# Semi-Explicit Linear MPC Using a Warm-Started Active-Set QP Algorithm with Exact Complexity Guarantees

Daniel Arnström and Daniel Axehill

The self-archived postprint version of this conference paper is available at Linköping University Institutional Repository (DiVA):
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-185427

N.B.: When citing this work, cite the original publication.
Arnström, D., Axehill, D., (2021), Semi-Explicit Linear MPC Using a Warm-Started Active-Set QP Algorithm with Exact Complexity Guarantees, *2021 60TH IEEE CONFERENCE ON DECISION AND CONTROL (CDC)*, 2557-2562. https://doi.org/10.1109/CDC45484.2021.9683274

Original publication available at:
https://doi.org/10.1109/CDC45484.2021.9683274
Copyright: IEEE
http://www.ieee.org/
©2021 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

# Semi-Explicit Linear MPC Using a Warm-Started Active-Set QP Algorithm with Exact Complexity Guarantees

Daniel Arnström and Daniel Axehill

***Abstract*— We propose a semi-explicit approach for linear MPC in which a dual active-set quadratic programming algorithm is initialized through a pre-computed warm start. By using a recently developed complexity certification method for active-set algorithms for quadratic programming, we show how the computational complexity of the dual active-set algorithm can be determined offline for a given warm start. We also show how these complexity certificates can be used as quality measures when constructing warm starts, enabling the online complexity to be reduced further by iteratively refining the warm start. In addition to showing how the computational complexity of any pre-computed warm start can be determined, we also propose a novel technique for generating warm starts with low overhead, both in terms of computations and memory.**

## I. Introduction

Model Predictive Control (MPC) is a control strategy in which an optimization problem needs to be solved in each time step. For linear MPC, the optimization problems in question are quadratic programs (QPs) that depend on the current state of the system to control, making it a multi-parametric QP (mpQP). When MPC is used on embedded systems with limited memory and computational resources, the solvers that are applied to solve the resulting QPs need to be efficient and reliable.

One way of reducing the computations performed in each time step is to pre-compute the solution to the mpQP as a function of the parameter, which will be piecewise affine over polyhedral regions [1]. Online, all that is needed is to perform a simple look-up for the optimal control actions. This approach is known as *explicit* MPC and is both reliable and efficient for relatively small problems [2]. For larger problems, however, the memory required to store the optimal solution becomes unmanageable and the look-up can become too computationally demanding.

The alternative to explicit MPC, sometimes called *implicit* MPC, is to use a QP solver to solve the new QPs from scratch in each iteration. Contrary to explicit MPC, the memory requirements for implicit MPC is not a bottleneck when the dimensions of the QPs increase. This has led to the development of QP methods that are particularly suited for QPs encountered in MPC [3]–[7]. Still, a drawback of implicit MPC is that the worst-case computational complexity can be prohibitive for some QP instances that arise from the mpQP.

As a middle ground between the relatively high memory footprint and low computational complexity of explicit MPC and the relatively low memory footprint and high computational complexity of implicit MPC, several semi-explicit approaches that store compressed information to assist a QP solver online have been proposed [8]–[11].

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,axehill}@liu.se

In this paper we propose a semi-explicit approach in which the dual active-set QP algorithm presented in [7] is initialized through a pre-computed warm start. Moreover, we use the complexity certification framework presented in [12] to determine the *exact* computational complexity when the active-set algorithm is initialized with a pre-computed warm start. In addition to the reliability such certificates provide, we show that the certification method can be used as a quality measure to improve the constructions of warm starts.

The proposed semi-explicit approach is inspired by [11], where a primal active-set linear programming (LP) algorithm is initialized through a pre-computed warm start. The main differences between [11] and this paper are that we consider the, arguably, more popular mpQP formulation of linear MPC, instead of the mpLP formulation considered in [11], and that we consider a different way of representing and generating the pre-computed warm starts by exploiting structure in the dual active-set algorithm to reduce the memory and computational overhead from the warm starts.

The main contributions of this paper are, hence, twofold:

(i) We show how the complexity certification framework presented in [12] can be used to certify the complexity of parametric warm starts. This can be seen as extending the ideas presented in [11] from mpLPs to mpQPs.

(ii) We propose a method for generating parametric warm starts that does not require any regions to be explicitly stored. Instead, the regions are implicitly stored in the problem data of the QP, which makes the overhead of this semi-explicit scheme minor, both in terms of computations and memory.

## II. Preliminaries

It is well-known (see, e.g., [1]) that linear MPC problems can be cast into an mpQP in the form

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

where the decision variable $x \in \mathbb{R}^n$ is related to the control action and the parameter $\theta \in \Theta_0 \subseteq \mathbb{R}^p$ (where $\Theta_0$ is a polyhedron) is related to the state of the plant. The objective function is defined by $H \in \mathbb{S}^n_{++}, f \in \mathbb{R}^n$, and $f_\theta \in \mathbb{R}^{n\times p}$, while the feasible set is a polyhedron defined by $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$, and $W \in \mathbb{R}^{m\times p}$. Moreover, the constraints can be written componentwise as $[A]_i x \leq [b]_i + [W]_i \theta$, $i \in \mathcal{K} \triangleq \{1, \dots, m\}$, where $[\cdot]_i$ extracts the $i$:th row. More generally, $[\cdot]_{\mathcal{I}}$ denotes the operations of extracting all rows contained in an index set $\mathcal{I} \subseteq \mathcal{K}$. We also introduce the compact notation $f(\theta) \triangleq f + f_\theta \theta$ and $b(\theta) \triangleq b + W\theta$.

