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

PRISM: Recurrent Neural Networks and Presolve Methods for Fast Mixed-integer Optimal Control

Cauligi, Abhishek; Chakrabarty, Ankush; Di Cairano, Stefano; Quirynen, Rien

TR2022-039 May 19, 2022

Abstract

While mixed-integer convex programs (MICPs) arise frequently in mixed-integer optimal control problems (MIOCPs), current state-of-the-art MICP solvers are often too slow for real-time applications, limiting the practicality of MICP-based controller design. Although supervised learning has been proposed to hasten the solution of MICPs via convex approximations, they are not designed to scale well to problems with >100 decision variables. In this paper, we present PRISM: Presolve and Recurrent network-based mixed-Integer Solution Method, to leverage deep recurrent neural network (RNN) architectures such as long short-term memory (LSTMs) networks, in conjunction with numerical optimization tools to enable scalable acceleration of MICPs arising in MIOCPs. Our key insight is to learn the underlying temporal structure of MIOCPs and to combine this with presolve routines employed in MICP solvers. We demonstrate how PRISM can lead to significant performance improvements, compared to branch-and-bound (B&B) methods and to existing supervised learning techniques, for stabilizing a cart-pole with contact dynamics, and a motion planning problem under obstacle avoidance constraints.

Learning for Dynamics and Control Conference (L4DC) 2022

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

PRISM: Recurrent Neural Networks and Presolve Methods for Fast Mixed-integer Optimal Control

Abhishek Cauligi ABHISHEK.S.CAULIGI@JPL.NASA.GOV
Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305, USA.

Ankush Chakrabarty ACHAKRABARTY@IEEE.ORG
Stefano Di Cairano DICAIRANO@MERL.COM
Rien Quirynen QUIRYNEN@MERL.COM
Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA.

Editors: R. Firoozi, N. Mehr, E. Yel, R. Antonova, J. Bohg, M. Schwager, M. Kochenderfer

Abstract

While mixed-integer convex programs (MICPs) arise frequently in mixed-integer optimal control problems (MIOCPs), current state-of-the-art MICP solvers are often too slow for real-time applications, limiting the practicality of MICP-based controller design. Although supervised learning has been proposed to hasten the solution of MICPs via convex approximations, they are not designed to scale well to problems with $>$100 decision variables. In this paper, we present PRISM: Presolve and Recurrent network-based mixed-Integer Solution Method, to leverage deep recurrent neural network (RNN) architectures such as long short-term memory (LSTMs) networks, in conjunction with numerical optimization tools to enable scalable acceleration of MICPs arising in MIOCPs. Our key insight is to learn the underlying temporal structure of MIOCPs and to combine this with presolve routines employed in MICP solvers. We demonstrate how PRISM can lead to significant performance improvements, compared to branch-and-bound (B&B) methods and to existing supervised learning techniques, for stabilizing a cart-pole with contact dynamics, and a motion planning problem under obstacle avoidance constraints.

Keywords: Mixed-integer convex programming, optimal control, deep learning, learning for optimization, recurrent architectures, LSTM, contact dynamics, motion planning, MPC.

1. Introduction

The recent development of embedded optimization solvers has led to the proliferation of optimization-based planning and control in aerospace, robotic, and automotive applications (Di Cairano and Kolmanovsky, 2018). MICPs allow one to formulate and solve a broad range of optimal control problems that arise, e.g., in switched dynamical systems (Chen et al., 2021), discrete/quantized actuation (Walsh et al., 2016), motion planning with obstacle avoidance (Landry et al., 2016), logic rules and temporal logic specifications (Sahin et al., 2020), among others. State of the art MICP solvers are based on B&B methods (Mosek APS; Gurobi Optimization, LLC, 2020), using advanced presolve techniques (Achterberg et al., 2019) to accelerate the B&B solution process. B&B methods can provide exact solutions for MIOCPs; however, in the worst-case scenario, these algorithms can incur an exponential number of convex relaxations (Floudas, 1995). Therefore, B&B methods for MIOCPs are not yet readily deployed in applications with relatively fast sampling rates and limited computational resources. Although there exist heuristic methods to solving MICPs such as rounding fractional integer solutions or terminating B&B early, these approaches can lead to low-quality solutions and thus degrade closed-loop performance.

