IEEE Control Systems Letters paper presented at
2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Learning for Online Mixed-Integer Model Predictive Control with Parametric Optimality Certificates

Luigi Russo*, Siddharth H. Nair*, Luigi Glielmo, Francesco Borrelli

***Abstract*— We propose a supervised learning framework for computing solutions of multi-parametric Mixed Integer Linear Programs (MILPs) that arise in Model Predictive Control. Our approach also quantifies sub-optimality for the computed solutions. Inspired by Branch-and-Bound techniques, the key idea is to train a Neural Network or Random Forest which, for a given parameter, predicts a strategy consisting of (1) a set of Linear Programs (LPs) such that their feasible sets form a partition of the feasible set of the MILP and (2) an integer solution. For control computation and sub-optimality quantification, we solve a set of LPs online in parallel. We demonstrate our approach for a motion planning example and compare against various commercial and open-source mixed-integer programming solvers.**

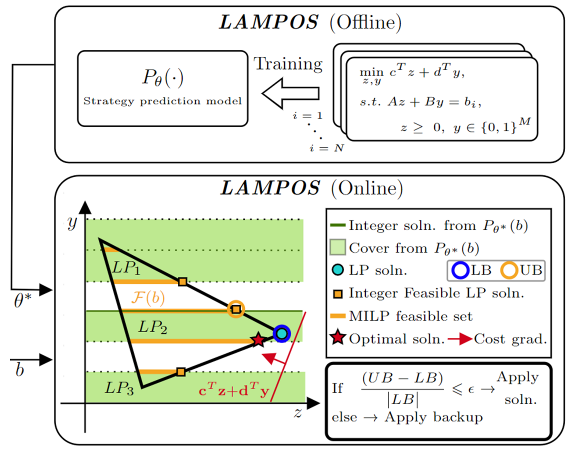

Fig. 1. We propose LAMPOS, a strategy-based solution approach for mp-MILPs for real-time MPC. Offline, a prediction model $P_\theta(\cdot)$ is trained on various MILP instances to learn a strategy $s(\cdot)$, mapping parameters b to an optimal integer solution and a set of LPs (called a cover) obtained from the leaves of the BnB tree. Online, a solution to the MILP is obtained from the predicted strategy $s(b)$ by solving a set of LPs in parallel. The proposed strategy allows (1) sub-optimality quantification of MILP feasible solutions, and (2) recovery of MILP solution if none were found from the LPs.

I. Introduction

Multi-parametric Mixed-Integer Programming (mp-MIP) is a convenient framework for modelling various non-convex motion planning and constrained optimal control problems [1]. The mixed-integer formulation can model constraints such as collision avoidance [2], mixed-logical specifications [3] and mode transitions for hybrid dynamics [4]. The multi-parametric nature of these mp-MIPs arises from requiring to solve these problems for different initial conditions, obstacles configurations or system constraints—all of which affect the MIP solution. When Model Predictive Control (MPC) [5], [6] is used for such class of problems, an MIP has to be solved in a receding horizon fashion at each time step. However, computing solutions for MIPs is $\mathcal{NP}-$hard and challenging for real-time ($\geq$ 10Hz) applications.

There are two broad approaches towards solving these MIPs online for real-time MPC. The first approach is Explicit MPC [6], [7] which involves offline computation of the solution map of the mp-MIP explicitly as piece-wise functions over partitions of the parameter space, so that online computation is reduced to a look-up. However this approach is best suited for mp-MIPs of moderate size because the complexity of the online look-up and offline storage of partitions, increases rapidly with scale [8]. The second approach for real-time mixed-integer MPC relies on predicting warm-starts for the mp-MIP by training Machine Learning (ML) models on large offline datasets [9], [10], [11], [12]. The authors of [9], [10] use various supervised learning frameworks to predict the optimal integer variables for the mp-MIP at a given parameter so that the online computation is reduced to solving a convex program.

In [11], [12], the authors define the notion of an *optimal strategy* for a mp-MIP as a mapping from parameters to the complete information required to efficiently recover an optimal solution. For multi-parametric Mixed-Integer Linear/Quadratic Programs (mp-MILPs/MIQPs), an optimal strategy is defined as *a set of integer variables and active constraints at the optimal solution*. Given an optimal strategy, an optimal solution can be recovered by solving a linear system of equations which is computationally inexpensive compared to tree search methods typically used for solving MIPs, such as Branch-and-Bound (BnB). Thus, a prediction model is trained offline to predict the optimal strategy for efficiently solving the MIPs online. However, a common issue that plagues these ML-based approaches is the inability to assess the quality of the predicted warm-start/strategy to guard against poor predictions, which can lead to sub-optimal or infeasible solution predictions. Indeed, prediction models may perform poorly for various reasons: insufficient richness of the model parameterization, distribution shift between the training and test sets, convergence of the training algorithm towards a sub-optimal minimum [13].

In this work, we focus on mp-MILPs and propose a supervised learning framework for predicting strategies to efficiently solve the MILP, along with a mechanism to measure the sub-optimality of the prediction. The authors

SHN and LR contributed equally. SHN and FB are with the Model Predictive Control lab, University of California, Berkeley, USA. LR is with the Dept. of Engineering, Università del Sannio, Benevento, Italy. LG is with the Dept. of Electrical Engineering and Information Technologies, Università di Napoli Federico II, Italy. Corresponding author: siddharth_nair@berkeley.edu

of [14] propose a framework for certifying the quality of predicted solutions for parametric convex Quadratic Programs (QPs) using strong duality. The main idea is to train prediction modules offline that predict optimal solutions for both the primal QP and its Lagrangian dual. For primal and dual feasible predictions (after projection to the feasible set if necessary), the quality of the predictions can be assessed from the duality gap because of strong duality. Since strong duality does not hold in general in the framework of Lagrangian duality for MILPs, we do not adopt this approach. However we draw inspiration from [14] and the optimality certification procedure in Branch-and-Bound, to propose a strategy definition for mp-MILPs, accompanied by bounding functions to quantify the sub-optimality of the strategy. This enables us to efficiently recover the solution of an MILP online from the predicted strategy by solving some LPs online in parallel, and also measure the sub-optimality of the recovered solution. Using ideas from multi-parametric programming, we show the parametric behaviour of our proposed strategy definition. We complement this insight with a supervised learning framework for training a prediction model offline, for predicting strategies online.

