Quickly Finding Recursively Feasible Solutions for MPC with Discrete Variables

Liren Yang, Amey Karnik, Necmiye Ozay

Abstract— In this paper, we consider designing real-time Model Predictive Control (MPC) for embedded control applications where both continuous-valued and discrete-valued control inputs are present. The online optimization in MPC is formulated as a Mixed Integer Quadratic Program (MIQP), and can be achieved using a branch and bound algorithm where multiple relaxed Quadratic Programs (QPs) are solved. Due to the computational constraints for embedded applications, we impose a limit on the number of relaxed problems to be solved. Tailored heuristics in the branch and bound algorithm are developed taking into account the problem structure in MPC framework to generate early feasible solutions. To provide recursive feasibility guarantee to the MPC solved with such limited branch and bound algorithm, we propose to supervise the MPC with a correct-by-construction switching protocol. The paper describes these concepts, provides chronometric estimates for some problems, and gives a numerical example to explain the behavior. Moreover, a fuel cell control problem from the literature with five continuous states, three continuous inputs and three discrete inputs is used to show that the proposed approach reaches a feasible solution much faster than standard branch and bound solutions.

I. Introduction

Model Predictive Control (MPC) is widely used in process control industry because of its capability to handle constraints and to incorporate optimality [20]. Due to the increasing computational power of microprocessors, it is now applicable to real-time embedded control applications as well [25], [10]. However, MPC suffers from several potential challenges when used in embedded applications where the dynamics are relatively fast and the real-time computation needs to be done with limited computational resources. While linear and gain-scheduled MPC methods have been developed [1], the implementation for systems with both continuous and discrete actuators is more challenging [3]. Hence reducing the online computation effort becomes a key concern. In what follows, we will discuss two of the issues related to reducing the computational effort.

1) Feasibility Issue: One potential issue in real-time MPC is not being able to guarantee feasibility of the successive optimization problems [28]. In real-time applications, for simplicity of the computation, it is usually preferred to predict the system states using a nominal, linearized model. Consequently, some hard constraints may get violated at

some point due to the model mismatch. This means the online optimization problem seeking for the optimal open-loop strategy stops being feasible. We hence refer to this issue as the feasibility issue. Moreover, even if an exact model is assumed, an MPC controller may still steer the state to a region starting from where the violation of hard state constraints cannot be avoided. This happens because MPC is "greedy" in its nature, i.e., it only searches for the optimal strategy within a finite horizon. Again, in real-time applications, we may not be able to afford a sufficiently long prediction horizon due to the limited computational resources. This hence aggravates the feasibility issue.

Several solutions towards the feasibility issue are proposed. In [32], [27], the issue is handled by soft-constraints, i.e., by introducing an additional penalty term into the objective function that corresponds to a constraint violation. This requires the constraints to be calibrated more conservatively. An alternative solution is to provide a guarantee on the recursive feasibility by adding additional "artificial" constraints [18]. An MPC is called recursively feasible if it always keeps the states in a region from where the online optimization problem has a feasible solution. One way to achieve this is to restrict the states within a pre-computed robust controlled invariant set [15]. These robust controlled invariant sets, however, can have a complicated geometry. Therefore, restricting the states in these sets may introduce too many state constraints and render the online optimization problem computationally expensive to solve.

2) Discrete Variable Issue: Another drawback of real-time MPC is that it is not easy to incorporate discrete (i.e., integer) variables in the online optimization formulation. These integers represent the discrete decisions to be made at each time of execution. They may be integer numbers of some selected facilities, or they may come from actuators that are on-off switches, or continuous actuators that are limited to operate at several quantized levels. The application of MPC to such systems requires the solution of a Mixed Integer Programming (MIP) problem at each time step [6], [26]. Again, this increases the online computational burden for the embedded controller.

In this paper, we consider applying real-time MPC to embedded control problems where both continuous-valued and discrete-valued control inputs are present. A quadratic cost function and a linear plant model are used for the MPC. We formulate a Mixed Integer Quadratic Program (MIQP) that can be used in the MPC framework, and solve this MIQP by branch and bound.

Branch and bound is an algorithm design paradigm that is

developed for solving discrete optimization problems [16], [9]. The paradigm also applies to solving MIQPs. The idea is to break solving an MIQP down into solving a sequence of convex Quadratic Programs (QP), called relaxed problems. Branch and bound is pointed out in [4] to be superior to other alternative methods when applied to MIQPs, and is now a standard procedure used for solving MIQPs in many commercial solvers, e.g., Gurobi [12], CPLEX [8], which are integrated in MATLAB Hybrid Toolbox and can be applied to embedded control problems [3]. However, the time complexity for branch and bound algorithm to solve an MIQP, in the worst case, still grows exponentially with the number of discrete variables in the formulation. Since the computational resources allocated to a real-time MPC need to match the worst case computational complexity, we need to reserve considerable resources for computation if global optimality is pursued. Such high computational demand may not always be practical in real-time applications. This hence requires research on reducing or avoiding the high complexity of the branch and bound algorithm, when it is used to solve the MIQP formulation of a hybrid MPC. For example, [2] solves an MIQP with the branch and bound method, where the relaxed problems are solved with a solver tailored for MPC problems. However, branch and bound heuristics, one factor that dominates the branch and bound algorithm's performance [5], are not considered. In more recent works like [11], [22], [30] a suboptimal solution is searched without using a branch and bound algorithm. While the method in [11] is very specific to piece-wise-affine systems and hence not directly relevant to the setting in this paper, branch and bound heuristics can still be combined with the approaches from [2] and [22] to improve their performance.