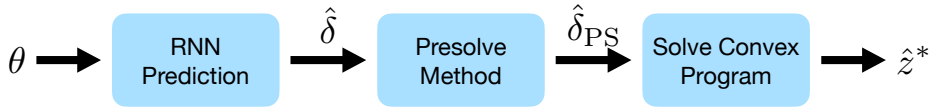

Figure 1: Schematic of PRISM approach for solving MIOCPs with RNN and presolve. The problem parameters θ are used to make a prediction for the candidate binary optimizer $\hat{\delta}$, which is then updated to $\hat{\delta}_{\mathrm{PS}}$ using presolve and the continuous optimizer $\hat{z}^\star$ recovered.

Related work: To maintain high-quality solutions without exorbitant computational expenditure, recent work has adopted machine learning for efficient optimization (Bengio et al., 2021). For continuous optimization, a recent approach has been to learn warm starts for the continuous variables in trajectory optimization problems (Zhang et al., 2019; Wang et al., 2020; Chen et al., 2022). For MIOCPs, using supervised learning to replicate solutions of B&B methods procured offline and inferring these solutions online at high speed has resulted in dramatic improvement of solution times (Masti and Bemporad, 2019; Bertsimas and Stellato, 2019; Srinivasan et al., 2021). Alternatively, supervised disagreement learning has been proposed to perform real-time solver selection for MIOCPs in (Chakrabarty et al., 2021). In addition, reinforcement learning has been used to learn tree-search policies for B&B (Mern et al., 2021; Khalil et al., 2022), but these methods may still require enumerating the full B&B tree in the worst case.

For parametric MIOCPs, supervised learning can approximate the mapping from problem parameters to the binary variable solution of an MICP directly. Multiple techniques have been explored to formulate this supervised learning problem, including the use of kernels and non-parametric classification methods (Chakrabarty et al., 2016; Zhu and Martius, 2020), deep neural networks for regression (Masti and Bemporad, 2019; Srinivasan et al., 2021), and classification (Bertsimas and Stellato, 2019; Cauligi et al., 2020, 2022). Löhr et al. (2020) proposed a supervised learner that only predicts the binary variables associated with the first time step; however, this is likely to incur significant suboptimality. Alternatively, the policy induced by an exact solver for the MIOCP could be imitated directly using deep neural networks (Srinivasan et al., 2021), or a value function could be learned (Landry et al., 2021) by leveraging data obtained via offline simulations. A drawback of these approaches is that they are effective for small-size problems, but require extensive data collection or sampling for high-dimensional MIOCPs. Further, these techniques do not consider how the convex relaxation is solved downstream, i.e., agnostic to the use of numerical optimization methods such as heuristics or presolve (Hespanhol et al., 2019; Liang et al., 2021).

Our Contributions: We propose presolve and recurrent network-based mixed-integer solution method (PRISM), an approach that integrates presolve techniques and a supervised learning framework that learns the underlying temporal structure inherent to MIOCPs to efficiently solve long-horizon MIOCPs with hundreds of binary decision variables. In the offline data collection phase of PRISM, we solve a set of MICPs for representative problem parameter values θ and collect the set of binary optimizers $\delta^\star$. Subsequently, we train an RNN, such as LSTM, to learn a mapping from problem parameters θ to binary optimizers $\delta^\star$. As shown in Figure 1, given new problem parameters θ, the solution phase of PRISM leverages the trained RNN to predict a binary solution candidate $\hat{\delta}$. This $\hat{\delta}$ is then followed by a presolve routine to provide an updated candidate $\hat{\delta}_{\mathrm{PS}}$. This candidate is used in the solution of a subsequent convex program to quickly generate a feasible solution for the original MICP. We demonstrate the efficacy of PRISM through numerical experiments on two case studies: a cart-pole with soft contact system and motion planning in the presence of obstacles.

Our results demonstrate that PRISM computes solutions an order of magnitude faster than existing commercial solvers while introducing only up to 5% suboptimality.

2. Preliminaries

This section provides a brief discussion on the parametric MIOCPs studied in this work, how recurrent layers can be used to learn sequential information in control action sequences, and how to improve the inference of deep neural networks via presolve techniques prevalent in MICP solvers.