II. Problem Formulation

Consider the general formulation for Mixed-Integer MPC (MIMPC) adapted from [6]:

$$\begin{aligned} V^\star(x_t) = \min_{\substack{\boldsymbol{x}_t, \boldsymbol{u}_t,\\ \boldsymbol{\delta}_t, \boldsymbol{z}_t}} \ & ||Px_{t+N|t}||_p + \sum_{k=t}^{t+N} ||Q\begin{bmatrix} x_{k|t} \\ \delta_{k|t}\end{bmatrix}||_p + ||Ru_{k|t}||_p, \\ \text{s.t. } & x_{k+1|t} = Ax_{k|t} + B_1u_{k|t} + B_2\delta_{k|t} + B_3z_{k|t}, \\ & E_2\delta_{k|t} + E_3z_{k|t} \leq E_1u_{k|t} + E_4x_{k|t} + E_5, \\ & x_{t|t} = x_t, \ \delta_{k|t} \in \{0,1\}^{n_\delta} \ \forall k = t,..,t+N-1 \end{aligned} \tag{1}$$

where x_t is system state at time t, $p = 1$ or ∞, and the decision variables $\boldsymbol{x}_t = [x_{t|t}, .., x_{t+N|t}]$, $\boldsymbol{u}_t$, $\boldsymbol{\delta}_t$, $\boldsymbol{z}_t$ (defined similarly) are the states, inputs, binary variables and auxiliary variables respectively. The optimal solution to (1) defines the MPC policy as $\pi_{MPC}(x_t) = u^\star_{t|t}$.

The optimization problem (1) can be expressed as a multi-parametric Mixed-Integer Linear Program (mp-MILP), with the parameters being the system state x_t. The mp-MILP can be concisely expressed as follows:

$$\begin{aligned} V^\star(b) = \min_{z,y} \quad & c^\top z + d^\top y, \\ \text{s.t.} \quad & Az + By = b, \\ & z \geq 0, \ y \in \{0,1\}^M \end{aligned} \tag{2}$$

with continuous decision variables $z \in \mathbb{R}^n$, binary decision variables $y \in \{0,1\}^M$ and parameters $b \in \mathbb{R}^m$. Let $z^\star(b), y^\star(b)$ be an optimal solution to (2) and $V^\star(b)$ be the optimal cost. For a given parameter b, let $\mathcal{F}(b)$ be the set of (z,y) feasible for (2) and $V(b,z,y)$ be the cost of any $(z,y) \in \mathcal{F}(b)$, with sub-optimality given by $\frac{V(b,z,y)-V^\star(b)}{|V^\star(b)|}$. Also define $\mathbb{B} = \{b \in \mathbb{R}^m | \mathcal{F}(b) \neq \emptyset\}$ as the set of parameters for which (2) is feasible.

In this work, we aim to exploit the parametric nature of the mp-MILP (2) to predict a solution $(\tilde{z}(b), \tilde{y}(b)) \in \mathcal{F}(b)$ for real-time MPC, and quantify its sub-optimality using *strategies*. The strategy maps a parameter b to an element of a finite and discrete set $\mathbb{S}$, which describes the complete information necessary to recover a feasible point $(z(b), y(b))$ for (2) (if it exists), formally defined next.

Definition 1: A function $s : \mathbb{B} \to \mathbb{S}$ is a strategy for mp-MILP (2) if there exists a map $R(\cdot)$ such that $\forall b \in \mathbb{B}$: $R(b, s(b)) = (z(b), y(b)) \in \mathcal{F}(b)$, and evaluating $R(b, s(b))$ is computationally efficient compared to solving (2) directly.

For example, in [11] the set $\mathbb{S}$ is given by all possible sets of active constraints for (2) and for each $b \in \mathbb{B}$, $s(b)$ picks the active constraints for a $(z,y) \in \mathcal{F}(b)$. The recovery map is then given as the solution of a linear system of equations.

The strategy $s^\star(b)$ is said to be *optimal* at $b \in \mathbb{B}$ if $R(b, s^\star(b)) = (z^\star(b), y^\star(b))$. We construct functions $V_{\text{lb}}(\cdot,\cdot)$, $V_{\text{ub}}(\cdot,\cdot)$ that satisfy the following properties:

1) For any $(z,y) \in \mathcal{F}(b)$ such that $R(b, s(b)) = (z,y)$,
$$V_{\text{lb}}(b, s(b)) \leq V(b,z,y) \leq V_{\text{ub}}(b, s(b)).$$
2) For the optimal strategy $s^\star(b)$,
$$V_{\text{lb}}(b, s^\star(b)) = V^\star(b) = V_{\text{ub}}(b, s^\star(b)).$$

For any $b \in \mathbb{B}$, we use $V_{\text{lb}}(\cdot,\cdot)$, $V_{\text{ub}}(\cdot,\cdot)$ to estimate the quality of a strategy $s(b)$ with respect to $s^\star(b)$. In particular, the sub-optimality of a predicted strategy $\tilde{s}(b)$ and the recovered solution $R(b, \tilde{s}(b)) = (\tilde{z}(b), \tilde{y}(b))$ is over-estimated as

$$\text{sub\_opt}(b, \tilde{s}(b)) = \left| \frac{V_{\text{ub}}(b, \tilde{s}(b)) - V_{\text{lb}}(b, \tilde{s}(b))}{V_{\text{lb}}(b, \tilde{s}(b))} \right|. \tag{3}$$

Organization: First, we present our choice of strategy $s(\cdot)$, the recovery map $R(\cdot)$, and the bounding functions $V_{\text{lb}}(\cdot)$, $V_{\text{ub}}(\cdot)$ that meet the desired properties in Section III. Section IV presents a supervised learning framework to approximate the optimal strategy $s^\star(b)$, and evaluate $R(b, s^\star(b))$, $V_{\text{lb}}(b, s^\star(b)), V_{\text{ub}}(b, s^\star(b))$ efficiently for predicting solutions to (1) online, and evaluate its sub-optimality. Finally, we demonstrate our approach for motion planning using MIMPC and compare against open-source and commercial MILP solvers in Section V.

