MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

A Structure Exploiting Branch-and-Bound Algorithm for Mixed-Integer Model Predictive Control

Hespanhol, P.; Quirynen, R.; Di Cairano, S.

TR2019-050 June 29, 2019

Abstract

Mixed-integer model predictive control (MI-MPC) requires the solution of a mixed-integer quadratic program (MIQP) at each sampling instant under strict timing constraints, where part of the state and control variables can only assume a discrete set of values. Several applications in automotive, aerospace and hybrid systems are practical examples of how such discrete-valued variables arise. We utilize the sequential nature and the problem structure of MI-MPC in order to provide a branch-and-bound algorithm that can exploit not only the block-sparse optimal control structure of the problem but that can also be warm started by propagating information from branch-and-bound trees and solution paths at previous time steps. We illustrate the computational performance of the proposed algorithm and compare against current state-of-the-art solvers for a standard hybrid MPC case study, based on a preliminary implementation in MATLAB and C code.

European Control Conference (ECC)

A Structure Exploiting Branch-and-Bound Algorithm for Mixed-Integer Model Predictive Control

Pedro Hespanhol[1], Rien Quirynen[1], Stefano Di Cairano[1]

***Abstract*— Mixed-integer model predictive control (MI-MPC) requires the solution of a mixed-integer quadratic program (MIQP) at each sampling instant under strict timing constraints, where part of the state and control variables can only assume a discrete set of values. Several applications in automotive, aerospace and hybrid systems are practical examples of how such discrete-valued variables arise. We utilize the sequential nature and the problem structure of MI-MPC in order to provide a branch-and-bound algorithm that can exploit not only the block-sparse optimal control structure of the problem but that can also be warm started by propagating information from branch-and-bound trees and solution paths at previous time steps. We illustrate the computational performance of the proposed algorithm and compare against current state-of-the-art solvers for a standard hybrid MPC case study, based on a preliminary implementation in MATLAB and C code.**

I. INTRODUCTION

Optimization based control and estimation techniques, such as model predictive control (MPC) and moving horizon estimation (MHE), allow a model-based design framework in which the system dynamics and constraints can directly be taken into account [1]. This framework can be further extended to hybrid systems [2], providing a powerful technique to model a large range of problems, e.g., including dynamical systems with mode switchings or quantized control, problems with logic rules or obstacle avoidance constraints. However, the resulting optimization problems are highly non-convex because they contain variables that only take integer values. When using a quadratic objective in combination with linear system dynamics and linear inequality constraints, the resulting optimal control problem (OCP) can be formulated as a mixed-integer quadratic program (MIQP).

We aim to solve MIQP problems of the following form:

$$\min_{X,\,U} \quad \frac{1}{2}\sum_{i=0}^{N-1} x_i^\top Q_i x_i + u_i^\top R_i u_i + x_N^\top P x_N \tag{1a}$$

$$\text{s.t.} \quad x_0 - \hat{x}_0 \;=\; 0, \tag{1b}$$

$$A_i x_i + B_i u_i + a_i \;=\; x_{i+1}, \quad i \in \{0,\dots,N-1\}, \tag{1c}$$

$$l_i^c \le C_i x_i + D_i u_i \le u_i^c, \qquad i \in \{0,\dots,N-1\}, \tag{1d}$$

$$F_i u_i \in \{0,1\}, \qquad i \in \{0,\dots,N-1\}, \tag{1e}$$

$$l_N^c \le C_N x_N \le u_N^c, \tag{1f}$$

where the optimization variables are the state $X = [x_0^\top,\dots,x_N^\top]^\top$ and control trajectory $U = [u_0^\top,\dots,u_{N-1}^\top]^\top$. The set of constraints (1e) are binary equality constraints, since the left-hand side needs to be equal to either 0 or 1. For simplicity of notation, we further consider only binary control variables instead of more general integer constraints for an affine function of both state and control variables. MPC for several classes of hybrid systems can be straightforwardly formulated as in (1). Notable examples are mixed logical systems [2], where auxiliary continuous and discrete variables can be added to the input vector. Moreover, in combination with the binary constraints (1e), the affine inequalities (1d) can model various complicated but practical restrictions on the feasible region, such as obstacle avoidance and disjoint polyhedral constraints for states and inputs.

A hybrid MPC controller aims to solve the MIQP (1) at every sampling time instant. This is a difficult task, given that mixed-integer programming is $\mathcal{NP}$-hard in general, and several methods for solving such a sequence of MIQPs have been explored in the literature. These approaches can be divided into heuristic techniques, which seek to efficiently find sub-optimal solutions to the problem, and optimization algorithms which attempt to solve the MIQPs to optimality. Examples of the former include rounding and pumping schemes [3], approximate optimization algorithms [4], and approximate dynamic programming [5]. The downside of fast heuristic approaches is often the lack of guarantees for finding an optimal or even an integer-feasible solution.