In this work, we consider a simple and practical way of searching for suboptimal solutions with a branch and bound algorithm, that is, we limit the total number of relaxed problems to solve. In the rest of the paper, we refer to this approach as "limited branch and bound". This approach will provide an upper bound on the worst case complexity that is required for each execution. However, it may lead to termination with no feasible solution unless additional care is taken. To tackle this problem, we make two main contributions in this paper. First, we develop several branch and bound heuristics that are tailored for MPC. These heuristics aim at finding early feasible solutions and hence prevent the limited branch and bound algorithm from terminating with an infeasible solution. Secondly, in order to maintain recursive feasibility of the MPC with limited branch and bound, we propose to supervise the MPC controller using an invariance enforcing switching protocol. Such a switching protocol has the following notable features:

(1) it takes the form of a multi-dimensional lookup table that describes the sets of recursive feasibility (invariance) guaranteeing inputs;

(2) it can be synthesized using abstraction-based techniques [29] and is correct-by-construction; the computation for abstraction and synthesis can be demanding, but is done offline and feasible for systems with less than 10 states

(with recent developments pushing this number up);

(3) the abstraction-based synthesis framework is able to handle disturbances and hybrid models with nonlinear dynamics [24], [31], [21].

Features (1) and (2) indicate that the computational complexity is shifted from online to offline, from time to space. This hence provides an additional dimension to trade off. Feature (1) also distinguishes the proposed approach from [15], where the recursive feasibility is imposed as extra state constraints rather than control authority constraints as in this paper. This avoids adding too many state constraints in the MPC formulation or constructing a complex lookup table as in explicit MPC literature. Finally, Feature (3) reduces the occurrence of the feasibility issue due to model mismatch.

The rest of the paper is organized as follows. In Section II, we introduce the hybrid plant model considered together with the MPC notation. In Section III, we show how to formulate the MPC optimization problem as an MIQP. In Section IV, we solve the MIQP using a limited branch and bound technique and present some heuristics tuned for MPC problems. Section V discusses how the MPC can be supervised with a switching protocol enforcing invariance. Finally we illustrate the proposed framework with several numerical examples in Section VI.

II. PLANT AND MPC NOTATIONS

In this work, we consider switched systems of the form:

$$x(t+1) = A_{\sigma(t)}x(t) + B_{\sigma(t)}u(t) + K_{\sigma(t)}, \qquad (1)$$

where $x \in [\underline{x}, \overline{x}] \subseteq \mathbb{R}^n$ is the state, $u \in [\underline{u}, \overline{u}] \subseteq \mathbb{R}^m$ is the continuous control input, and $\sigma \in \{1, 2, \cdots, M\}$ is the mode and is determined by the controller. Under a certain switching mode σ, the dynamics is governed by an affine system defined by matrices A_σ, B_σ and K_σ. The term K_σ captures the fact that the system may have different zero-input equilibria for different switching modes.

We consider designing an MPC controller for the above system. The length of the prediction horizon is denoted by N. At time instant t_0, we are given the current state $x(t_0) = x_{t_0}$ and are to determine a control pair $u(t_0)$ and $\sigma(t_0)$. For this purpose, control sequences $u(t_0), \ldots, u(t_0 + N - 1)$ and $\sigma(t_0), \ldots, \sigma(t_0 + N - 1)$ are determined to optimize a quadratic cost function:

$$J = \sum_{t=t_0}^{N-1} \Big(\big(x(t+1) - x_{\text{targ}}\big)^T Q \big(x(t+1) - x_{\text{targ}}\big)$$
$$+ \big(u(t) - u_{\text{targ}}\big)^T R \big(u(t) - u_{\text{targ}}\big) \Big), \qquad (2)$$

where Q and R are positive semidefinite matrices, x_{targ} is the target state to follow, and u_{targ} is the "cheapest" control, the distance from which determines the cost of using a control u. One may also add a penalty term (see Section III-C) in the cost function to avoid overly frequent mode switching.

III. MPC WITH MIXED INTEGER PROGRAMMING FORMULATION

In this section, we formulate the MPC problem as an MIQP.

A. Naive Formulation

In what follows, we give a set of constraints, labeled as (C_1)-(C_5), that captures the switched system dynamics in Eq. (1). We will refer to this formulation as the naive formulation. This is because the formulation is intuitive but cannot be solved efficiently without any reformulation.

$$a_\sigma \in \{0, 1\}, \quad \forall \sigma, t. \tag{C_1}$$

$$\sum_{\sigma=1}^{M} a_\sigma(t) = 1, \quad \forall t, \tag{C_2}$$

$$x(t+1) = \sum_{\sigma=1}^{M} a_\sigma(t)\Big(A_\sigma x(t) + B_\sigma u(t) + K_\sigma\Big), \quad \forall t, \tag{C_3}$$

$$\underline{x} \le x(t) \le \overline{x}, \quad \forall t, \tag{C_4}$$

$$\underline{u} \le u(t) \le \overline{u}, \quad \forall t, \tag{C_5}$$

$$x(t_0) = x_{t_0}, \tag{C_6}$$

where $\forall \sigma$ is a short notation for $\forall \sigma \in \{1, 2, \ldots, M\}$, and $\forall t$ for $\forall t \in \{t_0, t_0 + 1, \ldots, t_0 + N\}$. The above formulation is explained below:

- (C_1): Binary variable $a_\sigma(t)$ represents the on-off status of mode σ at time t, i.e., $a_\sigma(t) = 1$ if the system is in mode σ at time t, and $a_\sigma(t) = 0$ otherwise.
- (C_2): Exactly one mode is on at each time instant.
- (C_3): At time instant t, the next state $x(t+1)$ is updated by the dynamics of the only active mode σ, whose associated $a_\sigma(t)$ is nonzero.
- (C_4), (C_5): State and control upper/lower bounds.
- (C_6): Initial state is equal to x_{t_0}.