III. Strategy-based Solution to mp-MILPs

In this section, we present our design of the strategy $s(\cdot)$, the recovery map $R(b, s(b))$ and the bounding functions $V_{\text{lb}}(b, s(b)), V_{\text{ub}}(b, s(b))$, along with theoretical justification using ideas from the mp-MILP literature.

A. Preliminaries: Solving MILPs using Branch-and-Bound

Branch-and-Bound (BnB) is a tree search algorithm that solves MILPs, with each node given as the LP sub-problem

$$\begin{aligned} V^\star_{LP}(b, \text{lb}, \text{ub}) = \min_{z,y} \quad & c^\top z + d^\top y, \\ \text{s.t} \quad & Az + By = b, \\ & z \geq 0, \ \text{lb} \leq y \leq \text{ub}, \end{aligned} \tag{4}$$

where the binary variable bounds $\text{lb}, \text{ub} \in \{0,1\}^M$. For any $b \in \mathbb{R}^m$, let $\mathcal{F}_{LP}(b, \text{lb}, \text{ub})$, $(z^\star(b, \text{lb}, \text{ub}), y^\star(b, \text{lb}, \text{ub}))$ denote its feasible set and optimal solution respectively. At iteration i of BnB, a collection of sub-problems identified by

$\mathcal{C}^i = \{\{\text{lb}_k^i, \text{ub}_k^i\}_{k=1}^{n_i}\}$ is maintained such that they form a *cover* over the set of binary sequences $\{0,1\}^M$:

$$\bigcup_{k=1}^{n^i} [\text{lb}_k^i, \text{ub}_k^i] \supseteq \{0,1\}^M \Rightarrow \bigcup_{k=1}^{n^i} \mathcal{F}_{LP}(b, \text{lb}_k^i, \text{ub}_k^i) \supseteq \mathcal{F}(b).$$

A lower bound on $V^\star(b)$ at iteration i is given as

$$\underline{V}^i(b) = \min_{k\in\{1,\dots,n^i\}} V_{LP}^\star(b, \text{lb}_k^i, \text{ub}_k^i) \leq V^\star(b),$$

which can be shown in three steps:

1) Let $(\bar{z}, \bar{y}) = \arg\min\{c^\top z + d^\top y | (z,y) \in \bigcup_{k=1}^{n^i} \mathcal{F}_{LP}(b, \text{lb}_k^i, \text{ub}_k^i)\}$ and $\bar{k} \in \{1,..,n^i\}$ be the sub-problem such that $(\bar{z}, \bar{y}) \in \mathcal{F}_{LP}(b, \text{lb}_{\bar{k}}^i, \text{ub}_{\bar{k}}^i)$. Then $c^\top \bar{z} + d^\top \bar{y} = V_{LP}^\star(b, \text{lb}_{\bar{k}}^i, \text{ub}_{\bar{k}}^i)$ due to global optimality of the $\bar{k}$th LP sub-problem.
2) Observe that $V_{LP}^\star(b, \text{lb}_{\bar{k}}^i, \text{ub}_{\bar{k}}^i) = \underline{V}^i(b)$, because otherwise, $\exists l \in \{1,..,n_i\}$ such that $V_{LP}^\star(b, \text{lb}_l^i, \text{ub}_l^i) < V_{LP}^\star(b, \text{lb}_{\bar{k}}^i, \text{ub}_{\bar{k}}^i)$, which implies the contradiction $\min\{c^\top z + d^\top y | \mathcal{F}_{LP}(b, \text{lb}_l^i, \text{ub}_l^i)\} < \min\{c^\top z + d^\top y | \bigcup_{k=1}^{n^i} \mathcal{F}_{LP}(b, \text{lb}_k^i, \text{ub}_k^i)\}$.
3) Finally since the sub-problems form a cover, $\underline{V}^i(b) = \min\{c^\top z + d^\top y | (z,y) \in \bigcup_{k=1}^{n_i} \mathcal{F}_{LP}(b, \text{lb}_k^i, \text{ub}_k^i)\} \leq \min\{c^\top z + d^\top y | (z,y) \in \mathcal{F}(b)\} = V^\star(b)$.

Define the set of indices $\mathcal{I}^i \subseteq \{1,\dots,n^i\}$ such that their corresponding sub-problems have solutions that are also feasible for (2), i.e.,

$$\mathcal{I}^i = \{k \in 1,\dots,n^i | (z_{LP}^\star(b, \text{lb}_k^i, \text{ub}_k^i), y_{LP}^\star(b, \text{lb}_k^i, \text{ub}_k^i)) \in \mathcal{F}(b)\}.$$

Then, an upper bound on $V^\star(b)$ at iteration i is given as

$$V^\star(b) \leq \bar{V}^i(b) = \begin{cases} \min_{k\in\mathcal{I}^i} V_{LP}^\star(b, \text{lb}_k^i, \text{ub}_k^i), & \mathcal{I}^i \neq \emptyset, \\ \infty & \mathcal{I}^i = \emptyset \end{cases}$$

which is evident because $V^\star(b) \leq V_{LP}^\star(b, \text{lb}_k^i, \text{ub}_k^i)\ \forall k \in \mathcal{I}^i$. If $\mathcal{I}^i = \emptyset$, often rounding heuristics are applied to some sub-problem solutions to produce a feasible solution in $\mathcal{F}(b)$. This describes the *bounding* process of BnB.

If $\underline{V}^i(b) \neq \bar{V}^i(b)$, then the search proceeds to the next iteration via the *branching* process, which constructs a new cover $\mathcal{C}^{i+1}$ from $\mathcal{C}^i$ by splitting a sub-problem, say $\{\text{lb}_k^i, \text{ub}_k^i\}$, into two new sub-problems $\{\{\text{lb}_k^{i+1}, \text{ub}_k^{i+1}\}, \{\text{lb}_{k+1}^{i+1}, \text{ub}_{k+1}^{i+1}\}\}$ by fixing one or more variables to 0 in one sub-problem, and to 1 in the other. The branching decisions depend on $\underline{V}^i(b), \bar{V}^i(b)$, the optimal sub-problem solutions, and some tree search heuristics.