As for solving these problems to optimality, most of the optimization algorithms for MIQPs are based on the classical branch-and-bound (B&B) technique [6]. For the purpose of mixed-integer MPC, the standard B&B strategy has been combined with various methods for solving the relaxed convex QPs. For example, a B&B algorithm for mixed-integer MPC (MI-MPC) has been proposed in combination with a dual active-set solver in [7], with an interior point algorithm in [8], dual projected gradient methods in [4], a nonnegative least squares solver in [9], and the alternating direction method of multipliers (ADMM) in [10].

Another important research topic focuses on general pre-processing and modeling techniques to reduce the size and strengthen the mixed-integer problem formulations [11]. These *presolve* techniques are vital to the good performance of current state-of-the-art mixed-integer solvers [12], such that these methods can often solve seemingly intractable problems in practice. The branch-and-bound method itself has been extensively studied with several improvements in branching and variable selection techniques [13], [14]. Finally, the branch-and-bound strategy has been generalized further, e.g., using cutting planes to tighten the convex problem relaxations, resulting in *branch-and-cut* or *branch-*

[1]Control and Dynamical Systems, Mitsubishi Electric Research Laboratories, Cambridge, MA, 02139, USA. quirynen@merl.com

and-price variants of the algorithm [6], [11]. Unlike state-of-the-art mixed-integer solvers, e.g., GUROBI [15] and MOSEK [16], our aim is to propose a tailored algorithm and its solver implementation for fast embedded MI-MPC applications, i.e., running on microprocessors with considerably less computational resources and available memory. The optimization algorithm should be relatively simple to code with a moderate use of resources, while the software implementation is preferably compact and library independent.

In this paper, our first contribution is to propose a branch-and-bound based MPC algorithm, which exploits the features of a recently proposed structure-exploiting primal active-set solver called PRESAS [17]. The latter algorithm is tailored to efficiently solve QPs with a block-sparse optimal control structure. Our second contribution is to bring various mixed-integer programming techniques, such as bound strengthening, domain propagation, and advanced branching rules, to the context of MI-MPC. In particular, we present an algorithm that exploits the sequential nature of MPC, in order to warm-start the branch-and-bound search tree and to re-use information gathered at previous time steps. Finally, the computational performance of the proposed algorithm, for a preliminary implementation in MATLAB and C code, is illustrated and compared against current state-of-the-art solvers for a standard hybrid MPC case study.

II. MIXED-INTEGER QUADRATIC PROGRAMMING

We first introduce some of the basic concepts in mixed-integer programming based on branch-and-bound methods, such as convex relaxations and branching strategies.

A. Convex Quadratic Program Relaxations

A standard approach to solve the MIQP (1) is to create convex relaxations of this problem and then solve the relaxations in order to approach the solution to the original MIQP. A straightforward idea is to obtain convex QP relaxations by dropping the binary equality constraints (1e) and instead enforcing the affine inequality constraints $0 \leq F_i u_i \leq 1$. Other convex relaxations for MIQPs have been studied in the literature such as moment or SDP relaxations that are often tighter than QP relaxations [18], but they can be relatively expensive to solve for larger problems.

For the purpose of this paper, we will focus our attention on QP relaxations where we allow the binary variables to take on real values. The main reason for choosing this relaxation is that we utilize a tailored structure exploiting active-set solver, called PRESAS [17], proposed recently for efficiently solving the convex QP relaxations. The latter solver has been shown to be competitive with state-of-the-art QP solvers for embedded MPC, and it benefits strongly from warm-starting, which can be exploited when solving the sequence of QPs within the branch-and-bound strategy. Note that the relaxations need to be convex, i.e., the weight matrices Q_i, R_i and P need to be positive (semi-) definite in (1a) such that each solution to a QP relaxation is globally optimal.

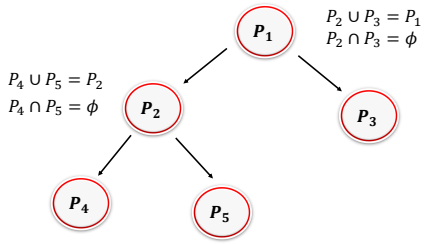

Fig. 1: Illustration of the branch-and-bound method as a binary search tree. A selected node can be either *branched*, resulting in two partitions for each bound value in (1e), or *pruned* based on feasibility or the current upper bound.