2.1. Parametric MIOCPs

Given a vector of problem parameters $\theta \in \mathbf{R}^{n_p}$, a parametric MIOCP can be written as

$$\begin{array}{ll} \underset{x_{0:N}, u_{0:N}, \delta_{0:N}}{\text{minimize}} & \sum_{t=0}^{N} g_t(x_t, u_t, \delta_t; \theta) \\ \text{subject to} & x_0 = x_{\text{init}}(\theta) \\ & x_{t+1} = \psi_t(x_t, u_t, \delta_t; \theta), \quad t = 0, \ldots, N-1 \\ & f_{t,i}(x_t, u_t, \delta_t; \theta) \leq 0, \quad t = 0, \ldots, N, \quad i = 1, \ldots, n_{\text{f}} \\ & \delta_t \in \{0,1\}^{n_\delta}, \quad t = 0, \ldots, N, \end{array} \tag{1}$$

where the state $x_t \in \mathbf{R}^{n_x}$ and control $u_t \in \mathbf{R}^{n_u}$ are the continuous decision variables and $\delta_t \in \{0,1\}^{n_\delta}$ are the binary decision variables. For simplicity, we denote $\delta \in \{0,1\}^{N_\delta}$, where $N_\delta = (N+1)n_\delta$, as the stacked binary decision variable vector, i.e., $\delta_{0:N}$. Similarly, $z \in \mathbf{R}^{(N+1)(n_x+n_u)}$ refers to the continuous decision variables, i.e., $(x_{0:N}, u_{0:N})$. Here, the stage cost $g_t(\cdot)$, terminal cost $g_N(\cdot)$, and inequality constraints $f_{t,i}(\cdot)$ are assumed to be convex functions. The dynamics $\psi_t(\cdot)$ are assumed to be linear, e.g., possibly using a mixed-integer formulation of piecewise affine (PWA) dynamics (Marcucci and Tedrake, 2019). The objective function and constraints are functions of the parameter vector $\theta \in \Theta$, where $\Theta \subseteq \mathbf{R}^{n_p}$ is the admissible set of parameters. Parameters could include, for instance, a given initial state, target state, obstacle coordinates, to name a few.

In (1), the binary decision variables δ are the "complicating" variables: i.e., if δ are fixed, then the remaining problem is convex and much easier to solve. Concretely, if the optimal binary solution $\delta^\star$ for (1) is fixed, the continuous optimizers $x^\star_{0:N}$ and $u^\star_{0:N}$ can be computed by solving a single convex optimization problem,

$$\begin{array}{ll} \underset{x_{0:N}, u_{0:N}}{\text{minimize}} & \sum_{t=0}^{N} g_t(x_t, u_t, \delta^\star_t; \theta) \\ \text{subject to} & x_0 = x_{\text{init}}(\theta) \\ & x_{t+1} = \psi_t(x_t, u_t, \delta^\star_t; \theta), \quad t = 0, \ldots, N-1 \\ & f_{t,i}(x_t, u_t, \delta^\star_t; \theta) \leq 0, \quad t = 0, \ldots, N, \quad i = 1, \ldots, n_{\text{f}}, \end{array} \tag{2}$$

which is considerably cheaper to solve than the MIOCP in (1). An interesting approach to solving problems of the form (1) involves learning a map between the vector of problem parameters θ and the discrete optimizer $\delta^\star$ (Masti and Bemporad, 2019; Cauligi et al., 2020), such that a convex program of the form (2) needs to be solved instead. Although such approaches lack the completeness guarantees provided by B&B, learning an effective mapping to identify $\delta^\star$ would allow the user to avoid having to search through a B&B tree. Another limitation of such approaches is that they neglect the sequential decision making structure of the MIOCP. For example, the binary solution trajectory $\delta^\star_{0:N}$ for (1) may often attain identical values for part of the trajectory in time. In such cases, incorporating the temporal nature of the binary solution into the formulation may improve the performance of learning a mapping between θ and $\delta^\star$.

2.2. Recurrent Neural Network Architectures

To this end, we propose using RNN architectures to incorporate the sequential decision making structure of MIOCPs in learning the mapping between parameter vector θ and the binary optimizer $\delta^\star$. Concretely, we learn a mapping between θ and $\delta_t^\star$ at each step of the RNN for $t = 0, \ldots, N$. We choose LSTMs for our network architecture (Hochreiter and Schmidhuber, 1997), but PRISM can be easily extended to accommodate, e.g., more traditional RNNs or gated recurrent units (GRUs). LSTMs have allowed for tremendous strides in time-series forecasting and temporal predictions and have demonstrated effectiveness in controls and robotics applications (Chung et al., 2015; Morton et al., 2017). Herein, the LSTM learns the temporal structure of the binary optimizer for the MIOCP. The primary advantage of considering the temporal structure of the binary solution is that it can reduce the number of target labels in supervised learning. To illustrate, consider that for the problem (1), the maximum number of target labels is 2^{N_δ}, where $N_\delta = (N+1)n_\delta$, for n_δ binary variables. Conversely, decomposing the binary variables over time leads to at most 2^{n_δ} target labels, which is substantially lower than 2^{N_δ} for large N. Having fewer target labels for a given number of training samples results in improved learning quality by the LSTM.