The search begins with the root node given by $\mathcal{C}^0 = \{\{\mathbf{0}_M, \mathbf{1}_M\}\}$ defining the LP relaxation of (2). The search terminates at iteration "$\star$" when $\underline{V}^\star(b) = \bar{V}^\star(b)$ and the optimal solution is given by the feasible solution that yields $\bar{V}^\star(b)$. This optimality certificate is represented by

1) the optimal cover $\mathcal{C}^\star(b) = \{\{\text{lb}_k^\star, \text{ub}_k^\star\}_{k=1}^{n_\star}\}$ describing the LP sub-problems at the terminal iteration,
2) the optimal binary solution $y^\star(b)$ obtained from the sub-problem corresponding to the upper-bound $\bar{V}^\star(b)$.

B. Strategy Description for Parametric MILPs

Inspired by the optimality certificate obtained from BnB, we propose the following strategy, bounding functions and recovery map:

$$s(b) = \{\mathcal{C}^\star(b), y^\star(b)\}, \tag{5a}$$

$$V_{\text{lb}}(b, s(b)) = \min_{k\in\{1,\dots,n_\star\}} V_{LP}^\star(b, \text{lb}_k^\star, \text{ub}_k^\star), \tag{5b}$$

$$V_{\text{ub}}(b, s(b)) = \min_{Az+By^\star(b)=b,\ z\geq 0} \quad c^\top z + d^\top y^\star(b), \tag{5c}$$

$$R(b, s(b)) = \arg\min_{Az+By^\star(b)=b,\ z\geq 0} \quad c^\top z + d^\top y^\star(b). \tag{5d}$$

The strategy $s(\bar{b})$ for parameter $\bar{b}$ is optimal if it certifies optimality of the MILP (2) via $V_{\text{lb}}(\bar{b}, s(\bar{b})) = V^\star(\bar{b}) = V_{\text{ub}}(\bar{b}, s(\bar{b}))$. The next theorem highlights the parametric behaviour of the optimality certificate provided by $s(\bar{b})$, i.e., the set of parameters $\mathcal{P}_{\bar{b}}$ for which $s(\bar{b})$ remains optimal. Thus, for any parameter $b \in \mathcal{P}_{\bar{b}}$, the optimal solution can be computed via (5d) without BnB.

Theorem 1: Let $s^\star(\bar{b}) = \{\mathcal{C}^\star(\bar{b}), y^\star(\bar{b})\}$ be the optimal strategy for solving MILP (2) with the parameter $\bar{b}$. Then there is a set of parameters $\mathcal{P}_{\bar{b}} \subset \mathbb{B}$, given by a union of convex polyhedra for which $s^\star(\bar{b})$ is also optimal,

$$V_{\text{lb}}(b, s^\star(\bar{b})) = V^\star(b) = V_{\text{ub}}(b, s^\star(\bar{b})) \quad \forall b \in \mathcal{P}_{\bar{b}}$$

Proof: Let $\mathcal{S}_{\bar{b}}^\star \subset \{1,\dots,n_\star\}$ be the set of feasible sub-problems in the cover $\mathcal{C}^\star(\bar{b}) = \{\{\text{lb}_k^\star, \text{ub}_k^\star\}_{k=1}^{n_\star}\}$, and let $\bar{k}$ be the optimal sub-problem for which $y^\star(\bar{b}, \text{lb}_{\bar{k}}^\star, \text{ub}_{\bar{k}}^\star) = y^\star(\bar{b})$ and $V_{LP}^\star(\bar{b}, \text{lb}_{\bar{k}}^\star, \text{ub}_{\bar{k}}^\star) = V^\star(\bar{b})$.

For sub-problem $k \in \mathcal{S}_{\bar{b}}^\star$, we have from [6, Theorems 6.2, 6.5] that there exists a (convex) polyhedron of parameters b given by $\mathcal{K}^k = \cup_{i=1}^{p_k} \mathcal{K}_i^k \subset \mathbb{B}$ such that each $\mathcal{K}_i^k$ is polyhedral, and $(z^\star(b, \text{lb}_k, \text{ub}_k), y^\star(b, \text{lb}_k, \text{ub}_k))$ are affine functions of b for $b \in \mathcal{K}_i^k$. Define the set $\mathcal{Z}^k = \cup_{i=1}^{p_k} \{(z,y,b) \mid b \in \mathcal{K}_i^k, (z,y) = (z^\star(b, \text{lb}_k, \text{ub}_k), y^\star(b, \text{lb}_k, \text{ub}_k))\}$ for each $k \in \mathcal{S}_{\bar{b}}^\star$ and for the optimal sub-problem $\bar{k}$, define the set $\mathcal{Z}^\star = \{(z,y,b) \mid (z,y,b) \in \mathcal{Z}^{\bar{k}},\ y = y^\star(\bar{b})\}$.

For any parameter $b \neq \bar{b}$, the solution of sub-problem $\bar{k}$ is also optimal for the MILP (4) at b if $V_{LP}^\star(b, \text{lb}_{\bar{k}}^\star, \text{ub}_{\bar{k}}^\star) = \min_{i\in\mathcal{S}_{\bar{b}}^\star} V_{LP}^\star(b, \text{lb}_i^\star, \text{ub}_i^\star)$ and $y^\star(b, \text{lb}_{\bar{k}}^\star, \text{ub}_{\bar{k}}^\star)) \in \{0,1\}^M$. Thus, the strategy $s^\star(\bar{b})$ is optimal for b if

$$\begin{aligned} & V_{LP}^\star(b, \text{lb}_{\bar{k}}^\star, \text{ub}_{\bar{k}}^\star) = V_{\text{lb}}(b, s^\star(\bar{b})),\ y^\star(b, \text{lb}_{\bar{k}}^\star, \text{ub}_{\bar{k}}^\star) = y^\star(\bar{b}) \\ \Leftrightarrow\ & c^\top z^{\bar{k}} + d^\top y^{\bar{k}} \leq c^\top z^k + d^\top y^k, \\ & (z^{\bar{k}}, y^{\bar{k}}, b) \in \mathcal{Z}^\star, (z^k, y^k, b) \in \mathcal{Z}^k\ \forall k \in \mathcal{S}_{\bar{b}}^\star \backslash \{\bar{k}\}. \end{aligned}$$

