\mathcal{W}$.

1) Case 1: Until the first constrained stationary point has been reached, the iterates $x_k(\theta)$ will not necessarily be affine in θ, which implies that α_k^j are not necessarily affine in θ from the definition in (10). To still be able to obtain explicit expressions for Θ_k^j and Θ_k^*, they will be expressed in terms of a related quantity $\tilde{\alpha}_k^j$, instead of α_k^j, defined as

$$\tilde{\alpha}_k^j(\theta) \triangleq \frac{[b]_j + [W]_j\theta - [A]_j\pi_k x_0(\theta)}{[A]_j\tilde{\pi}_k p_0(\theta)} \tag{23}$$

where x_0 is the starting iterate and $p_0(\theta) = x_0^*(\theta) - x_0(\theta)$. $\tilde{\alpha}_k^j$ can be seen as a measure of the distance between $\pi_k x_0$ and the half-plane $[A]_j x = [b(\theta)]_j$ along the search direction. Figure 1 depicts a simple two-dimensional case to capture the relationship between α_k^j and $\tilde{\alpha}_k^j$.

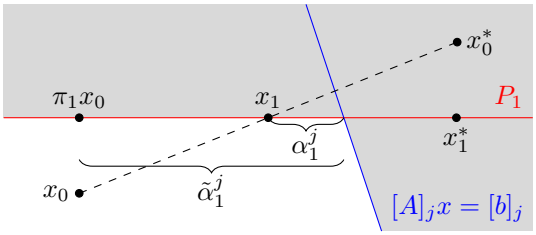


Fig. 1: Relationship between α_1^j and $\tilde{\alpha}_1^j$ for a fix θ. Note that $\tilde{\alpha}_1^j$ and α_1^j are fractions of a full step to x_1^* and not geometric distances. The white and grey areas mark the feasible set and its complement, respectively.

The following lemma makes the relationship between $\tilde{\alpha}_k^j$ and α_k^j more explicit

Lemma 3: If no constraint has been removed by Algorithm 1 up until iteration k, $\tilde{\alpha}_k^j = \tau + (1-\tau)\alpha_k^j$, $\tau \in [0, 1]$.

Proof: Since only constraints have been added to $\mathcal{W}$ since Algorithm 1 started, it follows from Corollary 1 and 2 that $x_k = \pi_k x_0 + \tau\tilde{\pi}_k p_0$ and $p_k = (1-\tau)\tilde{\pi}_k p_0$ for some $\tau \in [0, 1]$. This inserted into (10) gives

$$\begin{aligned}\alpha_k^j(\theta) &= \frac{[b(\theta)]_j - [A]_j\pi_k x_0(\theta) - \tau[A]_j\tilde{\pi}_k p_0(\theta)}{[A]_j(1-\tau)\tilde{\pi}_k p_0(\theta)} \\ &= \frac{1}{1-\tau}\tilde{\alpha}_k^j(\theta) - \frac{\tau}{1-\tau}\end{aligned}$$

which is equivalent to $\tilde{\alpha}_k^j(\theta) = \tau + (1-\tau)\alpha_k^j(\theta)$. ■

The following equivalences follow from Lemma 3 and Corollary 1: For $i, j \in \mathcal{C}_k$

$$\begin{aligned}\alpha_k^i < \alpha_k^j \Leftrightarrow \tilde{\alpha}_k^i < \tilde{\alpha}_k^j, \qquad \alpha_k^i < 1 \Leftrightarrow \tilde{\alpha}_k^i < 1 \\ \alpha_k^i \geq 1 \Leftrightarrow \tilde{\alpha}_k^i \geq 1, \qquad [A]_i p_k > 0 \Leftrightarrow [A]_i\pi_k p_0 > 0\end{aligned} \tag{24}$$

As a result, Θ_k^j and Θ_k^*, defined in (21) and (22), respectively, can equivalently be expressed in $\tilde{\alpha}$ instead of α and $\pi_k p_0$ instead of p_k. Importantly, $\tilde{\alpha}_k^j$ is a linear fraction of θ

$$\tilde{\alpha}_k^j(\theta) = \frac{B^j\theta + C^j}{D^j\theta + E^j} \tag{25}$$

with B^j, C^j, D^j and E^j given by

$$B^j \triangleq [W]_j - [A]_j(\tilde{\pi}_k F_0 + A_k^* W_k) \tag{26a}$$
$$C^j \triangleq [b]_j - [A]_j(\tilde{\pi}_k G_0 + A_k^* b_k) \tag{26b}$$
$$D^j \triangleq [A]_j\tilde{\pi}_k(F_0^* - F_0) \tag{26c}$$
$$E^j \triangleq [A]_j\tilde{\pi}_k(G_0^* - G_0) \tag{26d}$$