B. Branch-and-Bound Algorithm

The main idea of the branch-and-bound (B&B) algorithm is to sequentially create partitions of the original problem and then attempt to solve those partitions. While solving each partition may still be challenging, it is fairly efficient to obtain local lower bounds on the optimal objective value, by solving relaxations of the mixed-integer program or by using duality. If we happen to obtain an integer-feasible solution while solving a relaxation, we can then use it to obtain a global upper bound for the solution to the original problem. This may help to avoid solving or branching certain partitions that were already created, i.e., these partitions or nodes can be *pruned*. The general algorithmic idea of partitioning is better illustrated as a binary search tree, see Figure 1.

A key step in this approach is how to create the partitions, i.e., which node to choose and which binary variable to select for branching. Since we solve a QP relaxation at every node of the tree, it is natural to branch on one of the binary variables with fractional values in the optimal solution of the QP relaxation. Therefore, if a variable, e.g., $u_{i,k} \in \{0, 1\}$ has a fractional value in a given QP relaxation, then we create two partitions where we respectively add the equality constraint $u_{i,k} = 0$ and $u_{i,k} = 1$. Another key step is how to choose the order in which the created subproblems are solved. These two steps have been extensively explored in the literature and various heuristics are implemented in state-of-the-art tools [13]. We provide next a brief description of strategies that we implemented in our B&B solver.

C. Tree Search: Node Selection Strategies

A common implementation of the branch-and-bound method is based on a *depth-first* node selection strategy, which can be readily implemented using a last-in-first-out (LIFO) buffer. The next node to be solved is selected as one of the children of the current node and this process is repeated until a node is pruned, i.e., the node is either infeasible, optimal or dominated by the upper bound, which is followed by a backtracking procedure. Instead, a *best-first* strategy selects the node with the lowest local lower bound so far. In what follows, we will employ a combination of the depth-first and best-first node selection approach. This idea is motivated by aiming to find an integer-feasible

solution quickly at the start of the branch-and-bound procedure (depth-first) to allow for early pruning, followed by a more greedy search for better feasible solutions (best-first).

D. Reliability Branching for Variable Selection

The idea of *reliability branching* is to combine two powerful concepts for variable selection: strong branching and pseudo-costs [13]. Strong branching relies on temporarily branching, both up (to higher integer) and down (to lower integer), for every binary variable that has a fractional value in the solution of a QP relaxation in a given node, before committing to the variable that provides the highest value for a particular score function. The increase in objective values $\Delta^+_{i,k}$, $\Delta^-_{i,k}$ are computed when branching the binary variable $u_{i,k}$, respectively, up and down. Given these quantities, a simple scoring function $\text{score}(\cdot,\cdot)$ is computed for each binary variable. For instance, based on the product [14]:

$$S_{i,k} = \text{score}(\Delta^-_{i,k}, \Delta^+_{i,k}) = \max(\Delta^+_{i,k}, \epsilon) \cdot \max(\Delta^-_{i,k}, \epsilon), \tag{2}$$

given a small positive value $\epsilon > 0$. This branching rule has been empirically shown to provide smaller search trees in practice [13]. The downside is that this procedure is relatively expensive since several QP relaxations are solved in order to select one variable to branch on.

The idea of pseudo-costs aims at approximating the increase of the objective function to decide which variable to branch on, without having to solve additional QP relaxations. This can be done by keeping statistic information for each binary variable, i.e., the *pseudo-costs* that represent the average increase in the objective value per unit change in that particular binary variable when branching. Every time that a given variable is chosen to be branched on, and the resulting relaxation is feasible, then we update each corresponding pseudo-cost with the observed increase in the objective, divided by the distance of the real to the binary value, in the form of a cumulative average. Therefore, each variable has two pseudo-costs, $\phi^-_{i,k}$ when the variable was branched "down" and $\phi^+_{i,k}$ when it was branched "up". Given the solution to a QP relaxation, one can then use the pseudo-costs to select the binary variable with the highest score value to be branched on next:

$$S_{i,k} = \text{score}(\bar{u}_{i,k}\,\phi^-_{i,k},\ (1-\bar{u}_{i,k})\,\phi^+_{i,k}), \tag{3}$$

given a fractional value $\bar{u}_{i,k}$ in the QP relaxation.