The main focus of this paper is to reduce the computational complexity when computing a minimizer $x^*$ for all possible QPs that can arise in the mpQP in (1) (i.e., $\forall \theta \in \Theta_0$). The algorithm considered herein for solving a resulting QP (i.e., when $\theta$ is given) is a dual active-set algorithm that is introduced in Section II-A.

In a dual algorithm, the QPs in (1) are not solved directly, but instead their so-called *dual* is considered, which is another related QP [13]. By introducing the matrices and vectors

$$M \triangleq AR^{-1},\ v(\theta) \triangleq R^{-T} f(\theta),\ d(\theta) \triangleq b(\theta) + Mv(\theta), \tag{2}$$

where $R$ is an upper Cholesky factor of $H$ (i.e., $H = R^T R$ and $R$ is upper triangular), the dual of (1) is

$$\underset{\lambda \geq 0}{\text{minimize}} \quad \frac{1}{2} \lambda^T M M^T \lambda + d(\theta)^T \lambda, \tag{3}$$

where $\lambda \in \mathbb{R}^m$ is called a dual variable. If a minimizer $\lambda^*$ to (3) has been computed, a minimizer $x^*$ to (1) can (by the stationarity KKT condition) be calculated by

$$x^* = -H^{-1}(A\lambda^* + f(\theta)). \tag{4}$$

## A. A dual active-set algorithm

In this paper we consider the dual active-set algorithm given in Algorithm 1 for solving QPs generated by the mpQP in (1). A brief overview of the algorithm is given below and we refer the reader to [7] for a more detailed description.

The algorithm operates on the dual problem (3) and tries to find the *optimal active set*:

*Definition 1 (Optimal active set):* The optimal active set $\mathcal{A}^* : \Theta_0 \to \mathbb{P}(\mathcal{K})$ as a function of $\theta \in \Theta_0$ is defined as $\mathcal{A}^*(\theta) \triangleq \{i \in \mathcal{K} : [A]_i x^*(\theta) = [b]_i + [W]_i \theta\}$, where $\mathbb{P}(\mathcal{K})$ denotes the power set of $\mathcal{K}$.

If the optimal active set is known, an optimal solution $x^*$ to (1) can be obtained directly by solving a single system of linear equations. Because of this, the main objective of active-set algorithms is to identify the optimal active set by iteratively updating a so-called working set $\mathcal{W}$. The working set $\mathcal{W}$ can be seen as an estimate of the optimal active set and is, in Algorithm 1, updated by adding/removing a single index to/from it in each iteration.

Since Algorithm 1 is iterative, $\mathcal{W}_k$ and $\lambda_k$ denote the working set and dual variable at iteration $k$. Moreover, $\bar{\mathcal{W}}_k$ denotes the complement of $\mathcal{W}$ at iteration $k$, i.e., $\bar{\mathcal{W}}_k \triangleq \mathcal{K} \setminus \mathcal{W}_k$. Finally, we use the shorthand notations $M_k \triangleq [M]_{\mathcal{W}_k}$, $d_k \triangleq [d]_{\mathcal{W}_k}$, $\bar{M}_k \triangleq [M]_{\bar{\mathcal{W}}_k}$, and $\bar{d}_k \triangleq [d]_{\bar{\mathcal{W}}_k}$ for convenience.

If $\mathcal{W}_0$ could be initialized close to $\mathcal{A}^*$, Algorithm 1 would require few iterations before terminating. This motivates *warm-starting* the algorithm, where the objective is to make an intialization such that $\mathcal{W}_0 \approx \mathcal{A}^*$.