Θ_k^j can now be explicitly stated, by inserting (25) in (21) and rearranging terms to remove the fractions, as all $\theta \in \Theta_k$ satisfying

$$\theta^T Q^{j,i}\theta + R^{j,i}\theta + S^{j,i} < 0, \quad \forall i \in \mathcal{C}_k \setminus \{j\} \tag{27a}$$
$$-D^j\theta - E^j < 0 \tag{27b}$$
$$(B^j - D^j)\theta + (C^j - E^j) \leq 0 \tag{27c}$$

with $Q^{j,i}, R^{j,i}$ and $S^{j,i}$ given by

$$Q^{j,i} \triangleq (D^i)^T B^j - (D^j)^T B^i \tag{28a}$$
$$R^{j,i} \triangleq C^j D^i + E^i B^j - C^i D^j - E^j B^i \tag{28b}$$
$$S^{j,i} \triangleq C^j E^i - C^i E^j \tag{28c}$$

where (27a) ensures that $[A]_j x \leq [b(\theta)]_j$ is the closest blocking constraint, (27b) ensures that $[A]_j\tilde{\pi}_k p_0 > 0$ and (27c) ensures that $\tilde{\alpha}_k < 1$. Thus, the parameter space will be partitioned by quadratic and linear inequalities when a constraint is added to the working-set under Case 1.

Remark 3: Some of the quadratic terms in (27a) will be redundant. Hence, the use of redundancy removal methods is important for efficiency.

Similarly, Θ_k^* can be explicitly stated, by inserting (25) in (22), as all $\theta \in \Theta_k$ satisfying

$$(D^i - B^i)\theta + (E^i - C^i) \leq 0, \quad \forall i \in \mathcal{C}_k \tag{29}$$

2) Case 2: When a constraint has been removed from $\mathcal{W}$, it follows from Corollary 3 that the search direction will be related to the latest removed constraint l, removed in iteration $\tilde{k}$, by

$$p_k(\theta) = (1-\tau)[\lambda_{\tilde{k}}]_l\tilde{\pi}_k H^{-1}[A]_l^T = \gamma(\theta)\tilde{\pi}_k\hat{p} \tag{30}$$

with the scaling factor $\gamma(\theta) \triangleq (1-\tau)[\lambda_{\tilde{k}}]_l$ and the distorted normal $\hat{p} \triangleq -H^{-1}[A]_l^T$. The following lemma shows that the iterates also have a simple structure: all subsequent iterates will be affine in θ after a constraint has been removed.

Lemma 4: If a constraint is removed in iteration κ, $x_k = F_k\theta + G_k$, $\forall k > \kappa$ for some $F_k \in \mathbb{R}^{n\times p}, G_k \in \mathbb{R}^n$.

Proof: Without loss of generality, let $\tilde{k} \geq \kappa$ be the latest iteration in which a constraint was removed and let l be the corresponding index of the constraint that was removed. Now, assume that $x_k = F_k\theta + G_k$ for $k > \tilde{k} \geq \kappa$ and first consider the case when there is a blocking constraint. Let j be the corresponding index of the first blocking constraint and let $\hat{p}$ and $\gamma(\theta)$ be defined as above. The assumption together with (30) inserted into (10) gives

$$\alpha_k(\theta) = \alpha_k^j(\theta) = \frac{[b]_j + [W]_j\theta - [A]_j(F_k\theta + G_k)}{\gamma(\theta)[A]_j\tilde{\pi}_k\hat{p}} \quad (31)$$

Moreover, recall that the subsequent iterate x_{k+1} is given by

$$x_{k+1}(\theta) = x_k(\theta) + \alpha_k(\theta)p_k(\theta) \quad (32)$$

By inserting (31) and (30) in (32), after simplification, one gets $x_{k+1} = F_{k+1}\theta + G_{k+1}$, where F_{k+1} and G_{k+1} are given by

$$\begin{aligned} F_{k+1} =& F_k + \tilde{\pi}_k\hat{p}\frac{[W]_j - [A]_jF_k}{[A]_j\tilde{\pi}_k\hat{p}} \\ G_{k+1} =& G_k + \frac{[b]_j - [A]_jG_k}{[A]_j\tilde{\pi}_k\hat{p}}\tilde{\pi}_k\hat{p} \end{aligned} \quad (33)$$

If instead there are no blocking constraints, $x_{k+1} = x_k^*$, which is affine in θ by (7), completing the induction step.