2.3. Presolve Techniques

Presolve techniques can tighten constraints or prune a subset of decision variables by fixing them to predetermined values, e.g., using domain propagation, coefficient strengthening, and probing. For MICPs, the development of effective presolve methods in commercial solvers has proven crucial in accelerating computation times (Achterberg et al., 2019). As an example of how presolve routines can be applied, consider an MICP including the inequality constraints $x_1 \leq M(\theta)\delta_1$ and $x_1 \geq 1$, where $x_1 \in \mathbf{R}$, $\delta_1 \in \{0, 1\}$, and $M(\theta) > 0$ is a given upper bound value. We see clearly that δ_1 cannot attain the value 0 as this would require both $x_1 \geq 1$ and $x_1 \leq 0$ simultaneously. This implies $\delta_1 = 1$ and we can therefore remove ("prune") δ_1 from the set of decision variables. Thus, pruning reduces the complexity of the original MICP. We refer to the MICP in (1) as $\mathcal{P}(\theta)$, and we use the compact notation $\mathcal{P}(\theta, \delta_\mathcal{I} = \hat{\delta})$ to denote the problem (1) after fixing $\delta_i = \hat{\delta}_i, i \in \mathcal{I}$ for an index set $\mathcal{I}$, in the following simplified definition of a presolve routine.

Definition 1 (Presolve) *Given the problem $\mathcal{P}(\theta)$ and a set of binary values $\{\hat{\delta}_i\}_{i\in\mathcal{I}}$ for the index set $\mathcal{I} \subseteq \{1, \ldots, N_\delta\}$, the presolve routine computes*

$$\{\texttt{flag}, \hat{\delta}^+, \mathcal{I}^+\} \leftarrow \textit{Presolve}(\mathcal{P}(\theta), \hat{\delta}, \mathcal{I}), \tag{3}$$

resulting in an updated set of binary values $\{\hat{\delta}_i^+\}_{i\in\mathcal{I}^+}$ for the index set $\mathcal{I}^+ \subseteq \{1, \ldots, N_\delta\}$, for which the following conditions are satisfied:

1. *The new set of indices includes at least the original set, i.e., $\mathcal{I} \subseteq \mathcal{I}^+$.*
2. *Problem $\mathcal{P}(\theta, \delta_{\mathcal{I}^+} = \hat{\delta}^+)$ is infeasible or unbounded, i.e., $\texttt{flag} = \texttt{False}$, if and only if $\mathcal{P}(\theta, \delta_\mathcal{I} = \hat{\delta})$ is infeasible or unbounded.*
3. *Any feasible or optimal solution of $\mathcal{P}(\theta, \delta_{\mathcal{I}^+} = \hat{\delta}^+)$ can be mapped to a feasible or optimal solution of $\mathcal{P}(\theta, \delta_\mathcal{I} = \hat{\delta})$, and their objective values are identical.*

A presolve routine applied to a root node in B&B corresponds to Definition 1 with $\mathcal{I} = \emptyset$. In general, presolve cannot prune all of the binary (or integer) decision variables, but it can often lead to a reduced problem that is significantly faster to solve.

3. PRISM: An LSTM-Presolve Pipeline for Solving MIOCPs

3.1. Offline Data Generation

Algorithm 1 provides an overview for the offline phase of PRISM. First, a set of problem parameter values $\{\theta^i\}_{i=1}^M$ representative of the control problem are sampled, where M is the number of samples drawn from Θ. For each parameter value θ^i, the MICP is solved and, if the problem is feasible, the continuous and binary optimizers $(z^{i,\star}, \delta^{i,\star})$ are returned (Line 3). A binary optimizer set Δ collects the unique binary variable solutions and, for each time step $t \in [0, N]$ associated with the MIOCP horizon, the binary solution $\delta_t^{i,\star}$ for time t is added to Δ if it is not already included, i.e., $\Delta = \cup_{i,t}\delta_t^{i,\star}$. The class label y_t^i associated with $\delta_t^{i,\star}$ is identified (Line 6) and $\left\{\left(\theta^i, t, \delta_t^{i,\star}, y_t^i\right)\right\}_{t\in[0,N]}$ is then appended to the training data $\mathcal{D}$ (Line 8). Finally, an LSTM neural network h_ϕ is trained using the data and labels contained in $\mathcal{D}$ (Line 11).