**Algorithm 1** Dual active-set method for solving (1) for a given $\theta \in \Theta_0$[7].

```
Input: $M, d(\theta), v(\theta), R^{-1}, \mathcal{W}_0, \lambda_0$
Output: $x^*, \lambda^*, \mathcal{A}^*$
 1: while true do
 2:   if $M_k M_k^T$ is nonsingular then
 3:     $[\lambda_k^*]_{\mathcal{W}_k} \leftarrow$ solve $M_k M_k^T [\lambda_k^*]_{\mathcal{W}_k} = -d_k(\theta)$
 4:     if $\lambda_k^* \geq 0$ then
 5:       $[\mu_k]_{\bar{\mathcal{W}}_k} \leftarrow \bar{M}_k M_k^T [\lambda_k^*]_{\mathcal{W}_k} + \bar{d}_k(\theta),\ \lambda_{k+1} \leftarrow \lambda_k^*$
 6:       if $\mu_k \geq -\epsilon_p$ then optimum found, goto 16
 7:       else  $j \leftarrow \text{argmin}_{i \in \bar{\mathcal{W}}_k} [\mu_k]_i,\ \mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \cup \{j\}$
 8:     else ($\lambda_k^* \ngeq 0$)
 9:       $p_k \leftarrow \lambda_k^* - \lambda_k,\ \mathcal{B} \leftarrow \{i \in \mathcal{W}_k : [\lambda_k^*]_i < 0\}$
10:       $[\lambda_{k+1}, \mathcal{W}_{k+1}] \leftarrow$ FIXCOMPONENT$(\lambda_k, \mathcal{W}_k, \mathcal{B}, p_k)$
11:   else ($M_k M_k^T$ singular)
12:     $[p_k]_{\mathcal{W}_k} \leftarrow$ solve $M_k M_k^T [p_k]_{\mathcal{W}_k} = 0,\ p_k^T d(\theta) < 0$
13:     $\mathcal{B} \leftarrow \{i \in \mathcal{W}_k : [p_k]_i < 0\}$
14:     $[\lambda_{k+1}, \mathcal{W}_{k+1}] \leftarrow$ FIXCOMPONENT$(\lambda_k, \mathcal{W}_k, \mathcal{B}, p_k)$
15:   $k \leftarrow k + 1$
16: return $x^* \leftarrow -R^{-1}(M_k^T [\lambda_k^*]_{\mathcal{W}_k} + v), \lambda_k^*, \mathcal{W}_k$

17: procedure FIXCOMPONENT$(\lambda_k, \mathcal{W}_k, \mathcal{B}, p_k)$
18:   $j \leftarrow \text{argmin}_{i \in \mathcal{B}} - [\lambda_k]_i / [p_k]_i$
19:   $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \setminus \{j\},\ \lambda_{k+1} \leftarrow \lambda_k + (-[\lambda_k]_j / [p_k]_j) p_k$
```

*Remark 1:* Most of the ideas herein are not limited to Algorithm 1; any active-set algorithm covered by the framework in [12] could be considered. The dual algorithm in Algorithm 1 is, however, particularly well-suited when considering warm starts since the algorithm can be initialized with *any* nonnegative starting iterate ($\lambda_0 \geq 0$). In contrast, primal active-set algorithms often require more work to find a feasible starting iterate given a working set.

*Remark 2 (Forming problem):* Before Algorithm 1 is applied to a QP originating from (3) for a fixed $\theta \in \Theta_0$, $d(\theta)$ and $v(\theta)$ are computed, which, from (2), are both affine functions of $\theta$.

## III. Complexity-certified warm starts

From Definition 1 it is clear that the optimal active set varies for different parameters $\theta$, which in turn means that a good starting guess such that $\mathcal{W}_0 \approx \mathcal{A}^*$ in Algorithm 1 will, also, be parameter dependent. The main idea is, therefore, to initialize Algorithm 1 with different starting working sets $\mathcal{W}_0$ for different regions of the parameter space. In this section we formalize parameter dependent warm starts to Algorithm 1 and show how the quality of such warm starts can be evaluated $\forall \theta \in \Theta_0$.

### A. Parametric warm starts

First, we need to split $\Theta_0$ into regions on which Algorithm 1 is initialized with different $\mathcal{W}_0$. This motivates the introduction of *polyhedral partitions*:

*Definition 2 (Polyhedral partition):* The polyhedral collection $\{\Theta^i\}_{i=1}^N$ is said to be a *polyhedral partition* of $\Theta_0$ if $\cup_{i=1}^N \Theta^i = \Theta_0$ and if $\Theta^i \cap \Theta^j = \emptyset, \forall i \neq j$.

A *parametric warm start* of Algorithm 1 is then defined as a collection of pairings of regions $\Theta^i \subseteq \mathbb{R}^p$ and starting working sets $\mathcal{W}_0^i \subseteq \mathcal{K}$ such that the regions form a polyhedral partition of $\Theta_0$:

*Definition 3 (Parametric warm start):* The collection of tuples $\{(\Theta^i, \mathcal{W}_0^i)\}_{i=1}^N$ is said to be a *parametric warm start* of (1) on $\Theta_0$ if $\mathcal{W}_0^i \subseteq \mathcal{K}, \forall i \in \{1, \dots, N\}$ and if $\{\Theta^i\}_{i=1}^N$ is a polyhedral partition of $\Theta_0$.

Provided that a parametric warm start is available, warm-starting Algorithm 1 given a parameter $\theta$ is done in two steps. First, a parameter region $\Theta^j$ that contains $\theta$ is identified and, secondly, the associated working set $\mathcal{W}_0^j$ is used as $\mathcal{W}_0$ in Algorithm 1. These two steps are summarized in Algorithm 2.

**Algorithm 2** Warm-starting Algorithm 1 parametrically.

```
Input: $\theta$; Warm start $\{(\Theta^i, \mathcal{W}_0^i)\}_{i=1}^N$ on $\Theta_0$; $\delta_0 > 0$
Output: $x^*, \lambda^*$
1: Find $j$ such that $\theta \in \Theta^j$ for $\Theta^j \in \{\Theta^i\}_{i=1}^N$
2: Apply Algorithm 1 with initial working set $\mathcal{W}_0 = \mathcal{W}_0^j$
   and starting iterate $\lambda_0$ s.t. $[\lambda_0]_{\mathcal{W}^j} = 0,\ [\lambda_0]_{\bar{\mathcal{W}}^j} = \delta_0$
```

*Remark 3 (Parameter independendent starting iterate):* In Algorithm 2 we consider a constant (i.e., parameter independent) starting iterate $\lambda_0$ on all regions $\Theta^i$ forming the polyhedral partition. In principle one could, however, allow $\lambda_0$ to be, e.g., affine in $\theta$. There are two main advantages for restricting $\lambda_0$ to be constant: First, a constant $\lambda_0$ means that we do not have to store $\lambda_0^i$ for each region, which reduces the memory footprint. Secondly, as is shown in [12], the complexity certification becomes more tractable when a parameter independent starting iterate is used.