Similarly, since a constraint was removed in iteration $\tilde{k}$, $x_{\tilde{k}+1} = x_{\tilde{k}}^*$, which is affine in θ by (7). This proves the base case and the lemma follows by induction. ∎

Now, Lemma 4 can be used together with (30) to express the step length α_k^j in (10) as

$$\alpha_k^j(\theta) = \frac{B^j\theta + C^j}{\gamma(\theta)E^j} \quad (34)$$

with $B^j, C^j$and E^j given by

$$B^j \triangleq [W]_j\text{-}[A]_jF_k, \;\; C^j \triangleq [b]_j\text{-}[A]_jG_k, \;\; E^j \triangleq [A]_j\tilde{\pi}_k\hat{p}$$

Moreover, Lemma 4 can also be used to express the denominator of (10) as

$$[A]_jp_k(\theta) = [A]_j(F_k^* - F_k)\theta + [A]_j(G_k^* - G_k) \quad (35)$$

Now, by inserting (34) and (35) in (21), Θ_k^j can be explicitly stated as all $\theta \in \Theta_k$ satisfying

$$(E^iB^j - E^jB^i)\theta + (C^jE^i - C^iE^j) < 0, \quad \forall i \in \mathcal{C}_k \setminus \{j\} \quad (36a)$$

$$[A]_j(F_k^* - F_k)\theta + [A]_j(G_k - G_k^*) < 0 \quad (36b)$$

$$([W]_j - [A]_jF_k^*)\theta + ([b]_j - [A]_jG_k^*) \leq 0 \quad (36c)$$

Remark 4: (36) only contains affine inequality constraints, in contrast to (27) where also quadratic inequalities are introduced.

Furthermore, by inserting (35) in (22), Θ_k^* can be explicitly stated as all $\theta \in \Theta_k$ satisfying

$$([A]_iF_k^* - [W]_i)\theta + ([A]_iG_k^* - [b]_i) \leq 0, \quad \forall i \in \mathcal{C}_k \quad (37)$$

Remark 5: Since all inequalities introduced in Case 2 are affine and that Case 1 never occurs again once it has been left, once the first constrained stationary point has been found *all further partitioning of the parameter space will exclusively be done by half-planes.*

The result from Section IV-A.1 and IV-A.2 are summarized in Algorithm 3, which describes how regions of the parameter space that are not locally optimal are partitioned.

Algorithm 3 Partition Θ based on if a full step is taken or if a constraint is added to $\mathcal{W}$. (Case 1/Case 2)

```
ADDCONSTRAINT((Θ, W, F, G, s, k, n̂) , mpQP)
++k
Compute F* and G* according to (8a)
for all i in C do
    Calculate Θ^i according to (27)/(36)
    Calculate F_+ and G_+ according to (-)/(33)
    if Θ^i ≠ ∅ then
        Append (Θ^i, W ∪ {i}, F_+, G_+, 0, k, n̂) to P
Calculate Θ* according to (29)/(37)
if Θ* ≠ ∅ then
    Append (Θ*, W, F*, G*, 1, k, n̂) to P
return P
```

B. Removing constraints and checking for global optimality

A global optimum has been found at iteration k if all $\lambda_k(\theta)$ are non-negative. Otherwise, a constraint l corresponding to a negative dual-variable is removed from the working-set. From Algorithm 1 line 6, l is chosen as

$$l = \underset{i \in \mathcal{W}_k}{\operatorname{argmin}} [\lambda_k(\theta)]_i \quad (38)$$

Hence, the set Θ_k^j of all parameters in iteration k resulting in constraint $j \in \mathcal{W}_k$ being removed from the working-set is given by

$$\begin{aligned} \Theta_k^j = \{\theta \in \Theta_k | & [\lambda_k(\theta)]_j < 0 \\ & [\lambda_k(\theta)]_j < [\lambda_k(\theta)]_i, \forall i \in \mathcal{W}_k \setminus \{j\}\} \end{aligned} \quad (39)$$