Remark 1. Notice that Constraints (C_1)-(C_5) are imposed on variables $x(t)$, $u(t)$ and $a_\sigma(t)$. However, Constraint (C_3) contains bilinear terms $x(t)a_\sigma(t)$ and $u(t)a_\sigma(t)$, which break linearity and make the optimization problem hard.

B. Reformulation

The formulation is modified in this section to obtain an equivalent problem with only linear constraints. For this purpose, auxiliary variables $x_\sigma(t)$, $u_\sigma(t)$ are introduced [7]. In particular, replacing bilinear term $a_\sigma(t)x(t)$ by $x_\sigma(t)$ and $a_\sigma(t)u(t)$ by $u_\sigma(t)$, gives a linear constraint

$$x(t+1) = \sum_{\sigma=1}^{M} \Big(A_\sigma x_\sigma(t) + B_\sigma u_\sigma(t) + a_\sigma K_\sigma\Big), \quad \forall t \tag{$C_3{}'$}$$

where $x_\sigma(t)$, $u_\sigma(t)$ satisfy

$$a_\sigma(t)\underline{x} \le x_\sigma(t) \le a_\sigma(t)\overline{x}, \quad \forall \sigma, \forall t, \tag{$C_4{}'$}$$

$$a_\sigma(t)\underline{u} \le u_\sigma(t) \le a_\sigma(t)\overline{u}, \quad \forall \sigma, \forall t. \tag{$C_5{}'$}$$

Remark 2. When $a_\sigma(t) = 0$, constraint $(C_4{}')$ implies that $x_\sigma(t) = 0$; and when $a_\sigma(t) = 1$, constraint $(C_4{}')$ implies that $\underline{x} \le x_\sigma(t) \le \overline{x}$. A similar argument applies to constraint $(C_5{}')$ and $u_\sigma(t)$.

The term $x(t+1)$ on the left hand side of constraint $(C_3{}')$ can be replaced by $\sum_{\sigma=1}^{M} x_\sigma(t+1)$. Similarly, the control sequence $u(t)$ can be reconstructed from $u_\sigma(t)$, i.e., $u(t) =$

$\sum_{\sigma=1}^{M} u_\sigma(t)$. Finally, the initial condition can be enforced by

$$\sum_{\sigma=1}^{M} x_\sigma(t_0) = x_{t_0}. \tag{$C_6{}'$}$$

C. Penalty on Switching

We add a term to cost J to penalize switching, i.e., changing of the modes:

$$J_\sigma = \sum_{t=t_0}^{t_0+N-1} \sum_{\sigma=1}^{M} c_t |a_\sigma(t) - a_\sigma(t-1)|, \tag{3}$$

where c_t is a weighting coefficient that may vary with time t, and $a_\sigma(t_0-1)$ is also a parameter, whose value is determined by the mode selected at the last execution.

D. Overall Formulation

In summary, the optimization problem to solve at each execution of MPC is formulated as follows:

$$\begin{aligned} \min \quad & J + J_\sigma \\ \text{s.t.} \quad & (C_1), (C_2), (C_3{}'), (C_4{}'), (C_5{}'), (C_6{}'). \end{aligned} \tag{4}$$

In the optimization problem defined by Eq. (4), the objective is quadratic[1], the constraints are all linear and with both continuous-valued variables $x_\sigma(t)$ $u_\sigma(t)$ and binary variables $a_\sigma(t)$. The formulation is hence an MIQP.

IV. Solving MIQP Using Branch and Bound

In this section, we develop several branch and bound heuristics to solve the MIQP corresponding to the MPC problem.

A. Pseudo Code

The formal description of branch and bound algorithm is given by the following pseudo code named BRANCHAND-BOUND. This pseudo code is adopted from the one given by [14], with three minor modifications. First, we prune the "useless" branches (line 20) and mark the fully explored branches (line 12). Secondly, we explicitly update the candidate optimal solution w^* (line 18). Thirdly, each branch can generate more than two sub-problems (line 25). The procedure calls several sub-procedures, including LOWERBOUND, UPPERBOUND, BRANCH and PRUNE. These procedures are defined later in this part. Finally, the boxes mark the lines that involve the heuristics.

1: **procedure** $[J^*, w^*]$ = BRANCHANDBOUND(MIQP$_0$)
2:     $\mathcal{L}$ = {MIQP$_0$}
3:     MIQP$_0$.status = {unknown, unexplored}
4:     $LB = 0$
5:     $UB = +\infty$
6:     **while** $UB - LB \ge \varepsilon$ **do**
7:         Pick MIQP$_i \in \mathcal{L}$ that is unexplored
8:         $[lb_i, v_i]$ = LOWERBOUND(MIQP$_i$)
9:         $\boxed{[ub_i, w_i] = \text{UPPERBOUND(MIQP}_i, v_i)}$
10:         **(Rounding Heuristic)**

if $ub_i - lb_i < \varepsilon$ **then**
12:          $\text{MIQP}_i.\text{status} = \{\text{known, explored}\}$
13:      **else**
14:          $\text{MIQP}_i.\text{status} = \{\text{unknown, explored}\}$
15:      **end if**
16:      **if** $ub_i < UB$ **then**
17:          $UB = ub_i$
18:          $w^* = w_i$
19:      **end if**
20:      $\mathcal{L} = \text{PRUNE}(\mathcal{L}, UB)$
21:      $LB = \min_{\text{MIQP}_\ell \in \mathcal{L}} lb_\ell$
22:      **if** All problems in $\mathcal{L}$ are explored **then**
23:          $\boxed{\text{Pick MIQP}_j \in \mathcal{L} \text{ that is unknown}}$
24:          (**Diving Heuristic**)
25:          $\boxed{[\text{MIQP}_{j1}, \ldots, \text{MIQP}_{jK}] = \text{BRANCH}(\text{MIQP}_j)}$
26:          (**Branching Heuristic**)
27:          $\mathcal{L} = \mathcal{L} \cup \{\text{MIQP}_{j1}, \ldots, \text{MIQP}_{jK}\}$
28:          $\text{MIQP}_{jk}.\text{status} = \{\text{unknown, unexplored}\}$
29:          for $k = 1, \ldots, K$
30:      **end if**
31: **end while**
32: $J^* = UB$
33: **return** J^*, w^*
34: **end procedure**