This way, we select variables based on their past behavior throughout the branch-and-bound tree. However, at the beginning of the algorithm, the pseudo-costs are not yet initialized, which is when branching decisions typically impact the tree size the most. *Reliability branching* uses strong branching to initialize the pseudo-costs until a certain condition of reliability is satisfied, e.g., one switches to using pseudo-costs only once that particular variable has been branched on a specified number η_{rel} of times [13]. The resulting branching rule is summarized in Algorithm 1. Note that reliability branching coincides with pseudo-cost branching if $\eta_{rel} = 0$, with strong branching if $\eta_{rel} = \infty$, but typically a value $1 \leq \eta_{rel} \leq 4$ is chosen.

Algorithm 1 Reliability Branching Strategy

Input: η_{rel}, set C of candidate variables for branching.

for candidate variables $u_{i,k}$ in C **do**
  if $\#\texttt{branch}(u_{i,k}) \leq \eta_{\text{rel}}$ **then**
    Strong branching on $u_{i,k}$ to compute score $S_{i,k}$.
    Update pseudo-costs $\phi^-_{i,k}$ and $\phi^+_{i,k}$.
  else
    $S_{i,k} = \text{score}(\bar{u}_{i,k}\,\phi^-_{i,k},\ (1-\bar{u}_{i,k})\,\phi^+_{i,k})$.
  end if
end for

Output: Select variable with highest score $S^* = \max_{i,k} S_{i,k}$.

III. PRESOLVE TECHNIQUES FOR MIXED-INTEGER OPTIMAL CONTROL

As mentioned earlier, presolve techniques are often crucial in making convex relaxations tighter such that typically fewer nodes need to be explored, sometimes to such an extent that seemingly intractable problems become tractable.

A. Domain Propagation for Condensed QP Subproblem

Several strengthening techniques are implemented as part of "presolve" routines in commercial solvers [12]. One particular technique that is suitable to mixed-integer optimal control is based on *domain propagation*, in which the goal is to strengthen bound values based on the inequality constraints (1d)-(1f) in the problem. However, the results of such a strategy are rather weak when directly applied to the block-sparse QP in (1), because the stage-wise coupling of the state variables (1c) needs to be taken into account. Therefore, we use instead the equivalent dense QP formulation in which the state variables are numerically eliminated, such that stronger bounds can be obtained for the control variables.

Let us concatenate all state variables in a vector X and all control variables in the vector U, such that Eqs. (1b)-(1c) can be written more compactly as

$$\bar{A}X = \bar{B}U + b + E_0\hat{x}_0, \tag{4}$$

where we define the block-sparse matrices

$$\bar{A} = \begin{bmatrix} I & & & \\ -A_1 & I & & \\ & \ddots & \ddots & \\ & & -A_{N-1} & I \end{bmatrix}, \tag{5a}$$

$$\bar{B} = \text{blkdiag}(B_0, \ldots, B_{N-1})\ ,\ E_0 = [A_0^\top, 0, \ldots, 0]^\top. \tag{5b}$$

The matrix $\bar{A}$ is invertible such that we can write:

$$X = \bar{A}^{-1}\bar{B}U + \bar{A}^{-1}(b + E_0\hat{x}_0). \tag{6}$$

Now, we can substitute the latter expression for the state vector in OCP (1) to obtain the condensed form

$$\min_U \quad \frac{1}{2}U^\top H_c U + h_c^\top U \tag{7a}$$

$$\text{s.t.} \quad \bar{l}^c \leq D_c U \leq \bar{u}^c \tag{7b}$$

$$F_i u_i \in \{0,1\}, \qquad i \in \{0, \ldots, N-1\}, \tag{7c}$$

where the condensed matrices and vectors read as

$$H_c = (\bar{A}^{-1}\bar{B})^\top Q\bar{A}^{-1}\bar{B} + R,\ D_c = C\bar{A}^{-1}\bar{B} + D, \quad (8a)$$

$$h_c = (\bar{A}^{-1}\bar{b})^\top Q\bar{A}^{-1}\bar{B}, \quad (8b)$$

$$\bar{l}^c = l^c - C\bar{A}^{-1}\bar{b}, \quad \bar{u}^c = u^c - C\bar{A}^{-1}\bar{b}, \quad (8c)$$

where $\bar{b} := b + E_0\hat{x}_0$ is defined and given $Q = \text{blkdiag}(Q_1, \dots, Q_{N-1}, P)$, $R = \text{blkdiag}(R_0, \dots, R_{N-1})$, and $l^c = [l_1^\top, \dots, l_N^\top]^\top$ and $u^c = [u_1^\top, \dots, u_N^\top]^\top$.