The set Θ_k^* of all parameters in iteration k resulting in a global optimum is given by

$$\Theta_k^* = \{\theta \in \Theta_k | [\lambda_k(\theta)]_i \geq 0, \forall i \in \mathcal{W}_k\} \quad (40)$$

Let I^j be a $|\mathcal{W}_k| \times |\mathcal{W}_k|$ matrix defined as the identity matrix with the j:th column replaced by negative ones. Also recall from (7) that $\lambda_k(\theta)$ is affine in θ, i.e., $\lambda_k(\theta) = F_k^\lambda\theta + G_k^\lambda$. Using this, the regions Θ_k^j and Θ_k^* defined in (39) and (40), respectively, can be equivalently expressed as

$$\begin{aligned} \Theta_k^j =& \{\theta \in \Theta_k | I^j\lambda_k(\theta) > 0\} \\ =& \{\theta \in \Theta_k | -I^jF_k^\lambda < I^jG_k^\lambda\} \end{aligned} \quad (41)$$

$$\Theta_k^* = \{\theta \in \Theta_k | -F_k^\lambda\theta \leq G_k^\lambda\} \quad (42)$$

How regions of the parameter space that are locally optimal are partitioned is summarized in Algorithm 4.

Remark 6: Importantly, all further partitioning in (41) and (42) are made by hyperplanes.

Remark 7: An efficient active-set solver performs low-rank modifications to the factorization of relevant matrices when a constraint is removed or added to $\mathcal{W}$ [15]. The

same factorization techniques can be used to decrease the computational complexity of Algorithm 3 and 4.

Algorithm 4 Partition Θ based on if global optimality is reached or if a constraint is removed from $\mathcal{W}$

1: REMOVECONSTRAINT($(\Theta, \mathcal{W}, F, G, s, k, \hat{n})$, mpQP)
2: $++k$
3: Calculate F^λ and G^λ according to (8b)
4: **for** all i in $\mathcal{W}$ **do**
5: Calculate Θ^i according to (41)
6: **if** $\Theta^i \neq \emptyset$ **then**
7: Append $(\Theta^i, \mathcal{W}\backslash\{i\}, F, G, 0, k, [A]_i^T)$ to P
8: Calculate Θ^* according to (42)
9: **if** $\Theta^* \neq \emptyset$ **then**
10: Append $(\Theta^*, \mathcal{W}, F, G, 2, k, \hat{n})$ to P
11: **return** P

C. Checking for local optimality

If a full step was taken in the previous iteration, the next iterate will be a locally optimal point. However, there are cases when a local optimum is reached without a full step being taken in the previous iteration. To detect these cases, $p_k = 0$, or equivalently $x_k^* = x_k$, has to be checked explicitly. As has been discussed in Section IV-A, the iterates will not necessarily be affine in the parameter, and the same can be said about the search direction p_k. Thus, the analysis is split into the same two cases as before.

From Corollary 1 it follows that $\tilde{\pi}_k p_0 = 0 \Rightarrow p_k = 0$ in Case 1. Hence, the region is locally optimal if $\tilde{\pi}_k p_0 = 0$.

The affine structure in Case 2 simply means that $F_k^* = F_k$ and $G_k^* = G_k$ give $x_k^* = x_k$. The procedure for determining if a region is locally optimal is summarized in Algorithm 5.

Algorithm 5 Check if a region has reached local optimality

1: ISLOCALLYOPTIMAL($(\Theta, \mathcal{W}, F, G, s, k, \hat{n})$, mpQP)
2: Compute F^* and G^* according to (8a)
3: **if** $s == 1$ **then return** `true`
4: **if** Case 1 **then return** $\tilde{\pi}_k p_0 == 0$
5: **else return** $(F^* == F$ **and** $G^* == G)$

D. Special cases

As has been shown in (27), the application of the proposed method to a general mpQP might result in a partitioning of the state space using not only affine but also quadratic inequalities. The significant consequence of this is during the pruning of empty regions, done at line 7 and 10 of Algorithm 3 and line 6 and 9 of Algorithm 4, since to check consistency of the combination of linear and quadratic constraints is non-trivial. However, there are some relevant cases when the partitioning is solely composed of affine constraints, resulting in an easier analysis since to check whether an intersection of half-planes is empty or not can be done by solving one LP. Such special cases are described below.