Thus, the set of parameters for which $s^\star(\bar{b})$ is the optimal strategy is given by the set

$$\mathcal{P}_{\bar{b}} = \left\{ b \,\middle|\, \begin{array}{l} \exists (z^{\bar{k}}, y^{\bar{k}}, b) \in \mathcal{Z}^\star, \\ \exists (z^k, y^k, b) \in \mathcal{Z}^k\ \forall k \in \mathcal{S}_{\bar{b}}^\star \backslash \{\bar{k}\} : \\ \quad c^\top z^{\bar{k}} + d^\top y^{\bar{k}} \leq c^\top z^k + d^\top y^k \end{array} \right\}$$

which is a union of convex polyhedra ($\because$ affine projection of unions of convex polyhedra $\mathcal{Z}^\star, \mathcal{Z}^k$, intersected by affine halfspaces $c^\top z^{\bar{k}} + d^\top y^{\bar{k}} \leq c^\top z^k + d^\top y^k$). ■

The sets $\mathcal{P}_{\bar{b}_i}$ can be constructed using ideas from multi-parametric programming, but this approach would become intractable as the size of the problem increases. Instead, we propose a supervised classification approach to predict an optimal strategy for a given parameter in the next section. For a predicted strategy $\tilde{s}(b)$, the functions in (5b), (5c) are used to quantify its sub-optimality compared to $s^\star(b)$. If no feasible solution is found or the predicted strategy is too sub-optimal, an optimal solution can be retrieved from $\tilde{\mathcal{C}}(b)$ by solving MILP sub-problems.

IV. LAMPOS: Learning-based Approximate MIMPC with Parametric Optimality Strategies

This section presents LAMPOS: (A) an offline supervised learning framework for strategy prediction, and (B) an online algorithm for finding solutions to (1). The learning problem of predicting $s^\star(b)$ is split into two classification problems, from parameters b to corresponding labels $(\gamma^\star, \upsilon^\star)$ for optimal cover $\mathcal{C}^\star(b)$ and binary solution $y^\star(b)$, respectively. The number of possible strategies/labels is exponential in the problem size, which would make the classification problem intractable as well. To address this issue, we construct our dataset with a limited number of strategies using the approach in [11]. For online deployment, the predicted strategy is used to obtain solutions to the (1) using $R(\cdot)$, with sub-optimality quantification using $V_{\text{lb}}(\cdot)$, $V_{\text{ub}}(\cdot)$.

A. Offline Supervised Learning for Strategy Prediction

1) Dataset Construction: Our dataset consists of parameter-strategy pairs $(b_i, s(b_i))$ where the strategy $s(b_i) = (\gamma_i,\ \upsilon_i)$ consists of a tuple of labels for the cover and binary solution respectively. We solve (2) for each b_i and collect: *(i)* the set of leaves of the BnB tree that constitute our optimal cover $\mathcal{C}^\star(b_i)$ and *(ii)* the binary solution $y^\star(b_i)$, and assign a label to each. Let $\mathcal{S}(\mathcal{B}_T) = \{s_1, s_2, \ldots, s_M\}$ be M strategies corresponding to T i.i.d. parameter samples $\mathcal{B}_T = \{b_1, b_2, \ldots, b_T\}$. To determine the set of labels for the supervised classification problem, we assess the probability of encountering a new strategy with a new i.i.d. sample b_{T+1}, i.e., $\mathbb{P}(s(b_{T+1}) \notin \mathcal{S}(\mathcal{B}_T))$. We adopt the Good-Turing estimator $G = T_1/T$, where T_1 represents the number of strategies that have appeared exactly once, to bound this probability with confidence at least $1-\beta$ as

$$\mathbb{P}\left(s\left(b_{T+1}\right) \notin \mathcal{S}\left(\mathcal{B}_T\right)\right) \leq G + (2\sqrt{2}+\sqrt{3})\sqrt{\frac{1}{T}\ln\left(\frac{3}{\beta}\right)}.$$

For a fixed confidence $\beta \ll 1$, we sample parameters and update G until the right-hand side bound is less than a desired probability guarantee $\epsilon > 0$. To conclude the dataset construction, we use the insight from Theorem 1 to eliminate redundant strategies by searching locally around b_i in the dataset, for strategies $s(b_j)$ which maintain optimality at b_i.

2) Architecture and Learning problem: The classification problem for predicting the strategy can be solved using popular prediction architectures such as Deep Neural Networks (DNN) and Random Forests (RF), discussed as follows.

DNN-based Architecture: We use a feedforward DNN for cover prediction, with L layers composed together to define a function of the form $\hat{\gamma} = f_L(f_{L-1}(\ldots f_1(b)))$. The output of the lth layer is given by $y_l = f_l(y_{l-1}) = \sigma_l(W_l y_{l-1} + w_l)$ where $W_l \in \mathbb{R}^{p_l \times p_{l-1}}$ and $w_l \in \mathbb{R}^{p_l}$ are the layer's parameters, $y_0 = b$, $y_L = \hat{\gamma}$ and σ_l is the activation function used to model nonlinearities. For binary solution prediction for the MIMPC, we express $y^*(b) = [y_1^*(b), y_2^*(b), \ldots, y_N^*(b)]$ to divide the classification problem into N sub-problems, corresponding to each step along the MPC horizon N. Each sub-problem $j \in (1, 2, \ldots, N)$ consists of finding the label $\nu_j \in \Upsilon_j$ associated with the binary solution for step j for parameter b, where Υ_j is the set of labels for sub-problem j, and is predicted using a H layer DNN with parameters $W_h \in \mathbb{R}^{p_h \times p_{h-1}}$ and $w_h \in \mathbb{R}^{p_h}$ returning a label estimation $\hat{\nu}_j$. The label for $\hat{y}(b)$ is given by the vector of labels $\hat{\upsilon} = [\hat{\nu}_1, \hat{\nu}_2, \ldots, \hat{\nu}_N]$. This architecture makes the classification task easier than directly recovering the full binary solution $y^*(b)$ due to the high number of different binary solutions in the dataset. Alternatively, any state-of-the-art prediction architecture [15] can be adopted for capturing the temporal dependence in the binary solution. The training process for DNN consists of finding the network parameters that minimize a loss function that encodes misclassification error. For all the classification problems, the Cross Entropy loss function is chosen, defined as $H(p,q) = -\sum_{i=1}^{K} p_i \log(q_i)$, where p is the true label distribution over K labels and q is the predicted label distribution. The optimization problem for DNN training is solved using Stochastic Gradient Descent.