Given the condensed problem formulation, which can be computed offline and which is parametric in the current state value $\hat{x}_0$, we can then apply the following bound strengthening procedure, which is explained next for a single affine constraint $l_b \leq \sum_i d_i u_i \leq u_b$ in (7b). This constraint can be used to try and tighten bound values for all control variables u_i for which $d_i \neq 0$, where u_i denotes a single control variable in the vector U. Let $\bar{u}_i, \underline{u}_i$ be the current upper/lower bounds for u_i such that

$$d_i u_i \leq u_b - \sum_{j\neq i} d_j u_j \leq \underbrace{u_b - \sum_{j\neq i, d_j>0} d_j \underline{u}_j - \sum_{j\neq i, d_j<0} d_j \bar{u}_j}_{=:\bar{u}_{b,i}}, \quad (9)$$

in which we divide by d_i in order to obtain

$$u_i \leq \frac{\bar{u}_{b,i}}{d_i},\ \text{if } d_i > 0 \quad \text{or} \quad u_i \geq \frac{\bar{u}_{b,i}}{d_i},\ \text{if } d_i < 0. \quad (10)$$

This results, respectively, in the updated bound values

$$\bar{u}_i = \min\left(\bar{u}_i, \frac{\bar{u}_{b,i}}{d_i}\right), \quad \text{or} \quad \underline{u}_i = \max\left(\underline{u}_i, \frac{\bar{u}_{b,i}}{d_i}\right), \quad (11)$$

or, in case u_i is an integer or binary variable,

$$\bar{u}_i = \min\left(\bar{u}_i, \left\lfloor \frac{\bar{u}_{b,i}}{d_i} \right\rfloor\right), \quad \text{or} \quad \underline{u}_i = \max\left(\underline{u}_i, \left\lceil \frac{\bar{u}_{b,i}}{d_i} \right\rceil\right). \quad (12)$$

where $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ are the floor and ceiling operations, respectively. Thus, this can result in strengthening of bound values for both continuous and integer/binary control variables. The procedure can be executed for each control variable and each inequality constraint in an iterative manner, see Algorithm 2, since bound strengthening for one variable can lead to strengthening for other variables [12]. The process is typically stopped when the bound values do not sufficiently change or a certain limit on the computation time is met.

Domain propagation can lead to considerable reductions in the amount of explored nodes, e.g., because variables are fixed, when $\bar{u}_i = \underline{u}_i$, or because of infeasibility detection, when $\bar{u}_i < \underline{u}_i$, without the need to solve any QP relaxations. In addition, the updated bound values for all control variables can be used to strengthen QP relaxations in the future. Lastly, we can use domain propagation in order to improve and generalize Hessian-based fixing strategies, such as the one proposed in [19]. Hessian-based fixing typically can only be applied to unconstrained problems, since it fixes the variables solely based on the objective. Here, we propose to use domain propagation to compute the feasibility impact of certain variable fixings. More specifically, a particular variable can be fixed based on optimality, if and only if this fixing does not induce feasibility-based fixings.

Algorithm 2 Domain Propagation for Bound Strengthening

Input: Inequality constraints (7b), variable bounds $\bar{u}_i, \underline{u}_i$.

while stopping criterion == False **do**
  for every row of D_c **do**
    for every $u_i \in U,\ d_i \neq 0$ **do**
      Obtain bound values $\bar{u}_{b,i}, \bar{l}_{b,i}$ using Eq. (10).
      Update variable bounds using (11) or (12).
    end for
  end for
end while

Output: Updated bounds $\bar{u}_i, \underline{u}_i$ for all control variables.

Algorithm 3 B&B Method for the MIQP-OCP in (1)

Input: Upper bound UB, tolerance ϵ.

LB=$-\infty$ and initialize $L = \{P_0\}$ with root node.
Select current node $P_c \leftarrow P_0$.
while $UB - LB > \epsilon$ **do**
  Apply domain propagation to P_c using Alg. 2.
  Solve resulting QP relaxation with `PRESAS`.
  if QP is feasible and $J(\bar{X}, \bar{U}) \leq$ UB **then**
    if QP solution is not integer-feasible **then**
      $LB \leftarrow \min_{P\in L} J(P)$.
      Select branching variable v using Alg. 1.
      Create subproblems P_u "up" and P_l "down".
      Append $\{P_l, P_u\}$ to L if $(1-\bar{v})\phi_v^+ < \bar{v}\phi_v^-$ or append $\{P_u, P_l\}$ to L, otherwise.
    else
      $UB \leftarrow J(\bar{X}, \bar{U})$ and $(X^*, U^*) \leftarrow (\bar{X}, \bar{U})$.
    end if
  end if
  Remove current node P_c from to-do list in L.
  Select next node based on depth-first (last node in list L) or based on best lower bound.
end while

Output: MIQP solution vector (X^*, U^*).