1) No state constraints: In the case when there are no constraints on the states, a linear MPC problem can be formulated as an mpQP with $W = 0$. Additionally, an admissible control input can be picked as a fixed starting point, i.e., $F_0 = 0$. This will result in $B^j = 0$ in (26a) which in turn results in $Q^{j,i} = 0$ in (28a). Therefore, all partitioning of the parameter space will be done using half-planes, leading to a polytopic partition.

2) Starting in a constrained stationary point: When the initial point is a constrained stationary point, partitioning according to Case 1 will never occur. Since all further partitioning of the parameter space in Case 2 is done by half-planes, the final partition of the state space will be polytopic, under the assumption that Θ_0 is a polyhedron. A constrained stationary point can be constructed from any feasible point by introducing artificial constraints.

V. EXAMPLES

The proposed method was tested on some benchmark problems from the MATLAB Model Predictive Control Toolbox. The MPC problems considered were the control of a double integrator, an inverted pendulum and a linearized non-linear MIMO system. The tracking problem was considered, resulting in a parameter vector containing the state vector, the previous control signal and the reference signal. The same problems were also considered in the context of real-time certification for another method in [13], where they were considered a good representation of the kind of problems encountered in real-time MPC. For further details about the problems see [13]. Additionally, the method was tested on a randomly generated mpQP to exemplify the possibility of quadratic partitioning of the parameter space. The randomly generated mpQP is given by

$$H = \begin{pmatrix} 0.97 & 0.19 & 0.15 \\ 0.19 & 0.98 & 0.05 \\ 0.15 & 0.05 & 0.99 \end{pmatrix}, \quad A = \begin{pmatrix} 0.38 & 2.20 & 0.43 \\ 0.49 & 0.57 & 0.22 \\ 0.77 & 0.46 & 0.41 \end{pmatrix}$$

$$f = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T, \qquad b = \begin{pmatrix} 4.1 & 3.7 & 4.3 \end{pmatrix}^T$$

$$W = \begin{pmatrix} 0.19 & -0.89 \\ 0.62 & -1.54 \\ -0.59 & -1.01 \end{pmatrix}, \qquad f_\theta = \begin{pmatrix} 11.3 & -44.3 \\ -3.66 & -11.9 \\ -32.6 & 7.81 \end{pmatrix}$$

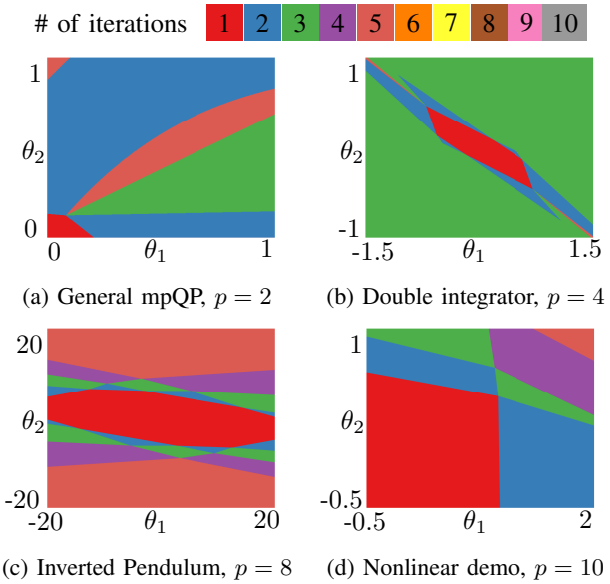
The starting iterate for all the examples was the origin, i.e. $x_0 = (0, ..., 0)^T$ and the starting working-set was the empty set, i.e. $\mathcal{W}_0 = \emptyset$. To decide if regions described by linear and quadratic inequalities were empty or not was done with YALMIP's [16] built-in global BMIBNB-solver.

The dimension of the resulting mpQPs for the examples are shown in Table I together with the maximum number of QP iterations $N_{\text{cert}}^{\max}$ determined by Algorithm 2. Furthermore, the maximum number of observed QP iterations $N_{\text{sim}}^{\max}$ when samples are extensively drawn from the parameter space and Algorithm 1 is applied to the resulting QPs is also shown in Table I.