Procedure LOWERBOUND(MIQP_i) approximates the lower bound of the minimum cost of the program MIQP_i by solving its relaxed problem QP_i. Quadratic program QP_i is obtained by replacing each binary variable in MIQP_i with a real variable in the interval [0,1]. This step returns the relaxed solution v_i and the cost lb_i of QP_i which is a lower bound of the minimum cost of MIQP_i.

Remark 3. It is worth emphasizing that the subproblems QP_i have a fixed structure, therefore, amenable to specialized fast embedded solvers [19].

Procedure UPPERBOUND(MIQP_i, v_i) approximates the upper bound of the minimum cost of MIQP_i. The relaxed solution v_i is rounded to give a feasible solution w_i to MIQP_i. We will refer to the rounding approach as the **rounding heuristic**. The approximated upper bound is then given by plugging w_i into the cost function in Eq. (2). If no feasible solution w_i is found, $+\infty$ is returned as a trivial upper bound.

Procedure BRANCH(MIQP_j) breaks MIQP_j down into K sub-problems, $\text{MIQP}_{j1}, \ldots, \text{MIQP}_{jK}$. The feasible set of MIQP_j is equal to the union of those of MIQP_{j1} through MIQP_{iK}. We will refer to the procedure splitting the feasible set as the **branching heuristic**.

Procedure PRUNE($\mathcal{L}, UB$) removes from list $\mathcal{L}$ the sub-problems that do not contribute to finding the tightest global lower bound LB. Particularly, a sub-problem MIQP_ℓ can be removed in the following two situations. First, all the children of MIQP_ℓ have already been explored, i.e., upper and lower bounded. In this case, the lower bound lb_ℓ is conservative (i.e., smaller) compared to the smallest lower bound given by its children. Hence it can be removed from list $\mathcal{L}$, i.e., replaced by its children. The second possible situation is that $lb_\ell > UB$. This indicates that the current candidate solution

w^* associated with the current bound UB gives a cost that is lower than any of MIQP_ℓ's relaxed solutions. Hence MIQP_ℓ will not contribute to the global minimum cost, and can be removed from the list $\mathcal{L}$ even if it is not branched yet.

B. Heursitics

This section describes the proposed heuristics, suitable for embedded MPC applications. Three types of heuristics, marked using boxes in the pseudo code, are necessary. We will present several options (denoted in italic) for these heuristics and indicate the heuristics implemented in bold.

1) Diving Heursitics: Whenever all the problems in list $\mathcal{L}$ are explored yet global convergence is not reached, we need to pick a sub-problem from the list $\mathcal{L}$ and split it into smaller sub-problems. This heuristic determines which sub-problem to pick. In [14], the heuristic is designed in such a way that ***the sub-problem with the smallest lower bound is picked***. This heuristic is known as "greedy-best-first" heuristic, and it aims to dive faster by picking the most "promising" leaf to branch.

2) Rounding Heursitics: In procedure UPPERBOUND, we need to round the solution of a relaxed problem to get a solution that is feasible for MIQP_i. This heuristic determines how to do the rounding.

An infeasible rounded solution gives $+\infty$ as upper bound (see procedure UPPERBOUND). Improperly rounded solution hence delays the termination of procedure BRANCHAND-BOUND. The rounding heuristics related to finding an early feasible solution is referred to as start heuristics [13]. Secondly, the solution should be rounded towards the optimal solution to rapidly reduce the global upper bound UB, and hence the termination of procedure BRANCHANDBOUND.

The most naive rounding is *to round $a_\sigma(t)$ to 1 iff $a_\sigma(t) \geq 0.5$ in the relaxed solution* [5]. However, such naive rounding completely omits the constraint $\sum_{\sigma=1}^{M} a_\sigma = 1$. With this constraint, it is more natural to *leverage the solution of the relaxed problem and round its largest $a_\sigma(t)$ to one and the rest to zero.*

Fig. 1: Illustration: max-a_σ rounding heuristic without cooperating with the feasibility.

The idea is illustrated with an example in Figure 1. To maintain feasibility after the rounding, we have two alternatives:

i) *We extract a sequence of switching modes by rounding the part of the solution that is supposed to be integer-valued before relaxation, and retain the continuous-valued variables that represent the continuous control input.* While this solution may violate the state or output constraints, the computational impact for this method is just generating the trajectory given the rounded solution w_i using dynamics.

ii) *An alternative is to re-optimize the continuous-valued control inputs under the new switching sequence defined by the rounded solution.* The drawback is that this requires to solve another QP, which lives at the bottom level of the decision tree.

A drawback of the max-a_σ rounding method is that if the states are close to their bounds, the switching sequence determined by the rounded solution may steer the trajectory outside the bounds. This leads to an infeasible rounded solution and hence delays the termination of procedure BRANCHANDBOUND. The issue is particularly significant when the number of modes is greater than 2, and when the relaxed solution does not show any particular preference to a certain mode. In this situation, the fractional values of a_σ in the relaxed solution are evenly distributed over σ. A relaxed solution like this suggests that a "fractional combination" of several different modes is required for the state to stay within its bounds. If we pick the mode σ with largest $a_\sigma(t)$, we are then ignoring the effect of other modes in the combination and this might be too aggressive.