B. Resulting MIQP Algorithm for Optimal Control

Algorithm 3 describes the most important steps in our proposed B&B method for solving the MIQP in (1). It solves a block-structured QP relaxation using `PRESAS` [17] at every node and utilizes reliability branching (Algorithm 1) to decide the branching variables. As discussed earlier, the node selection strategy is based on a depth-first search followed by a best-first search as soon as an integer-feasible solution has been found. Note that the upper bound value UB provided to Alg. 3 can be based on an integer-feasible solution guess or it can initially be set to $+\infty$.

IV. MIXED-INTEGER MPC ALGORITHM

In embedded applications of mixed-integer MPC, one needs to solve an MIQP (1) at each sampling instant under strict timing constraints. We can leverage the fact that we solve a sequence of similar problems in order to warm start the B&B optimization algorithm.

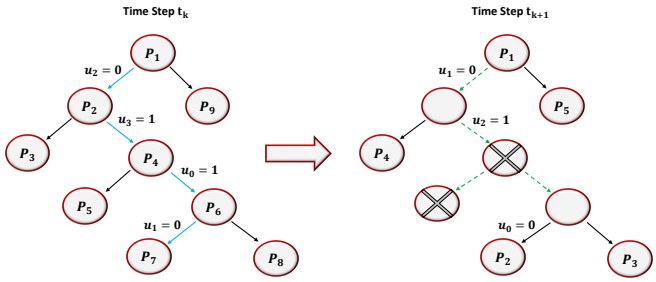

Fig. 2: Illustration of the tree propagation technique from one time point to the next in the MI-MPC algorithm: index i denotes the order in which each node P_i is solved.

A. Warm Starting based on Tree Propagation

The warm-starting procedure aims to use knowledge of one MIQP, i.e., the search tree after solving the problem, in order to improve the B&B search for the next MIQP. Our idea is to store the path from the root to the leaf node where the optimal solution to the MIQP was found, as well as the branching order of the variables. We can then perform a shifting of this path in order to obtain a "warm-started tree" to start our search to solve the MIQP at the next time step. We illustrate this procedure in Figure 2, where the optimal path at the current time step is denoted by the sequence of nodes $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_6 \rightarrow P_7$. Let us consider a corresponding sequence of variables $u_2 \rightarrow u_3 \rightarrow u_0 \rightarrow u_1$ that we branched on in order to create such optimal path. After shifting by one time step, all branched variables in the first control interval can be ignored, e.g., resulting in a shifted and shorter path of variables $u_1 \rightarrow u_2 \rightarrow u_0$.

At the subsequent time step, after obtaining the new state estimate, we execute all presolving techniques and we solve the QP relaxation corresponding to the root node. After removing from the warm-started tree the nodes that correspond to branched variables which are already integer feasible in the relaxed solution at the root node, we proceed by solving all the leaf nodes on the warm-started path. As we solve both children of a node on this path, we do not have to solve the parent node itself and therefore reduce computations by solving less QP relaxations. Hence, we go over the tree in the order depicted by the index of each node in Fig. 2. After the warm-started branch has been explored, we resume normal procedure of the B&B method. Algorithm 4 summarizes the proposed *tree propagation* technique.

The sequential nature of the problem also allows to shift and re-use the pseudo-cost information from one MPC time step to the next. This idea has the potential of producing smaller search trees as the MPC progresses, without the need to perform strong branching at every MPC step. The propagation of pseudo-costs can be coupled with an update of the reliability parameters to improve the overall performance. For example, the reliability number should be reduced for each variable from one time step to the next, in order to force strong branching for variables that have not been branched on in a sufficiently long time. In addition, nodes can be removed from the warm-started path in case they correspond to branched variables for which there is no pseudo-cost information or it is not sufficiently reliable, in an attempt to avoid bad branching decisions. Finally, these warm-started pseudo-costs can also be used to re-order the warm-started tree, in order to result in smaller search tree sizes.

Algorithm 4 Tree Propagation for Warm-Started B&B

Input: Optimal path P from root to leaf node.

1: Shift index of branched variables by 1 stage along path.
2: Solve root node of shifted path P, including presolve.
3: **for** (branched variables on stage -1 after shifting)
   || (variables are integer feasible in root node)
   || (variables without pseudo-costs) **do**
4: remove associated node from the path P.
5: **end for**
6: Re-order sequence of branched variables by scoring based on warm-started pseudo-cost information.
7: Initialize the B&B tree along the shifted path P, creating nodes along the path and their respective children.
8: Create the warm-started list L, excluding parent nodes.