Considering two extreme cases of parametric warm starts in Algorithm 2 is particularly enlightening:

(i) If the parametric warm start is equal to the explicit solution of (1), only the point location is necessary and Algorithm 2, essentially, breaks down to explicit MPC [1]. When the number of optimal active sets on $\Theta_0$ is large (i.e., if $|\mathbb{A}(\Theta_0)| \gg 0$) , using explicit MPC can be expensive in terms of worst-case computational complexity and, especially, in terms of the memory needed to store the explicit solution and critical regions.

(ii) If a trivial warm start $(\Theta_0, \mathcal{W}_0)$ is used, i.e., if the same $\mathcal{W}_0$ is used for all $\theta \in \Theta_0$, no point location is required and Algorithm 2 breaks down to Algorithm 1 cold started with $\mathcal{W}_0$. For a cold start, the worst-case number of iterations can be high since a lot of constraints might be active at the solution, implying that many iterations in Algorithm 1 have to be executed to find the optimum since only a single index is added/removed from/to $\mathcal{W}$ at a time.

A "good" warm start can be seen as intermediary of these two cases, where the point location is cheaper than explicit MPC (primarily in terms of the memory footprint) while being informative enough to result in fewer iterations in Algorithm 1 compared with a cold start.

## B. Complexity certification

Next, we consider how to determine the quality of a given parametric warm start. In [12], a method for certifying the computational complexity of a class of well-known active-set algorithms is presented, and Algorithm 2 belongs to this class. The certification method determines the worst-case computational complexity for solving any possible QP instance that can arise from a given mpQP and a polyhedral parameter set $\Theta$. Of special interest for our purpose is that the method in [12] can certify the worst-case complexity when the active-set algorithm is initialized with any nonnegative starting iterate that is affine in $\theta$ and with any working set. In other words, [12] provides a function $\bar{\kappa} = \textbf{certComplexity}(\text{mpQP}, \Theta, \mathcal{W}, \lambda)$ that determines the worst-case complexity $\bar{\kappa}$ when Algorithm 1 is initialized with $\mathcal{W}_0 = \mathcal{W}$ and $\lambda_0 = \lambda$ for all parameters $\theta \in \Theta$. Herein we consider $\bar{\kappa}$ to be a measure of the worst-case iteration complexity, but more generally $\bar{\kappa}$ can be a measure of the complexity in terms of, e.g., flops.

Hence, the function **certComplexity** can be used to determine the worst-case computational complexity for a given warm start by applying it separately for each pair $(\Theta^i, \mathcal{W}_0^i)$, which is summarized in Algorithm 3.

**Algorithm 3** Complexity certification of Algorithm 2 using the complexity certification method from [12].

```
Input: Warm start $\{(\Theta^i, \mathcal{W}_0^i)\}_{i=1}^N$ on $\Theta_0$; $\lambda_0 > 0$
Output: Worst-case computational complexity $\bar{\kappa}$
1: for $i \in \{1, \dots, N\}$ do
2:     $\bar{\kappa}^i$ = certComplexity(mpQP, $\Theta^i, \mathcal{W}_0^i, \lambda_0$)
3: $\bar{\kappa} = \max\{\bar{\kappa}^1, \dots, \bar{\kappa}^N\}$
```

The remainder of the paper describes specific ways of generating parametric warm starts, but we want to stress that Algorithm 3 can certify the complexity of *any* given parametric warm start, no matter the means of how it was obtained. Alternative ways of generating effective parametric warm starts for a given mpQP is an interesting subject for future research, where Algorithm 3 can be used to discern the best warm start from multiple alternatives.

*Remark 4 (Clarification of the method presented in [12]):* To be precise, the method in [12] partitions a polyhedral region $\Theta$ into multiple subregions $\tilde{\Theta}^j$ on which the worst-case computational complexity $\kappa^j$ might differ. For our purpose, however, only the worst-case complexity for when $\mathcal{W}$ is used on the entire region $\Theta$ is of interest and therefore the worst-case complexity $\bar{\kappa}$ that **certComplexity** outputs is given by $\bar{\kappa} = \max_j \kappa^j$.

*Remark 5 (Taking into account the point location):* In general, the point location performed in Step 1 in Algorithm 2 might require a significant amount of computations. However, in Section IV-B we propose a partition of $\Theta_0$ that allows the point locations to be done implicitly by a lookup in a static hash table, which can be done in $\mathcal{O}(1)$ time. We have therefore, for simplicity, omitted analyzing the computations needed during the point location in this paper.

## IV. Generating parametric warm starts

Next, we try to answer the following question: Given an mpQP in the form (1), how do you construct a parametric warm start with a low memory footprint that yields a low computational complexity in Algorithm 2?