If we have considerable continuous control authority, e.g., $\underline{u}, \overline{u}$ are loose, the issue can be solved by re-optimizing the continuous control to achieve feasibility. *To protect for the conditions where we do not have enough continuous control authority, we propose a new rounding heuristic, called "feasible-max-a_σ" rounding, with the following pseudo code:*

1: **procedure** $w_i = $ ROUND(v_i, MIQP_i)
2:    Extract $a_\sigma(t), u(t)$ for $t = 1, \ldots, N-1$ from relaxed solution v_i
3:    Get $x(0), \underline{x}, \overline{x}, A_\sigma, B_\sigma, K_\sigma$ from formulation MIQP$_i$
4:    **for** $t = 0$ to $N-1$ **do**
5:        $\sigma^* = \arg\max_\sigma a_\sigma(t)$
6:            s.t. $A_\sigma x(t) + B_\sigma u(t) + K_\sigma \in [\underline{x}, \overline{x}]$
7:        $x(t+1) = A_\sigma x(t) + B_\sigma u(t) + K_\sigma$
8:        $\widetilde{a}_{\sigma^*} = 1$ and $\widetilde{a}_\sigma = 0$ for all $\sigma \neq \sigma^*$
9:    **end for**
10:    QP$'_i$ = quadratic programming obtained by setting $a_\sigma(t) = \widetilde{a}_\sigma(t)$ in MIQP$_i$
11:    Solve QP$'_i$
12:    $w_i = $ optimal solution to QP$'_i$
13: **end procedure**

The idea is to combine approach i) and ii) and the max-a_σ rounding criteria. At each time t, we start with rounding $a_\sigma(t)$ by setting the largest $a_\sigma(t)$ to one and the rest to zero. We then regenerate the next state $x(t+1)$ under the modes determined by the rounded $a_{\sigma(t)}$ and check if the $x(t+1)$ stays inside the state bounds $[\underline{x}, \overline{x}]$ (line 5). If not, then we switch to the mode with the second largest $a_\sigma(t)$, etc. We proceed until a sequence of binary variables $\widetilde{a}_\sigma$ is obtained under which the trajectory stays in bounds within the entire horizon. The intuition behind this heuristic is that, when several modes are combined to generate a next state staying within the bounds, selecting the "strongest" mode might be too aggressive. In these cases, the other modes, e.g., the one with the second or third largest $a_\sigma(t)$, are "balancing" this

aggressive mode. Picking these other modes is more likely to lead to a feasible trajectory.

3) Branching Heuristic: In procedure BRANCH, we divide the feasible region of a problem MIQP$_j$ to obtain multiple sub-problems MIQP$_{j1}, \ldots, $MIQP$_{jK}$, whose feasible regions form a partition of that of MIQP$_j$. This heuristic determines how to form the new feasible regions and generates these new sub-problem MIQP$_{jk}$'s.

For an MPC problem where M modes are at choice at each time, we generate M sub-problems at once. This can be done by setting one of the $a_\sigma(t)$'s to be one and the rest to be zero. We refer to this heuristic as M-branching heuristic. This heuristic is different from the 2-branching heuristic commonly used in branch and bound for solving general mixed binary programs, which only creates two new sub-problems by setting one binary variable to be either zero or one. To determine mode $\sigma(t)$ at time t in the prediction horizon, the 2-branching heuristic may create $2M - 2$ sub-problems to solve in the worst case, while the former one always creates M sub-problems.

V. MPC SUPERVISED BY CORRECT-BY-CONSTRUCTION SWITCHING PROTOCOL

The methodology presented so far is henceforth called "unsupervised MPC". The heuristics developed in the previous section cannot guarantee recursive feasibility due to model mismatch or due to their greedy nature. Moreover, with the online optimization solved by a limited branch and bound algorithm, the chances of losing feasibility increase. Later in Section VI we will show a scenario where the unsupervised MPC violates a state constraint. To avoid this scenario, we propose to supervise an MPC controller by an invariance enforcing switching protocol that is correct-by-construction. The supervisor can guarantee recursive feasibility and account for the effect of model mismatch to some extent[2].

A switching protocol enforcing invariance can be synthesized offline, using abstraction-based techniques [29]. Moreover, such techniques can be used to extract the set of all allowable switching actions for a given state that will guarantee invariance [24], [23]. We propose to save this action set as a lookup table in the microprocessor. The lookup table is based on a partition of the state space and is used to generate constraints for MPC to ensure recursive feasibility. In particular, within each region R of the partition, there is
(1) a set $\Sigma(R)$ of modes,
(2) a collection of sets $U(R, \sigma)$ of continuous controls, one for each $\sigma \in \Sigma(R)$.
We call these actions invariance actions of region R. This is because the set $S = \bigcup_{R:\Sigma(R)\neq\emptyset} R$ is controlled invariant as long as one applies mode $\sigma \in \Sigma(R)$ and control $u \in U(R, \sigma)$ in region R. During execution, when the current state $x(t_0)$ is in a certain region $R(x(t_0))$, the corresponding invariance actions are imposed as control constraints on $u(t_0)$ and

---

[2]If the model used for supervisor synthesis is exact, then the effect of model mismatch is eliminated by the provable correctness of the supervisor. However, there could always be unmodeled dynamics.

$\sigma(t_0)$ in the MPC formulation. That is, we restrict $\sigma(t_0) \in \Sigma\Big(R(x(t_0))\Big)$ and $u(t_0) \in U\Big(R(x(t_0)), \sigma(t_0)\Big)$ Notice that it is sufficient to impose the constraints only for the *first* control actions, which will be carried out by MPC in practice.