Algorithm 1 PRISM Offline Supervision

Require: Batch of parameter values $\{\theta^i\}_{i=1,\dots,M}$, MIOCP in (1)
1: Initialize binary optimizer set $\Delta \leftarrow \{\}$, train set $\mathcal{D} \leftarrow \{\}$, and $k \leftarrow 0$
2: **for** each θ^i **do**
3: $(z^{i,\star}, \delta^{i,\star}, \texttt{feasible}) \leftarrow \text{SolveMICP}(\theta^i)$ in (1)
4: **if** `feasible == True` **then**
5: **for** $t \in [0, N]$ **do**
6: $\Delta \leftarrow \Delta \cup \delta_t^{i,\star}$ and identify class label y_t^i for $\delta_t^{i,\star}$ in Δ
7: **end for**
8: Add $\left\{\left(\theta^i, t, \delta_t^{i,\star}, y_t^i\right)\right\}_{t\in[0,N]}$ to $\mathcal{D}$
9: **end if**
10: **end for**
11: Choose network weights ϕ which minimize the training loss
12: **return** h_ϕ, Δ

For the inference step of the network h_ϕ, a given value for the parameter vector θ is first processed through a feedforward block FC_1 before being passed as input to the LSTM. The output hidden state h_t from the LSTM is then passed through a feedforward block FC_2 to generate the binary candidate $\hat{\delta}_t$ for that time step, and as input to the LSTM block at the next time step. We note that FC_1 and FC_2 do not vary per time step, but could be adjusted to accommodate problems with a different number of parameters or binary decision variables at each step (e.g., signal temporal logic). The predictions $\hat{\delta}_t$ for each time step $t \in [0, N]$ are used to construct a candidate binary optimizer $\hat{\delta}$. Training h_ϕ could be cast both as a classification or as a regression problem, and PRISM is agnostic to whether h_ϕ performs classification or regression. Furthermore, the developments in the remainder of this paper generalize to both frameworks.

Remark 1 (h_ϕ as Regressor) *If the network h_ϕ being trained is a regressor, that is, the network is trained to directly predict the value $\hat{\delta}_t$, then the output dimension for each time step is n_δ and the training loss used is binary cross-entropy loss with logit activations applied to the output layer.*

Remark 2 (h_ϕ as Classifier) *If the network h_ϕ denotes a classifier, we train it to select the best value $\hat{\delta}_t \in \Delta$; in this case, the output dimension is the cardinality $|\Delta|$ of the binary optimizer set and the training loss function is a cross-entropy loss.*

3.2. Online: Inference Procedure and Iterative Presolve

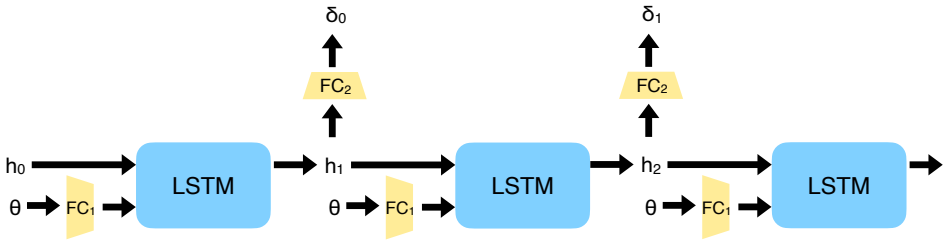

Figure 2: The PRISM inference procedure with the LSTM block in blue, the fully connected feed-forward block FC in yellow, and h_t the hidden state of the LSTM at time t.

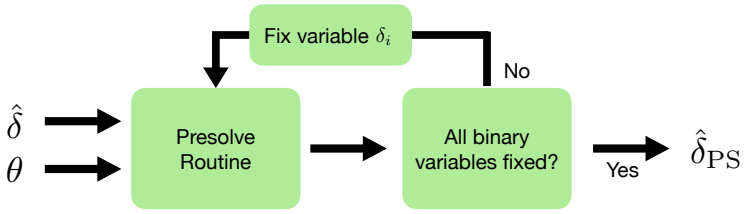

Figure 3: Iterative presolve method to fix binary variables in MIOCP.