We divide the construction of a parametric warm start into two steps: 1) Generate a partition of $\Theta_0$. 2) for each region, find a starting working set $\mathcal{W}_0$. In Section IV-A we address 2) by proposing techniques for selecting a working $\mathcal{W}^i$ given a region $\Theta^i$. In Section IV-B we address 1) by proposing a method for generating a polyhedral partition $\{\Theta^i\}_{i=1}^N$ that is suitable to use for parametric warm starts.

## *A. Selecting starting working sets*

For the moment, assume that a suitable polyhedral partition $\{\Theta^i\}_{i=1}^N$ of $\Theta_0$ is given and consider the problem of selecting starting working sets $\mathcal{W}_0^i$, $\forall i \in \{1,\dots,N\}$. That is, select $\mathcal{W}_0^i$ that minimize the computational complexity of Algorithm 1 when it is initialized with $\mathcal{W}_0 = \mathcal{W}_0^i$ for all $\theta \in \Theta^i$.

In principle, since the set of all possible working sets $\mathbb{P}(\mathcal{K})$ is finite, the starting working set $\mathcal{W}_0$ that yields the lowest computational complexity for all parameters in region $\Theta^i$ could be determined by computing the worst-case complexities $\bar{\kappa} = \textbf{certComplexity}(\text{mpQP}, \Theta^i, \mathcal{W}, \lambda_0)$ for all $\mathcal{W} \in \mathbb{P}(\mathcal{K})$ and then selecting $\mathcal{W}$ that minimizes $\bar{\kappa}$. However, this is not feasible in practice for large problems since $|\mathbb{P}(\mathcal{K})| = 2^m$, that is, the exponential growth in possible $\mathcal{W}_0$ renders a brute-force search intractable.

To circumvent the exponential nature of a brute-force search, heuristics can be used to find a set of candidates $\mathbb{W} = \{\mathcal{W}_j\}_j$ and then the candidate that leads to the lowest computational complexity on $\Theta^i$ can be selected, i.e.,

$$\mathcal{W}_0^i \in \underset{\mathcal{W}\in\mathbb{W}}{\text{argmin}}\, \textbf{certComplexity}(\text{mpQP}, \mathcal{W}, \Theta^i, \lambda_0) \tag{5}$$

*Remark 6 (Brute-force search):* A brute-force search is achieved by using $\mathbb{W} = \mathbb{P}(\mathcal{K})$ in (5).

A natural follow-up question to be able to use (5) in practice is then: How does one determine a suitable candidate set $\mathbb{W}$? A quick way of discriminating potential candidates is by using the so-called Hamming distance:

*Definition 4 (Hamming distance):* Let $\mathcal{A}, \mathcal{A}' \subseteq \mathcal{K}$. Then the Hamming distance $\mathcal{H} : \mathbb{P}(\mathcal{K}) \times \mathbb{P}(\mathcal{K}) \to \mathbb{N}$ between $\mathcal{A}$ and $\mathcal{A}'$ is defined as $\mathcal{H}(\mathcal{A}, \mathcal{A}') \triangleq |\mathcal{A} \cap (\mathcal{K} \setminus \mathcal{A}')|$.

The Hamming distance is a measure of how many changes (additions and removals) are required to transform one working set into another. Since Algorithm 1 transforms the initial working set $\mathcal{W}_0$ into the optimal active set $\mathcal{A}^*(\theta), \forall \theta \in \Theta^i$, by either adding or removing an index in each iteration, the Hamming distance can be used to determine a lower bound on how many iterations Algorithm 1 require. We formalize this lower bound in the following theorem:

*Theorem 1:* (Lower iteration bound) For any parameter $\theta \in \Theta$, let $k(\theta)$ denote the number of iterations needed for Algorithm 1 to solve the resulting QP in (1) when $\mathcal{W}_0$ is used as the starting working set. Then $\exists \tilde{\theta} \in \Theta$ such that $k(\tilde{\theta}) \geq \max_{\mathcal{A}' \in \mathbb{A}(\Theta)} \mathcal{H}(\mathcal{W}_0, \mathcal{A}')$, where $\mathbb{A}(\Theta)$ denotes *all* optimal active sets on $\Theta$, i.e., $\mathbb{A}(\Theta) \triangleq \{\mathcal{A} \in \mathbb{P}(\mathcal{K}) : \mathcal{A} = \mathcal{A}^*(\theta) \text{ for some } \theta \in \Theta\}$.

*Proof:* Since Algorithm 1 changes the working set one element at a time in each iteration, the minimum number of iterations required to transform $\mathcal{W}_0$ into $\mathcal{A}' \in \mathbb{A}(\Theta)$ is $\mathcal{H}(\mathcal{W}_0, \mathcal{A}')$. Moreover, by definition of $\mathbb{A}(\Theta)$, for any $\mathcal{A}' \in \mathbb{A}(\theta)$ there exist parameters in $\Theta$ for which $\mathcal{W}_0$ has to be transformed into $\mathcal{A}'$. The algorithm will, hence, at least require $\max_{\mathcal{A}' \in \mathbb{A}(\Theta)} \mathcal{H}(\mathcal{W}_0, \mathcal{A}')$ iterations. ∎