The main technical challenge is to find proper continuous control action sets $U(R, \sigma)$. In abstraction-based synthesis, the continuous controls are quantized and the system is treated as a pure switched system. This means the obtained invariance action sets $U(R, \sigma)$ consists of only finitely many isolated points. Such sets are not favored in the MPC formulation where $u(t_0) \in U\Big(R(x(t_0)), \sigma(t_0)\Big)$ is imposed as a constraint, as described in the previous paragraph. This constraint wastes continuous control authority while introducing more discrete variables in the MPC formulation. To tackle this problem, we introduce a quantization error ε during switching protocol synthesis so that the solution is robust to quantization errors:

$$(\sigma, u) \text{ is an invariance pair } \Rightarrow$$
$$(\sigma, \widetilde{u}) \text{ is an invariance pair, } \forall ||\widetilde{u} - u|| \leq \varepsilon. \quad (5)$$

This allows us to merge the sets of $\widetilde{u}$'s into a connected region. Then the largest rectangular subset of that connected region is selected to simplify the induced control authority constraint.

Remark 1: The lookup tables discussed in this section should not be confused with the lookup tables used in explicit MPC literature. First, our lookup tables store switching actions guaranteeing recursive feasibility, which are used to define constraints for the QPs solved online. The construction of these lookup tables is totally independent of the objective functions in QPs. Secondly, our lookup tables usually only consist of rectangular partitions as opposed to complex polytopic partitions in explicit MPC, therefore they are easier to use at run-time. For example, in the five dimensional fuel cell thermal control system in Example 1 in Table I (see also [31]), the size of the lookup table (partition) constructed is approximately 3000, and it is relatively simple to access each piece in the partition as they are rectangular.

VI. NUMERICAL EXAMPLE

In this section, we provide several examples to illustrate the usefulness of the proposed approach. In part VI-A, we benchmark the feasible-max-a_σ rounding heuristic with several examples, and illustrate that it eases the feasibility issue when the MPC is solved with limited branch and bound, by finding a feasible solution as early as possible. In part VI-B, we give an example where the unsupervised MPC violates a state constraint, and this violation can be avoided by the proposed supervision approach.

A. Branch and Bound Heuristic

We benchmark the feasible-max-a_σ rounding heuristics with three examples. Example 1 is a fuel cell thermal management problem adopted from [31]. Examples 2, 3 are numerical examples. The relaxed problems are solved using a custom solver generated by CVXGEN [19]. Table I

compares the naive, max-a_σ and feasible-max-a_σ rounding heuristics respectively, by presenting the average number and worst case number of QPs solved (i) to achieve global optimality and (ii) to find the first feasible solution. We also compare with a simple branch and bound solver implemented in YALMIP [17]. Notice that the number of QPs (i.e., relaxed problems) to solve is directly related to the computation time. Here we do not compare the time directly as the QPs are solved with general purpose solvers in YALMIP, hence comparison of time is not meaningful.

It can be seen from TABLE I that our branch and bound implementation has better average performance than and comparable worst case performance with respect to the YALMIP branch and bound solver. Meanwhile, our branch and bound with feasible-max-a_σ rounding outperforms other heuristics and solvers in terms of finding early feasible solutions. This suggests that the feasible-max-a_σ rounding can be useful when we limit the total number of QPs to solve in branch and bound, which is important in real-time applications because it upper bounds the worst case computational complexity. With restrictions on computational budget, the feasible-max-$a_\sigma(t)$ rounding can prevent termination with no feasible solution. Fig. 2 shows the plot of sub-optimal cost versus the max allowable number of QPs to solve in branch and bound, under the max-a_σ and the feasible-max-a_σ rounding heuristics. The curve is Pareto-like because the cost of the sub-optimal solution at the termination will drop if more relaxed problems are allowed to be solved. In case no feasible solution is found at termination, the cost is infinity. In this example, both rounding heuristics give exactly the same average and max number of QPs to solve for global optimality. However, the feasible-max-a_σ rounding heuristic gives a feasible solution after solving only 2 QPs, while max-a_σ rounding finds the first feasible solution after solving over 20 QPs.
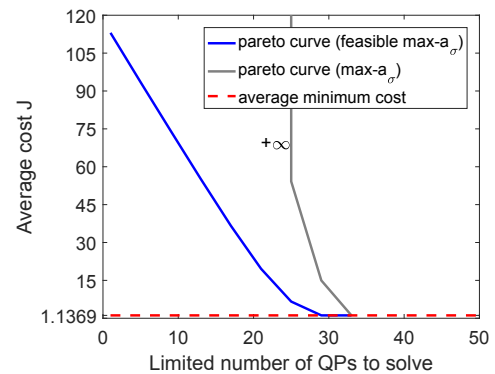

Fig. 2: Pareto-like curve: the average cost versus the limited number of QP's to solve.

B. Supervised MPC

The following example shows the necessity of the MPC supervision. Consider a switched linear system with three modes: $\dot{x} = A_\sigma x + B_\sigma u + K_\sigma, \sigma \in \{1, 2, 3\}$, where $x \in \mathbb{R}^2$,

TABLE I: Average and maximum number of relaxed problems solved to achieve global optimality under different rounding heuristics for problems with different n (dimension of state x), m (dimension of continuous control u), M (number of modes σ), and N (length of prediction horizon).