The inference procedure for PRISM is detailed in Algorithm 2. Given a vector of problem parameters θ, the inference procedure consists of a forward pass through the LSTM cell for each time step of the MIOCP as shown in Figure 2. The LSTM predictions $\hat{\delta}_t$ are used to construct a candidate binary optimizer $\hat{\delta}$ (Line 1). However, rather than directly applying $\hat{\delta}$, the iterative presolve routine is called to adjust the neural network prediction $\hat{\delta}$ to increase the rate of feasibility. The iterative presolve uses $\hat{\delta}$ to generate a new binary prediction $\hat{\delta}_{\text{PS}}$, which is used to fix the binary variables in a convex approximation (2) of the original MICP (Line 6). This procedure continues for n_{evals} binary predictions and the best feasible solution is stored (Lines 6-9).

Algorithm 2 PRISM Online Variable Fixing Approach

Require: Problem parameters θ, set Δ, trained neural network h_ϕ, n_{evals}

1: Generate candidate binary solutions $\{\hat{\delta}^j\}_{j=1,\dots,n_{\text{evals}}}$ using network h_ϕ
2: $z^\star \leftarrow \emptyset, \delta^\star \leftarrow \emptyset, J^\star \leftarrow \infty$, `success` $\leftarrow$ `False`
3: **for** $j = 1, \dots, n_{\text{evals}}$ **do**
4: $(\texttt{flag}, \hat{\delta}^j_{\text{PS}}) \leftarrow \text{IterativePresolve}(\hat{\delta}^j, \theta)$ in Alg. 3
5: **if** `flag` == `True` **then**
6: $(\bar{z}, \bar{J}, \texttt{feasible}) \leftarrow \text{SolveCP}(\theta, \hat{\delta}^j_{\text{PS}})$ in (2)
7: **if** `feasible` == `True` and $\bar{J} < J^\star$ **then**
8: $J^\star \leftarrow \bar{J}, z^\star \leftarrow \bar{z}, \delta^\star \leftarrow \hat{\delta}^j_{\text{PS}}$, `success` $\leftarrow$ `True`
9: **end if**
10: **end if**
11: **end for**
12: **return** $(\texttt{success}, z^\star, \delta^\star, J^\star)$

Algorithm 3 Iterative Presolve Routine

Require: Candidate binary solution $\hat{\delta}$, problem parameters θ
1: Initialize set of pruned binary variables $\mathcal{I} \leftarrow \emptyset$ and values $\hat{\delta}_{\text{PS}} \leftarrow \emptyset$
2: **for** $i \in \{1, \dots, N_\delta\}$ **do**
3: $\{\texttt{flag}, \hat{\delta}_{\text{PS}}, \mathcal{I}\} \leftarrow \text{Presolve}(\mathcal{P}(\theta), \hat{\delta}_{\text{PS}}, \mathcal{I})$
4: **if** $\texttt{flag} == \texttt{False}$ **then**
5: **return** $(\texttt{False}, \hat{\delta}_{\text{PS}})$
6: **end if**
7: **if** $|\mathcal{I}| == N_\delta$ **then**
8: **return** $(\texttt{True}, \hat{\delta}_{\text{PS}})$
9: **end if**
10: $j \leftarrow \texttt{VarSelect}(\{1, \dots, N_\delta\} \setminus \mathcal{I})$
11: Fix binary variable $\hat{\delta}_{\text{PS},j} = \hat{\delta}_j$, $\mathcal{I} \leftarrow \mathcal{I} \cup \{j\}$
12: **end for**