RF-based Architecture: The RF consists of multiple decision trees that are trained on random subsets of the training data, and the final prediction combines the predictions of all the decision trees in the forest by taking the majority vote. For the classification problems for binary and cover prediction, the Gini impurity criterion can be used as the splitting criterion, which measures the degree of impurity in a set of labels. The Gini impurity is defined as $\text{Gini}(p) = \sum_{i=1}^{K} p_i(1-p_i)$ where p_i is the fraction of samples in a given set that belong to class i. The Gini impurity is minimized by using a greedy approach for selecting the split of the parameter space that maximizes the reduction in impurity.

B. Online Deployment for MIMPC

After training the prediction models offline, the online deployment of our approach for MIMPC is described in Algorithm 1. The inputs to the algorithm are the trained strategy prediction model $P_{\theta^\star}(\cdot)$, the state of system x_t and the desired sub-optimality tolerance tol. The function `solve_MIMPC`$(\cdot)$ returns the MPC policy $\pi_{MPC}(\cdot)$. Inside it, we first query the prediction model at the current state to obtain a strategy consisting of the cover $\tilde{\mathcal{C}}(x_t)$ and a candidate binary solution $\tilde{y}(x_t)$. The list of LP sub-problems in the cover is augmented with another LP by fixing the binary variable bounds to $\tilde{y}(x_t)$. Then the LPs are solved in parallel, while keeping track of MILP feasible solutions. The solved sub-problems are sorted in the increasing order of cost, with ∞ assigned to the cost of infeasible LPs. The lower bound LB on the optimal cost is provided by the first LP sub-problem. The upper bound UB is obtained from the best MILP feasible solution, if any. If the estimated sub-optimality $\frac{UB-LB}{|LB|}$ is within tolerance, the MPC policy is obtained as $Sz^\star$ where $z^\star$ is the LP solution corresponding to the upper bound and S is a matrix that selects $u^\star_{t|t}$ from

Algorithm 1: LAMPOS (Online)

Input : $P_{\theta^\star}(\cdot)$, x_t, tol
Output: $\pi_{MPC}(x_t)$
Procedure solve_MIMPC(x_t):
  /* Predict strategy */
  $[\tilde{\mathcal{C}}(x_t) := \{\{\tilde{\text{lb}}_k, \tilde{\text{ub}}_k\}\}_{k=1}^{n_c},\quad \tilde{y}(x_t)] \leftarrow P_{\theta^\star}(x_t)$
  $\tilde{\mathcal{C}}(x_t) \leftarrow \tilde{\mathcal{C}}(x_t) \cup \{(\tilde{y}(x_t), \tilde{y}(x_t))\}$
  /* Solve LPs in parallel */
  $\tilde{\mathcal{I}} \leftarrow \emptyset$, $\pi_{MPC}(x_t) \leftarrow \emptyset$
  parfor $k = 1$ **to** $n_c + 1$ **do**
    $(V_k, z_k, y_k) \leftarrow$solve_LP$(x_t, \tilde{\text{lb}}_k, \tilde{\text{ub}}_k)$
    if $y_k \in \{0,1\}^M$ **then**
      $\tilde{\mathcal{I}} \leftarrow \tilde{\mathcal{I}} \cup \{k\}$
  end for
  /* Check sub-optimality */
  $\{(\bar{V}_k, \tilde{\text{lb}}_k, \tilde{\text{ub}}_k)\}_{k=1}^{n_c} \leftarrow \text{sort}(\{(V_k, \tilde{\text{lb}}_k, \tilde{\text{ub}}_k)\}_{k=1}^{n_c})$
  $LB = \bar{V}_1$
  if $\tilde{\mathcal{I}} \neq \emptyset$ **then**
    $UB = \min_{k \in \tilde{\mathcal{I}}} V_k$, $z^\star \leftarrow z_{\arg\min_{k \in \tilde{\mathcal{I}}} V_k}$
    if $UB - LB \leq tol \cdot |LB|$ **then**
      $\pi_{MPC}(x_t) = Sz^\star$
  if $\pi_{MPC} = \emptyset$ **then**
    $(\bar{V}, \bar{z}, \bar{y}) \leftarrow$find_sol $(\{(\bar{V}_k, \tilde{\text{lb}}_k, \tilde{\text{ub}}_k)\}_{k=1}^{n_c})$
    $\pi_{MPC}(x_t) = S\bar{z}$
return $\pi_{MPC}(x_t)$
Backup find_sol($\{(V_k, \text{lb}_k, \text{ub}_k)\}_{k=1}^{n_c}$):
  $V_{n_c+1} \leftarrow \infty$, $(\bar{V}, \bar{z}, \bar{y}) \leftarrow (\infty, \emptyset, \emptyset)$
  for $k = 1$ **to** n_c **do**
    $(\hat{V}_k, \hat{z}_k, \hat{y}_k) \leftarrow$solve_MILP$(x_t, \text{lb}_k, \text{ub}_k)$
    $(\bar{V}, \bar{z}, \bar{y}) \leftarrow$ best_sol$(\{(\hat{V}_i, \hat{z}_i, \hat{y}_i)\}_{i=1}^{k})$
    if $\bar{V} \leq V_{k+1}$ **then**
      break
  end for
return $(\bar{V}, \bar{z}, \bar{y})$

$z^\star$. If no MILP feasible solutions were found (meaning $\tilde{\mathcal{I}} = \emptyset$) or the predictions do not meet the sub-optimality tolerance, we send the sorted LP sub-problems to the backup procedure find_sol$(\cdot)$ which solves a sequence of MILP sub-problems. The backup returns an optimal solution if the MILP (1) is feasible, and nothing otherwise. A deficiency of Algorithm 1 is the requirement of a MILP solver for the backup procedure, which is unavoidable due to the $\mathcal{NP}$-hardness of (2) if optimal solutions are required (e.g., a stabilizing MPC policy for hybrid systems [7]). A common heuristic solution is to query the prediction model for multiple binary solution candidates [9], [12], which can be readily incorporated into Algorithm 1.