	Problem Size				Rounding Heuristics			YALMIP B&B		
	n	m	M	N	naive	max-a_σ	feasible-max-a_σ			
Example 1 (fuel cell [31])	5	3	3	9	180.06	**179.98**	**179.98**	≥ 2000	average #QPs solved	global optimal sol
					1760	**1760**	**1760**	≥ 2000	max #QPs solved	
					25.1579	8.6842	**3.0175**	15.13	average #QPs solved	1st feasible sol
					1454	26	**14**	44	max #QPs solved	
Example 2	2	1	4	8	**44. 94**	**44. 94**	**44. 94**	58.42	average #QPs solved	global optimal sol
					66	66	66	**64**	max #QPs solved	
					38.13	14.39	**2**	18.74	average #QPs solved	1st feasible sol
					58	50	**2**	21	max #QPs solved	
Example 3	2	1	3	8	**1244.90**	**1244.90**	**1244.90**	≥ 5000	average #QPs solved	global optimal sol
					3440	**3440**	**3440**	≥ 5000	max #QPs solved	
					323.50	41.58	**2**	17.29	average #QPs solved	1st feasible sol
					1526	278	**2**	56	max #QPs solved	

$u \in \mathbb{R}$ and

$$A_1 = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix}, B_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, K_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$
$$A_2 = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}, B_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, K_2 = \begin{bmatrix} 0 \\ 2 \end{bmatrix},$$
$$A_3 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, B_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.K_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}. \quad (6)$$

The control objective is to steer state x_1 to 0 from the positive side, i.e., $x_1 \geq 0$ holds. The continuous control u satisfies $-1 \leq u \leq 1$.

The model captures the dynamics of a spring-mass-damper system. For $\sigma = 1$, we have a damped oscillator with equilibrium at the origin. Since the sate should not be in the left half plane due to the state constraint $x_1 \geq 0$, mode 1 itself is insufficient to reach the equilibrium unless u required is small. When in mode $\sigma = 2$, only a constant force is applied to the damper for acceleration. Under mode $\sigma = 3$, the damper is disconnected while the spring is connected and the equilibrium is shifted.

The supervised and unsupervised MPC are designed for this system with sampling period 0.1s. The prediction horizon is 8 samples, which is selected to be short to reduce computation burden. Fig. 3 (left) shows the obtained switching protocol used for MPC supervision. The colored region is the obtained controlled invariant set. In each region, the allowable modes are plotted with a different color, and the continuous controls are upper and lower bounded to guarantee invariance. If (i) the state is initiated in the controlled invariant set, (ii) only the allowable discrete actions at that state are selected, and (iii) the continuous control obeys the upper and lower bounds, then the next state is guaranteed to stay in the controlled invariant set. The other two plots in Fig. 3 show the phase portrait of the closed-loop trajectories under the supervised (middle) and the unsupervised MPC (right). The zero-input (i.e., $u = 0$) vector fields of the three modes are plotted using different colors, and the closed-loop trajectories are plotted with a similar color whenever the dynamics is governed by the corresponding mode. The red dashed line marks the target $x_1 = 0$. The trajectory of the supervised MPC stays within the colored region and converges to $x_1 = 0$ from positive side, while the unsupervised trajectory violates the constraint $x_1 \geq 0$ on its way to the target. The simulation results are shown in Fig. 4 and these plots illustrate why the unsupervised MPC violates the state constraint. It can be seen that state x_1 converges to 0 faster (hence the cost J drops faster) under the unsupervised MPC. This suggests that the unsupervised MPC is "short sighted", in the sense that it tries to reach the target faster speed without considering recursive feasibility. Consequently, it leads to a state starting from where it is inevitable for state x_1 to drop below 0, which violates the state constraint $x_1 \geq 0$.

VII. Conclusions

In this paper, we considered MPC problems for systems with discrete actuators. For embedded applications, solving MPC problems fast is crucial. In order to limit the computation time, we proposed several branch and bound heuristics by taking into account the special structure of the MIQP problems resulting from the MPC framework. However fast solutions with limited branch and bound are often greedier and more susceptible to infeasibility. In order to eliminate potential infeasibility problems, we used the input constraints from an invariance enforcing switching protocol to supervise the MPC controller. Our framework provides a means to combine numerical objectives in MPC with infinite horizon safety guarantees from a correct-by-construction controller.