Hence, Theorem 1 can be used to find a candidate $\mathcal{W}_0^i$ by computing all optimal active sets $\mathbb{A}(\Theta^i)$ using an explicit mpQP method and then considering the minmax problem

$$\mathcal{W}_0^i \in \arg \min_{\mathcal{A}\in\mathbb{P}(\mathcal{K})} \max_{\mathcal{A}'\in\mathbb{A}(\Theta^i)} \mathcal{H}(\mathcal{A}, \mathcal{A}'). \tag{6}$$

Still, minimizing over $\mathbb{P}(\mathcal{K})$ is intractable for large $m$ and a more practical alternative is to instead perform the outer minimization over $\mathbb{A}(\Theta^i)$, i.e.,

$$\mathcal{W}_0^i \in \arg \min_{\mathcal{A}\in\mathbb{A}(\Theta^i)} \max_{\mathcal{A}'\in\mathbb{A}(\Theta^i)} \mathcal{H}(\mathcal{A}, \mathcal{A}'). \tag{7}$$

*Remark 7 (Insufficiency of Hamming distance):* Keep in mind that there might exist $\mathcal{W} \in \mathbb{P}(\mathcal{K})$ that yield fewer iterations in Algorithm 1 compared with the candidates generated from (6). This stems from Algorithm 1 sometimes making redundant additions/removals to/from $\mathcal{W}$. Such redundancies can, however, be identified and analyzed using the certification method presented in [12].

## *B. Implicit point location*

When generating a polyhedral partition of $\Theta_0$ to use in a warm start, we have two opposing objectives to keep in mind. First of, $\mathbb{A}(\Theta^i)$ tend to contain more elements when $\Theta^i$ is large, which, from Theorem 1, in turn means that the lower iteration bound increases. Therefore we want to make the partition of $\Theta_0$ as fine as possible. At the same time though, we want the point location that is performed in Step 1 in Algorithm 2 to be as inexpensive as possible, both in terms of the memory for storing the regions and in terms of the computations performed in the point location itself, both of which increase for finer partitions.

In this paper we take into account both of these opposing objectives by implicitly partitioning $\Theta_0$ using $d(\theta)$. As will be shown, such an implicit partition means that no regions have to be stored (reducing the memory footprint) and that the point location becomes computationally inexpensive (since $d(\theta)$ already needs to be computed in Algorithm 1, see Remark 2). In other words, we get an informative partition with negligible overhead, both in terms of memory and computations.

A motivation behind why using $d(\theta)$ might lead to an informative partition is that $d(\theta)$ is the constraint violation at the unconstrained minimizer of (1), that is, $d(\theta) = b(\theta) - Ax_u^*(\theta)$, where the unconstrained minimizer $x_u^*(\theta) = -H^{-1}f(\theta)$. Constraints that are violated at the unconstrained minimizer are often more likely to be active at the optimizer (although they are not necessarily active). Also note that since we can certify the exact complexity of the final parametric warm start, as was described in Section III-B, we can determine the viability of using $d(\theta)$ to generate a partition for a given mpQP.

The polyhedral partition defined by $d(\theta)$ is determined as follows: for any $\mathcal{V}^i \in \mathbb{P}(\mathcal{K})$, a region $\Theta^i$ is defined as

$$\Theta^i \triangleq \left\{\theta \in \Theta_0 : [d(\theta)]_{\mathcal{V}^i} < 0, \quad [d(\theta)]_{\mathcal{K}\setminus\mathcal{V}^i} \geq 0\right\}, \tag{8}$$

which results in $\Theta^i$ containing all parameters such that the constraints in $\mathcal{V}^i$ are violated at the unconstrained minimum. Most of the regions in (8) are, however, empty and the resulting non-empty regions can be found by, e.g., solving an mpLP, as is shown in the following theorem.

*Theorem 2 (Computing regions):* The regions described in (8) are the critical regions to the mpLP

$$\begin{aligned} \underset{\tau \in \mathbb{R}^m}{\text{minimize}} \quad & \|\tau\|_1 \\ \text{subject to} \quad & [\tau]_i \geq [d(\theta)]_i, \quad i = 1, \dots m \\ & \theta \in \Theta_0. \end{aligned} \tag{9}$$

*Proof:* For $\theta$ such that $[d(\theta)]_i \geq 0$ we have that $[\tau]_i = [d(\theta)]_i$ is optimal and, hence, the $i$:th constraint in (9) is active. Likewise, $[d(\theta)]_i < 0$ results in $[\tau]_i = 0$ being optimal, i.e., that the $i$:th constraint is inactive. Therefore, the constraints that are inactive at the optimum for (9) maps onto the violated constraint sets $\mathcal{V}^i$, which in turn means that the critical regions to (9) are equal to the regions in (8). ■

As previously mentioned, a major advantage of doing the partition through $d(\theta)$ is that the regions $\{\Theta^i\}_i$ given by (8) do not have to be stored explicitly. Online, $d(\theta)$ is first computed and then $\mathcal{V}^i$ is determined by $\mathcal{V}^i = \{i \in \mathcal{K} : [d(\theta)]_i < 0\}$, which, implicitly, means that $\theta \in \Theta^i$. This can be interpreted as using $d(\theta)$ to form a binary search tree, similar to [14], as is illustrated for an example with two constraints in Figure 1. In contrast to [14], the binary search tree defined by $d(\theta)$ requires no additional preprocessing, while computing the separating half-planes used in [14] quickly becomes intractable to compute as the problem size increases [15].