V. Numerical Experiments

In this section we demonstrate the effectiveness of our approach for a motion planning problem and compare the performance against MILP solvers: GLPK-MI [16], SCIP [17], Mosek [18] and Gurobi [19]. Our implementation is available at: https://github.com/shn66/LAMPOS (●).

A. MIMPC for 2D Motion planning

The motion planning problem is to steer the robot to the origin subject to state-input and obstacle avoidance constraints as depicted in Fig. 2. The robot is modelled

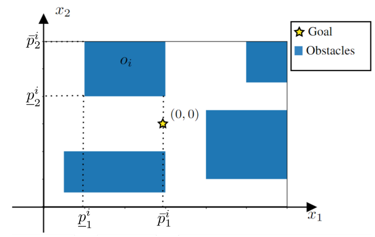

Fig. 2. Obstacle configuration of the 2D motion planning problem, with the ith obstacle's shape: $\{(X,Y)|\ [\underline{p}_1^i, \underline{p}_2^i] \leq [X,Y] \leq [\bar{p}_1^i, \bar{p}_2^i]\}$.

as an Euler discretized, double integrator with state $x_t = [X_t, \dot{X}_t, Y_t, \dot{Y}_t]$, control $u_t = [\ddot{X}_t, \ddot{Y}_t]$ and sampling period $dt = 0.1s$. Policy $\pi_{MPC}(x_t)$ is computed by solving the mp-MILP (6), which is parametric in x_t. The obstacle avoidance constraints are encoded using the big-M method, with binary vectors $\underline{\delta}_{k|t}^i, \bar{\delta}_{k|t}^i \in \{0,1\}^2$ introduced for each obstacle i at time k, totalling $4 \cdot n_{\text{obs}} \cdot N$ binary variables for a prediction horizon of N and $n_{\text{obs}} = 4$ obstacles. The vectors $\bar{x} = -\underline{x} = [3,3,2,2], \bar{u} = -\underline{u} = [2,2]$ define the state-input constraints in (6), and $Q = 10^3 I_4,\ R = 50 I_2,\ P = 10^5 I_4$ define the cost matrices. We model the problem using CVXPY and perform experiments for $N = 20, 40$.

$$
\begin{aligned}
\min_{\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\delta}_t} \quad & \|Px_{t+N|t}\|_\infty + \sum_{k=t}^{t+N-1} \|Qx_{k|t}\|_\infty + \|Ru_{k|t}\|_\infty \\
\text{s.t.} \quad & x_{k+1|t} = I_2 \otimes \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} x_{k|t} + I_2 \otimes \begin{bmatrix} 0 \\ dt \end{bmatrix} u_{k|t}, \\
& \underline{x} \leq x_{k+1|t} \leq \bar{x},\ \underline{u} \leq u_{k|t} \leq \bar{u}, \\
& \bar{p}^i - \bar{\delta}_k^i M \leq [X_{k+1|t}, Y_{k+1|t}] \leq \underline{p}^i + M \underline{\delta}_k^i, \\
& \mathbf{1}^\top \underline{\delta}_k^i + \mathbf{1}^\top \bar{\delta}_k^i \leq 3, \\
& \underline{\delta}_k^i, \bar{\delta}_k^i \in \{0,1\}^2 \quad \forall i = 1, \ldots, n_{\text{obs}} \\
& x_{t|t} = x_t, \qquad \forall k = t, \ldots, t+N-1.
\end{aligned} \tag{6}
$$

B. Implementation Details

1) Dataset construction: For dataset construction, we randomly sample parameters $b = x_t$ and solve MILP (6). We used SCIP to save the leaves of the BnB solution tree[1]. For meeting the probability bound defined in Sec. IV-A.1, we fix $\beta = 10^{-3}, \epsilon = 10^{-1}$. We collected $\sim 10^5$ samples for both $N = 20, 40$ cases. After data collection, we further process the dataset by reassigning strategies with covers with a large number of LP sub-problems, to another cover with the least sub-optimality and with fewer LP sub-problems than a pre-defined threshold. This limits the online computation for solving the LP sub-problems from the cover in parallel.

2) Supervised learning: For strategy predictions, we use RF for the $N = 20$ case and DNN for the $N = 40$ case. For RF implementation we used the *RandomForestClassifier* from *sci-kit* setting number of trees $n_t = 10$ and used weighted tree splitting for both cover and binary solution classification to mitigate unbalanced-ness in the dataset. The RFs were trained until prediction accuracies $\geq 97\%$ are achieved for binary and cover predictions. We use Pytorch

[1]If the LP sub-problems at the leaves of the BnB tree are unavailable, we provide a recursive algorithm (@github repo: ●) to construct a cover $\{\{\text{lb}_k, \text{ub}_k\}\}_{k=1}^{n_c}$ given the optimal sub-problem $\{\text{lb}^\star, \text{ub}^\star\}$, and a partial list of sub-problems $\{\{\text{lb}_k, \text{ub}_k\}\}_{k=1}^{n_p}$. The algorithm proceeds by adding disjoint *facets* $[\text{lb}_i, \text{ub}_i] \subset [0,1]^M$ until $\cup_{k=1}^{n_c} [\text{lb}_k, \text{ub}_k] \supset \{0,1\}^M$.

for our DNN implementation with architectures given by 2 hidden layers with width 64 for binary prediction, and 3 hidden layers with width 128 for cover prediction.