Importantly, $N_{\text{cert}}^{\max} = N_{\text{sim}}^{\max}$ for all examples, demonstrating the validity of the proposed method. Moreover, an important aspect of the proposed method is that it covers the *entire* region of interest in the parameter space, in contrast to Monte Carlo simulations which cannot guarantee full coverage.

	p	n	m	$N_{\text{cert}}^{\max}$	$N_{\text{sim}}^{\max}$	$t_{\text{cert}}[s]$	N^{reg}
General mpQP	2	3	3	5	5	6	6
Double integrator	4	3	6	7	7	2	33
Inverted pendulum	8	5	10	26	26	567	5249
Nonlinear demo	10	6	12	13	13	187	1717

TABLE I: Dimensions of the resulting mpQPs for the examples, the worst-case number of QP-iterations $N_{\text{cert}}^{\max}$ determined by Algorithm 2 and the worst-case number of QP-iterations $N_{\text{sim}}^{\max}$ determined by extensive simulation. N^{reg} is the number of regions in the final partition and t_{cert} is the time taken by a naive MATLAB implementation of Algorithm 2 executed on an Intel 2.7 GHz i7-7500U CPU.



(a) General mpQP, $p = 2$ (b) Double integrator, $p = 4$

(c) Inverted Pendulum, $p = 8$ (d) Nonlinear demo, $p = 10$

Fig. 2: 2D-slice of the parameter space with $\theta_i = 0$, $i > 2$. The same color means same number of QP iterations.

Figure 2 depicts a low-dimensional slice of the resulting regions, parameters in the same region result in the same number of QP iterations to reach optimality. However, this is only a subset of the information contained in the final partition since every region also contains the exact sequence of working-set changes performed to reach the solution.

Remark 8: The flop count can be determined in a similar manner to [12], [14].

Remark 9: The execution time t_{cert} is based on a naive implementation of Algorithm 2 in MATLAB. Modifications to the implementation, such as low-rank modifications and parallelizing computations, are expected to significantly reduce t_{cert}.

VI. CONCLUSION AND FUTURE WORK

In this paper we have presented a method for computing *exactly* which sequence of working-set changes, as a function of the parameters in an mpQP, a primal active-set QP solver will perform to find the optimum. This can be used to determine an upper bound on the number of QP iterations the solver will need when it is applied online, which is important to know in the context of real-time MPC, where hard real-time requirements have to be fulfilled. The method partitions the parameter space into regions, defined by affine and quadratic inequalities, representing parameter sets with the same sequence of working-set changes. The proposed method was successfully applied to a set of linear MPC problems to illustrate how it can be used to determine the worst-case number of iterations needed by the solver online.

Future work include analysis of early termination of the solver as well as complexity certification when the solver is warm started. Improvements to the certification method, such as reducing redundancies and to solve the quadratic feasibility problems more efficiently, will also be considered.

REFERENCES

[1] 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.

[2] J. Nocedal and S. Wright, *Numerical Optimization*. Springer Science & Business Media, 2006.

[3] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Mathematical Programming*, vol. 27, pp. 1–33, 9 1983.

[4] K. Kunisch and F. Rendl, "An infeasible active set method for quadratic problems with simple bounds," *SIAM Journal on Optimization*, vol. 14, pp. 35–52, 01 2003.

[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: IFAC-Affiliated Journal*, vol. 18, no. 8, pp. 816–830, 2008.

[6] 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.

[7] 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.

[8] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Transactions on Automatic Control*, vol. 59, pp. 18–33, 01 2014.

[9] D. Axehill and A. Hansson, "A dual gradient projection quadratic programming algorithm tailored for model predictive control," in *2008 47th IEEE Conference on Decision and Control*. IEEE, 2008, pp. 3057–3064.

[10] S. Richter, C. N. Jones, and M. Morari, "Real-time input-constrained MPC using fast gradient methods," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE, 2009, pp. 7387–7393.

[11] ——, "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1391–1403, 2012.

[12] G. Cimini and A. Bemporad, "Exact complexity certification of active-set methods for quadratic programming," *IEEE Transactions on Automatic Control*, vol. 62, pp. 6094–6109, 2017.

[13] ——, "Complexity and convergence certification of a block principal pivoting method for box-constrained quadratic programs," *Automatica*, vol. 100, pp. 29–37, 2019.

[14] 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, pp. 1524–1534, 07 2011.

[15] I. Nielsen and D. Axehill, "Low-rank modifications of Riccati factorizations for model predictive control," *IEEE Transactions on Automatic Control*, vol. 63, no. 3, pp. 872–879, 2017.

[16] J. Löfberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *Proceedings of the CACSD Conference*, vol. 3. Taipei, Taiwan, 2004.