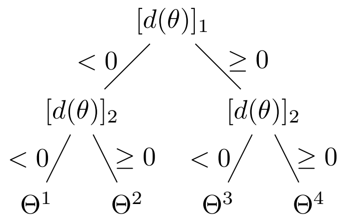


Fig. 1: Visualization of how $d(\theta)$ is used to represent a polyhedral partition of $\Theta_0$, for an example with $m = 2$.

Hence, what is needed online is a mapping that maps each $\mathcal{V}^i$, which can be stored as a binary string, to a memory location that contains the starting working set $\mathcal{W}_0^i$. An efficient way of mapping a binary string to a memory location, both in terms of storage and lookup effort, is through a static hash table [16]. A lookup in such a table can be made in $\mathcal{O}(1)$ time.

For example, consider a problem with 4 constraints. The region for which the second and third constraints are violated can be encoded as 0110, where a 1 corresponds to the constraint being violated. This means that only 4 bits in $\mathcal{V}^i$ are needed to implicitly represent a region. Generally, $\mathcal{V}^i$ requires $m$ bits to represent a region.

We summarize how to generate a warm start using $d(\theta)$ that yields a worst-case complexity of $\bar{\kappa}$ in Algorithm 4.

**Algorithm 4** Generate a warm start for (1) with worst-case complexity $\bar{\kappa}$ by using $d(\theta)$.

```
Input: mpQP
Output: Warm start $\{(\Theta^i, \mathcal{W}_0^i)\}_{i=1}^N$ on $\Theta_0$; $h$; $\bar{\kappa}$.
1: $\{\Theta^i\}_{i=1}^N \leftarrow$ generate critical regions for the mpLP in (9)
2: for $i \in \{1, \dots N\}$ do
3:     Generate $\mathcal{W}_0^i$ through, e.g., (7).
4:     $\bar{\kappa}^i \leftarrow$ certComplexity(mpQP, $\Theta^i, \mathcal{W}_0^i, \lambda_0$)
5: Generate hash function $h$ s.t. $h(\mathcal{V}^i) \rightarrow \mathcal{W}_0^i, \forall i$.
6: $\bar{\kappa} \leftarrow \max\{\bar{\kappa}^1, \dots, \bar{\kappa}^N\}$
```

In practice, Step 3 and 4 in Algorithm 4 can be repeated in an iterative fashion, where new candidates $\mathcal{W}_0^i$ are generated until the worst-case complexity $\bar{\kappa}^i$ is sufficiently low.

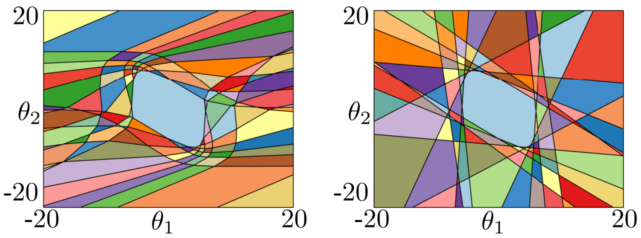
Again, we want to stress that the regions $\{\Theta^i\}_{i=1}^N$ that comprise the warm start generated in Algorithm 4 are not stored for the online use since these regions are implicitly given by $d(\theta)$ and the hash function $h$. The explicit parameter regions are only necessary for the offline complexity analysis performed in Step 4.

## V. Numerical Example

To illustrate the proposed semi-explicit approach we consider an mpQP that originates from the MPC of an inverted pendulum on a cart. This problem is part of the tutorials in the Model Predictive Control MATLAB toolbox and the same weights as in this tutorial have been used. By using a control horizon of 10, the resulting mpQP has the dimensions $n = 10$, $p = 8$ and $m = 20$. For the mpQP, Algorithm 4 was used to generate a parametric warm start.

### *A. Memory complexity*

A two-dimensional slice of the resulting partition for the generated parametric warm start is shown in Figure 2b. The partition for the explicit solution to (1) is shown in Figure 2a (which has to be stored if explicit MPC is used). By comparing Figure 2a and 2b, one can see that the granularity of both the partitions for this particular example is similar.



(a) Regions for explicit solution (b) Warm-start partition

Fig. 2: 2D-slice obtained for $\theta_i = 0 \; i \neq 1, 2$.

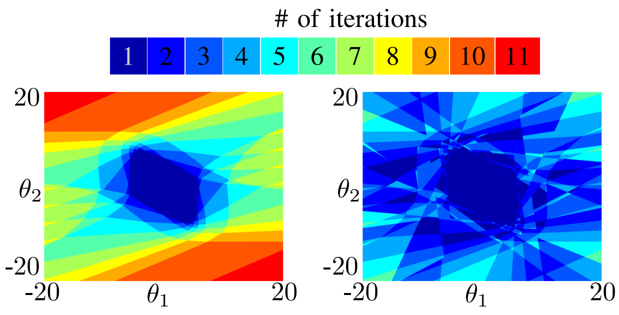
A major difference between the proposed semi-explicit approach and explicit MPC in terms of required storage can, however, be seen in Table I. The memory requirements for the proposed approach stem from storing the real numbers for $M, d(\theta), v(\theta)$ and $R$ used in Algorithm 1 together with the starting working set $\mathcal{W}_0$ on each region (since $m = 20$, 3 bytes were needed to store each starting working set). The memory requirements for explicit MPC have been computed based on the approach presented in [17], where two affine functions for each region are stored (one for the point-location problem and one for the optimal feedback law). In addition, an adjacency list that stores neighboring regions is also required to be stored.