C. Results

We tested our approach for cases $N = 20, 40$ by sampling 100 initial conditions x_0 and solve (6) for the policy $\pi_{MPC}(\cdot)$ until the robot reaches the origin. For Algorithm 1, the LPs were solved using ECOS [20] (for rapid infeasibility detection) and the backup MILP sub-problems using SCIP. We compare LAMPOS against GLPK_MI, SCIP, Mosek and Gurobi for solve times. The solve times of our approach are compared to other solvers in Fig. 3, 4. Our solve times include prediction time and LP sub-problem solve times. In addition, we also solve (6) with SCIP, Mosek and Gurobi with a time-limit of 50ms for $N = 20, 40$ [2], and compare against LAMPOS for sub-optimality of the obtained solution (if any). For each solver, we report the average sub-optimality of feasible solutions and % of instances where it timed-out.

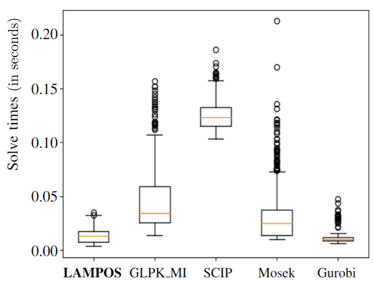

Fig. 3. Comparison with solve times of other solvers for $N = 20$

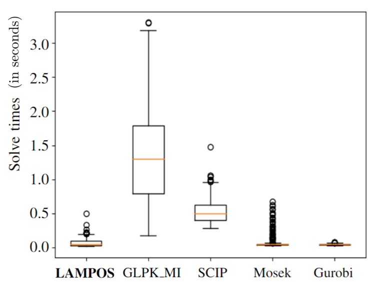

Fig. 4. Comparison with solve times of other solvers for $N = 40$

TABLE I

PERFORMANCE COMPARISON WITH $50ms$ TIME LIMIT

Horizon	Metric	Solver			
		LAMPOS	SCIP	Mosek	Gurobi
N=20	Sub-opt (Avg)	**0.04**	0.34	0.16	**1e-8**
	Time-out (%)	**0**	**0**	0.2	0.8
N=40	Sub-opt (Avg)	**0.07**	-	0.2	**1e-10**
	Time-out (%)	18.6	100	22.7	**10.8**

Discussion: In Fig. 3, 4 for solve times, we see that LAMPOS outperforms open-source solvers GLPK_MI, SCIP and is comparable to Mosek, Gurobi. Table I shows that LAMPOS, Gurobi reliably find high-quality solutions within the time limit compared to SCIP, Mosek (where we use (3) for quantifying LAMPOS' sub-optimality). In our experiments, we observed competitive solve times for LAMPOS when $\tilde{y}(b) = y^\star(b)$, but also quick recovery otherwise by reusing the LP sub-problem information from $\tilde{\mathcal{C}}(b)$ during backup calls (observed in 0.016%, 0.014% of the $N = 20, 40$ cases). To improve solve times in the future, instead of solving the LP sub-problems (4) explicitly, exploiting their parametric dependence on $(b, \mathrm{lb}, \mathrm{ub})$[14].

[2]No such interface for GLPK_MI in CVXPY

VI. CONCLUSION

We proposed a strategy-based prediction framework to solve mp-MILPs online with sub-optimality quantification, and demonstrate it for real-time MIMPC. By exploiting the parametric nature of the optimality certificate for mp-MILPs given by the optimal set of LP sub-problems and an optimal integer solution, we observed favourable performance compared to state-of-the-art MILP solvers. For future work, we aim to train prediction models for solving the parametric LP sub-problems to further improve solve-times.

REFERENCES

[1] D. Ioan, I. Prodan, S. Olaru, F. Stoican, and S.-I. Niculescu, "Mixed-integer programming in motion planning," *Annual Reviews in Control*, vol. 51, 2021.
[2] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *arXiv preprint arXiv:2205.04422*, 2022.
[3] S. Tokuda, M. Yamakita, H. Oyama, and R. Takano, "Convex approximation for ltl-based planning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021.
[4] W. P. Heemels, B. De Schutter, and A. Bemporad, "Equivalence of hybrid dynamical models," *Automatica*, vol. 37, 2001.
[5] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, 1999.
[6] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.
[7] A. Bemporad, F. Borrelli, and M. Morari, "Optimal controllers for hybrid systems: Stability and piecewise linear explicit form," in *Proceedings of the 39th IEEE Conference on Decision and Control*, vol. 2, IEEE, 2000.
[8] G. Cimini and A. Bemporad, "Exact complexity certification of active-set methods for quadratic programming," *IEEE Transactions on Automatic Control*, vol. 62, 2017.
[9] D. Masti and A. Bemporad, "Learning binary warm starts for multiparametric mixed-integer quadratic programming," in *2019 18th European Control Conference (ECC)*, IEEE, 2019.
[10] J.-J. Zhu and G. Martius, "Fast non-parametric learning to accelerate mixed-integer programming for hybrid model predictive control," *IFAC-PapersOnLine*, vol. 53, 2020.
[11] D. Bertsimas and B. Stellato, "Online mixed-integer optimization in milliseconds," *INFORMS Journal on Computing*, vol. 34, 2022.
[12] A. Cauligi, P. Culbertson, E. Schmerling, M. Schwager, B. Stellato, and M. Pavone, "Coco: Online mixed-integer control via supervised learning," *IEEE Robotics and Automation Letters*, vol. 7, 2021.
[13] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, 2021.
[14] X. Zhang, M. Bujarbaruah, and F. Borrelli, "Near-optimal rapid mpc using neural networks: A primal-dual policy learning framework," *IEEE Transactions on Control Systems Technology*, vol. 29, 2020.
[15] A. Cauligi, A. Chakrabarty, S. Di Cairano, and R. Quirynen, "Prism: Recurrent neural networks and presolve methods for fast mixed-integer optimal control," in *L4DC*, PMLR, 2022.
[16] A. Makhorin, "Glpk (gnu linear programming kit)," *http://www. gnu. org/s/glpk/glpk. html*, 2008.
[17] T. Achterberg, "Scip: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, 2009.
[18] M. ApS, "Mosek optimization toolbox for matlab," *User's Guide and Reference Manual, Version*, vol. 4, 2019.
[19] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual."
[20] A. Domahidi, E. Chu, and S. Boyd, "Ecos: An socp solver for embedded systems," in *European control conference (ECC)*, IEEE, 2013.