Output: Warm-started tree for next MIQP, given by list L.

The proposed tree propagation technique, with the additional re-use of pseudo-cost information, has been summarized in Algorithm 4. This procedure can improve the overall performance of the B&B method in multiple ways. First of all, the optimal path and pseudo-cost information is re-used to make better branching decisions for the mixed-integer program at the next time step, because the search trees are often similar for two subsequent problems. Also, the computational cost can be reduced by solving less QP relaxations to explore the warm-started tree. In addition, the shifted optimal path can be used in an attempt to efficiently obtain an integer-feasible solution, and therefore an important upper bound in the B&B algorithm, for the MPC problem at the next time step.

B. MI-MPC Algorithm Implementation

The proposed MI-MPC algorithm solves a sequence of MIQPs where the branch-and-bound tree is warm-started at every time step, as well as the pseudo-cost and QP condensing information. The B&B strategy and the presolve, warm-start and heuristic branching techniques have been implemented in MATLAB, based on a C code implementation of the `PRESAS` algorithm [17] to solve each QP relaxation.

V. HYBRID MPC: BENCHMARK EXAMPLE

We consider a hybrid MPC problem from [2], with the default settings as in `bm99sim.m`, which is a part of the Hybrid Toolbox for MATLAB. This demo example has been used also more recently for numerical comparisons in [9]. The system is modeled using the HYSDEL toolbox [20] to obtain the mixed logical dynamical (MLD) system formulation. Let us compare our algorithm with the state-of-the-art `GUROBI` [15] and `MOSEK` [16] solvers for mixed-integer programming. Figure 3 illustrates the average and worst-case

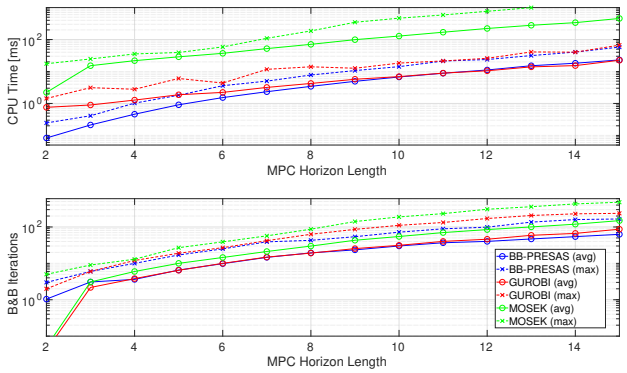

Fig. 3: Computational results for closed-loop mixed-integer MPC of the `bm99` example: `BB-PRESAS` versus `GUROBI` and `MOSEK` solvers for varying control horizon length N.

CPU times taken by our algorithm, `GUROBI` and `MOSEK` for a range of control horizon lengths N. All advanced presolve and heuristic options have been activated for both software tools, resulting in fair computational comparisons.

Table I presents a detailed comparison for this test example, including additional timing results for the `MI-NNLS` solver that are taken directly from [9]. The latter computational results can serve only as a reference since they have been obtained on a different computer, with respect to the one used here with a 2.80 GHz Intel Xeon E3-1505M v5 processor and 32 GB of RAM. An important feature of our method is that its worst-case computation time is often rather close to the average performance in closed-loop MI-MPC simulations. This highlights the effectiveness of our tree propagation warm-starting procedure, such that consecutive branch-and-bound trees have approximately the same size. In addition, it can be observed from Table I that our proposed `BB-PRESAS` solver is either competitive with, or is a factor 2 or 3 times faster than `GUROBI`. The computational speedup is much larger when compared with other state-of-the-art tools such as `MOSEK`, our solver can be more than 10 times faster in this particular MI-MPC test example. It shall be noted that `GUROBI` is a heavily optimized and fairly large software, which is unlikely to be amenable for embedded microprocessors, due to its code size, memory requirements, and software library dependencies.

VI. CONCLUSIONS & OUTLOOK

In this paper, we proposed a branch-and-bound algorithm for mixed-integer MPC that exploits the optimal control problem structure to strengthen variable bounds, re-use pseudo-costs and warm-start the search tree at every MPC time step. More specifically, tailored domain propagation and tree propagation strategies have been presented. We showed preliminary results that illustrate the computational performance of our algorithm for a hybrid MPC case study. A compact, efficient, but self-contained C code implementation of the proposed algorithm is under development to enable real-time embedded applications of hybrid MPC.