References

[1] Honeywell OnRAMP website: http://www.honeywellonramp.com.
[2] D. Axehill and A. Hansson. A mixed integer dual quadratic programming algorithm tailored for mpc. In *Decision and Control, 2006 45th IEEE Conference on*, pages 5693–5698. IEEE, 2006.
[3] A. Bemporad. Hybrid toolbox–user's guide. 2003.
[4] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.
[5] T. Berthold. Primal heuristics for mixed integer programs. 2006.
[6] E. F. Camacho, D. R. Ramírez, D. Limón, D. M. De La Peña, and T. Alamo. Model predictive control techniques for hybrid systems. *Annual reviews in control*, 34(1):21–31, 2010.
[7] Y. Cheng, Y. Wang, M. Sznaier, N. Ozay, and C. Lagoa. A convex optimization approach to model (in) validation of switched arx systems with unknown switches. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 6284–6290. IEEE, 2012.
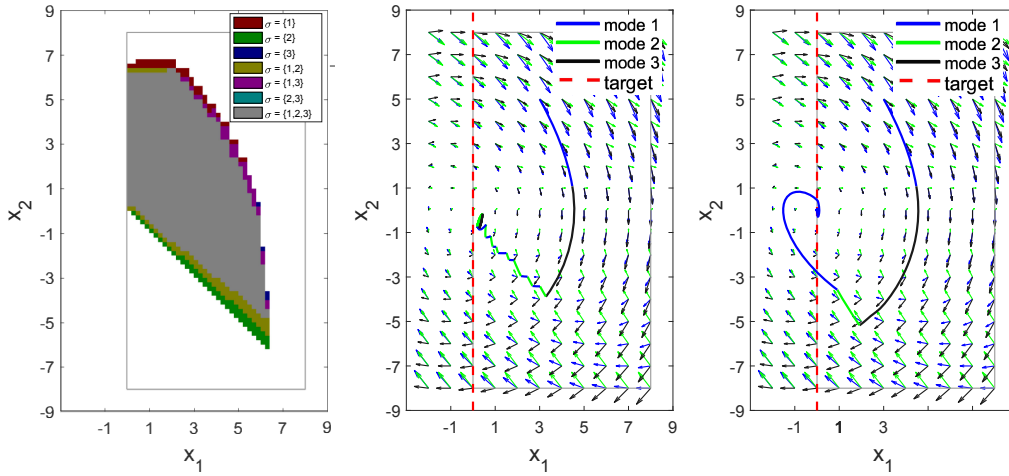
Fig. 3: Switching protocol for MPC supervision (left), phase portrait of closed-loop system: supervised (middle) and unsupervised (right).
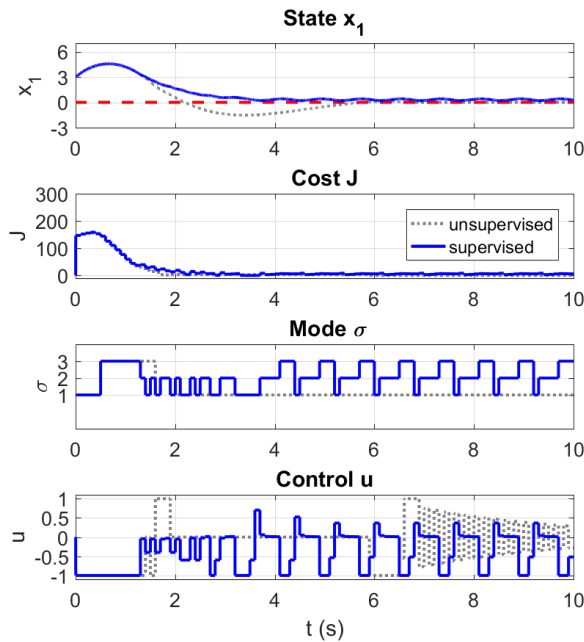

Fig. 4: Simulation: supervised MPC versus unsupervised MPC.

[8] I. I. CPLEX. V12. 1: Users manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[9] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The computer journal*, 8(3):250–255, 1965.

[10] H. J. Ferreau, S. Almér, H. Peyrl, J. L. Jerez, and A. Domahidi. Survey of industrial applications of embedded model predictive control. In *Control Conference (ECC), 2016 European*, pages 601–601. IEEE, 2016.

[11] D. Frick, A. Georghiou, J. L. Jerez, A. Domahidi, and M. Morari. Low-complexity method for hybrid mpc with local guarantees. *arXiv preprint arXiv:1609.02819*, 2016.

[12] I. Gurobi Optimization. Gurobi optimizer reference manual. *URL http://www. gurobi. com*, 2015.

[13] G. Hendel. New rounding and propagation heuristics for mixed integer programming. 2011.

[14] T. Kashima and S. P. Boyd. Cost optimal operation of thermal energy storage system with real-time prices. In *Control, Automation and Information Sciences (ICCAIS), 2013 International Conference on*, pages 233–237. IEEE, 2013.

[15] E. C. Kerrigan. *Robust constraint satisfaction: Invariant sets and predictive control*. PhD thesis, Citeseer, 2001.

[16] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[17] J. Lofberg. Yalmip: A toolbox for modeling and optimization in matlab. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 284–289. IEEE, 2004.

[18] J. Löfberg. Oops! I cannot do it again: Testing for recursive feasibility in mpc. *Automatica*, 48(3):550–555, 2012.

[19] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[20] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[21] T. Moor and J. Raisch. Abstraction based supervisory controller synthesis for high order monotone continuous systems. In *Modelling, Analysis, and Design of Hybrid Systems*, pages 247–265. Springer, 2002.

[22] V. V. Naik and A. Bemporad. Embedded mixed-integer quadratic optimization using accelerated dual gradient projection. *IFAC-PapersOnLine*, 50(1):10723–10728, 2017.

[23] P. Nilsson and N. Ozay. Incremental synthesis of switching protocols via abstraction refinement. In *Proc. of IEEE CDC*, pages 6246–6253, 2014.

[24] N. Ozay, J. Liu, P. Prabhakar, and R. Murray. Computing augmented finite transition systems to synthesize switching protocols for polynomial switched systems. In *Proc. of ACC*, pages 6237–6244, 2013.

[25] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[26] A. Richards and J. How. Mixed-integer programming for control. In *American Control Conference, 2005. Proceedings of the 2005*, pages 2676–2683. IEEE, 2005.

[27] N. Ricker, T. Subrahmanian, and T. Sim. Case studies of model-predictive control in pulp and paper production. In *Proceedings of the 1988 IFAC Workshop on Model Based Process Control*, pages 13–22, 1989.

[28] P. O. Scokaert and J. B. Rawlings. Feasibility issues in linear model predictive control. *AIChE Journal*, 45(8):1649–1659, 1999.

[29] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer, 2009.

[30] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad. A simple effective heuristic for embedded mixed-integer quadratic programming. *International Journal of Control*, pages 1–11, 2017.

[31] L. Yang, A. Karnik, B. Pence, M. T. B. Waez, and N. Ozay. Fuel cell thermal management: Modeling, specifications and correct-byconstruction control synthesis. In *Proceedings of American Control Conference*, 2017.

[32] A. Zheng and M. Morari. Stability of model predictive control with mixed constraints. *IEEE Transactions on Automatic Control*, 40(10):1818–1823, 1995.