TABLE I: Memory requirements when double precision is used (i.e., each real number is represented by 8 bytes).

| | Warm start | Explicit MPC |
|---|---|---|
| # of regions | 743 | 683 |
| Memory [kB] | 6.2 | 136.6 |

## B. Iteration complexity

The resulting iteration complexity for a two-dimensional slice when Algorithm 1 is initialized using the generated parametric warm start is visualized in Figure 3b, which is compared with a similar slice for when the algorithm is cold started with $\mathcal{W}_0 = \emptyset, \forall\theta \in \Theta_0$ in Figure 3a. The worst-case and average iteration complexity with the warm/cold start for the entire, eight-dimensional, $\Theta_0$ are reported in Table II.



(a) Cold start (b) Warm start

Fig. 3: 2D-slice obtained for $\theta_i = 0$ for $i \neq 1, 2$.

TABLE II: Number of iterations performed by Algorithm 1

| | Cold start | Warm start |
|---|---|---|
| Worst-case # iterations | 21 | 11 |
| Average # iterations | 9.0 | 6.2 |

Figure 3 and Table II show that both the worst-case and average iteration complexities are significantly decreased when the parametric warm start is used. The worst-case and average number of iterations were decreased by $48\%$ and $31\%$, respectively.

Importantly, note that since the method from [12] has been used, the reported iteration complexity is exact and covers the *entire* $\Theta_0$. Hence, the proposed warm start leads, with *complete certainty*, to a lower iteration complexity *for all* $\theta \in \Theta_0$ (not just for a finite number of samples, which is often reported when Monte-Carlo simulations are considered). This complete certainty is a strength in using Algorithm 3 to analyze parametric warm starts.

## VI. Conclusion

In this paper we have proposed a semi-explicit approach for linear MPC in which a dual active-set quadratic programming algorithm is initialized through a pre-computed warm start. By using a recently developed complexity certification method for active-set algorithms for quadratic programming, the computational complexity of the dual active-set algorithm when warm-started can be determined offline. Moreover, by exploiting structure in the dual active-set algorithm, we have proposed a novel technique for generating warm starts with low memory and computational overhead. The semi-explicit approach was illustrated on an mpQP originating from MPC of an inverted pendulum on a cart, where its memory requirements were shown to be significantly lower compared with explicit MPC and where the iterations performed by the active-set algorithm were shown to be significantly lower, both in the worst-case and on average, compared with cold-starting the algorithm.

## 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] A. Bemporad, “Explicit model predictive control,” in *Encyclopedia of Systems and Control*, J. Baillieul and T. Samad, Eds. London: Springer London, 2019, pp. 1–7.

[3] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, “qpOASES: A parametric active-set algorithm for quadratic programming,” *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[4] G. Frison and M. Diehl, “HPIPM: a high-performance quadratic programming framework for model predictive control,” *arXiv preprint arXiv:2003.02547*, 2020.

[5] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, “OSQP: An operator splitting solver for quadratic programs,” *Mathematical Programming Computation*, pp. 1–36, 2020.

[6] A. Bemporad, “A quadratic programming algorithm based on nonnegative least squares with applications to embedded model predictive control,” *IEEE Transactions on Automatic Control*, vol. 61, no. 4, pp. 1111–1116, 2015.

[7] D. Arnström, A. Bemporad, and D. Axehill, “A dual active-set solver for embedded quadratic programming using recursive LDL' updates,” *ArXiv e-prints*, 2021, arXiv:2103.16236.

[8] 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.

[9] G. Goebel and F. Allgöwer, “Semi-explicit MPC based on subspace clustering,” *Automatica*, vol. 83, pp. 309–316, 2017.

[10] F. Borrelli, M. Baotić, J. Pekar, and G. Stewart, “On the computation of linear model predictive control laws,” *Automatica*, vol. 46, no. 6, pp. 1035–1041, 2010.

[11] 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.

[12] D. Arnström and D. Axehill, “A unifying complexity certification framework for active-set methods for convex quadratic programming,” *IEEE Transactions on Automatic Control*, 2022, early access.

[13] W. S. Dorn, “Duality in quadratic programming,” *Quarterly of Applied Mathematics*, vol. 18, no. 2, pp. 155–162, 1960.

[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] M. L. Fredman, J. Komlós, and E. Szemerédi, “Storing a sparse table with 0(1) worst case access time,” *Journal of the ACM (JACM)*, vol. 31, no. 3, pp. 538–544, 1984.

[17] M. Baotić, F. Borrelli, A. Bemporad, and M. Morari, “Efficient online computation of constrained optimal control,” *SIAM Journal on Control and Optimization*, vol. 47, no. 5, pp. 2470–2489, 2008.