TABLE I: Timing results (ms) per sampling step of hybrid MPC test problem for different horizon lengths N. Computation times for `MI-NNLS` solver are taken directly from [9].

N	`BB-PRESAS` (mean/max)	`GUROBI` (mean/max)	`MOSEK` (mean/max)	`MI-NNLS` (mean/max)
2	0.1/0.2	0.7/1.4	2.1/4.0	2.0/2.6
3	0.2/0.3	1.0/2.3	15.1/24.7	2.5/4.8
4	0.4/0.9	1.7/4.6	21.7/35.5	3.1/6.9
5	0.9/1.7	2.5/4.9	28.7/39.3	3.9/13.0
6	1.5/3.5	3.2/7.5	36.8/58.8	5.1/18.3
7	2.3/4.9	4.0/6.9	51.8/109.3	6.4/30.2
8	3.5/7.6	5.1/10.0	70.4/185.8	8.1/43.4
9	5.1/10.3	6.6/12.5	98.7/347.1	11.1/69.8
10	6.8/14.3	8.4/16.1	126.7/465.3	14.4/103.2
11	8.8/22.1	9.8/17.2	168.2/587.8	20.6/179.1
12	11.3/23.7	11.6/20.5	219.2/765.0	26.9/263.4
13	15.0/31.6	14.3/29.5	276.3/996.0	35.5/384.9
14	17.8/35.1	16.4/44.6	334.1/1241.9	46.3/562.4
15	21.0/41.6	21.9/71.6	450.8/1606.8	61.7/766.9

REFERENCES

[1] D. Mayne and J. Rawlings, *Model Predictive Control*. Nob Hill, 2013.

[2] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, pp. 407–427, 1999.

[3] T. Achterberg and T. Berthold, "Improving the feasibility pump," *Discrete Optimization*, vol. 4, no. 1, pp. 77–86, 2007.

[4] V. V. Naik and A. Bemporad, "Embedded mixed-integer quadratic optimization using accelerated dual gradient projection," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 10 723–10 728, 2017.

[5] B. Stellato, T. Geyer, and P. J. Goulart, "High-speed finite control set model predictive control for power electronics," *IEEE Transactions on Power Electronics*, vol. 32, no. 5, pp. 4007–4020, May 2017.

[6] C. A. Floudas, *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, 1995.

[7] D. Axehill and A. Hansson, "A mixed integer dual quadratic programming algorithm tailored for MPC," in *Decision and Control, 2006 45th IEEE Conference on*. IEEE, 2006, pp. 5693–5698.

[8] D. Frick, A. Domahidi, and M. Morari, "Embedded optimization for mixed logical dynamical systems," *Computers & Chemical Engineering*, vol. 72, pp. 21–33, 2015.

[9] A. Bemporad and V. V. Naik, "A numerically robust mixed-integer quadratic programming solver for embedded hybrid model predictive control," in *Proc. 6th IFAC NMPC Conf.*, Madison, USA, 2018.

[10] B. Stellato, V. Naik, A. Bemporad, and P. Goulart, "Embedded mixed-integer quadratic optimization using the OSQP solver," in *European Control Conference*, 2018.

[11] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. New York, NY, USA: Wiley-Interscience, 1988.

[12] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger, "Presolve reductions in mixed integer programming," *ZIB Report*, pp. 16–44, 2016.

[13] T. Achterberg, T. Koch, and A. Martin, "Branching rules revisited," *Operations Research Letters*, vol. 33, no. 1, pp. 42–54, 2005.

[14] P. Le Bodic and G. Nemhauser, "An abstract model for branching and its application to mixed integer programming," *Mathematical Programming*, vol. 166, no. 1-2, pp. 369–405, 2017.

[15] L. Gurobi Optimization, "Gurobi optimizer reference manual," " 2018.

[16] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual.*, 2017.

[17] R. Quirynen, A. Knyazev, and S. Di Cairano, "Block structured preconditioning within an active-set method for real-time optimal control," in *Proc. European Control Conference (ECC)*, 2018.

[18] D. Axehill, L. Vandenberghe, and A. Hansson, "Convex relaxations for mixed integer predictive control," *Automatica*, vol. 46, no. 9, pp. 1540 – 1545, 2010.

[19] D. Axehill and A. Hansson, "A preprocessing algorithm for MIQP solvers with applications to MPC," in *CDC*, 2004, pp. 2497–2502.

[20] F. D. Torrisi and A. Bemporad, "HYSDEL-a tool for generating computational hybrid models for analysis and synthesis problems," *IEEE trans. control sys. techn.*, vol. 12, no. 2, pp. 235–249, 2004.