As illustrated in Figure 3, an iterative presolve method is used to correct predictions made by the supervised learner h_ϕ, given that the presolve operation can provide additional fixings that preserve feasibility and optimality; see Definition 1. The implication is that if presolve prunes a subset of the binary variables, then the supervised learner only needs to fix the remaining binary decision variables. This allows for improvements of the supervised learner in instances when presolve prunes a binary variable δ_i for which the supervised learner made an incorrect prediction that would have resulted in an infeasible solution. Algorithm 3 describes the proposed iterative presolve routine given a candidate binary solution $\hat{\delta}$ for a particular vector of problem parameters θ. Let $\mathcal{I}$ be the index set of pruned decision variables (i.e., δ_i for $i \in \mathcal{I}$ are fixed), $|\mathcal{I}|$ is the number of pruned binary decision variables, and $\mathcal{I}$, $\hat{\delta}_{\text{PS}}$ are initialized as empty sets (Line 1). The presolve routine is then called (Line 3) and returns whether infeasibility was detected, an updated set of indices $\mathcal{I}$, and corresponding values $\hat{\delta}_{\text{PS}}$. Algorithm 3 terminates if all of the binary variables have been fixed (Line 7) or if infeasibility is detected (Line 4). Otherwise, one of the remaining free binary variables δ_j for $j \in \{1, \dots, N_\delta\} \setminus \mathcal{I}$ is selected and fixed to the value proposed by the supervised learner, i.e., $\hat{\delta}_{\text{PS},j} = \hat{\delta}_j$ (Line 10-11). Here, we denote `VarSelect` as the procedure to choose the binary variable δ_j to fix next, which is closely related to the variable selection policy in B&B routines. Algorithm 3 is an iterative procedure, which calls the presolve routine (Line 3) at each iteration. One could also skip the presolve call when a particular limit on the computation time has been reached, i.e., such that all remaining free binary variables are fixed directly by the supervised learner (Line 11).

4. Numerical Experiments

We validate PRISM on two benchmark problems shown in Figure 4: (1) an underactuated cart-pole with soft contacts and (2) motion planning in the presence of obstacles.

We demonstrate the efficacy of using an LSTM architecture and compare against the fully connected neural network regressor from (Masti and Bemporad, 2019) and classifier from (Cauligi et al., 2022). We used the `PyTorch` machine learning library (Paszke et al., 2017) to implement our neural network models with the ADAM optimizer for training. The optimization problems are modeled and solved using Gurobi (Gurobi Optimization, LLC, 2020). For each system, the training set consists of 90,000 MICP solutions corresponding to problem parameters θ representative of the control task. The test set consists of 10,000 additional MICPs corresponding to θ sampled from the same parameter space. To assess the quality of our proposed method for each MICP in the test

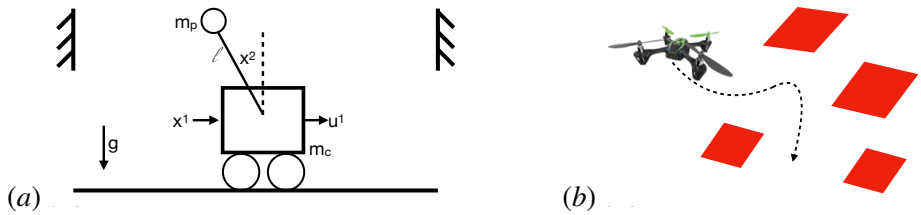

Figure 4: Benchmark problems. (a) Cart-pole with contact system. (b) Robot motion planning.

set, we report the percentage of failure to find a feasible solution, the computation time, and the suboptimality of the feasible solution found by each approach with respect to the globally optimal solution. We present results for predicting $\hat{\delta}$ by both a classifier and a regressor to highlight that the use of PRISM enables both choices. All reported computation times include both the inference time of the neural network architecture and the convex program solution time by Gurobi.

4.1. Cart-Pole with Soft Walls

We study a cart-pole with soft contacts shown in Figure 4, a problem representative of underactuated, multi-contact control problems (Aydinoglu et al., 2020; Mordatch et al., 2012). Although MICPs form an attractive framework for modeling such problems, controllers that can react and plan online for such systems are rarely real-time feasible or near-optimal (Marcucci et al., 2017). The MIOCP entails using the horizontal forces of the cart to regulate an inverted pendulum towards a goal state x_g and a vector of binary variables $\delta_t \in \mathbf{R}^4$ is used to model the forces generated when the pendulum strikes the side walls; see Figure 4(a). We select a horizon length of $N = 50$, resulting in $N_\delta = 204$ binary variables in this mixed-integer quadratic program (MIQP). The problem parameters $\theta \in \mathbf{R}^4$ for this problem consist of the initial condition x_{init}.

Results: We compare our proposed LSTM architecture in PRISM against a feedforward neural network classifier and regressor to demonstrate the effectiveness of the recurrent architecture for this problem. Through a cursory hyperparameter search, we empirically found that network performance was more sensitive to the network width compared to the depth. For the LSTM, the network follows what is shown in Figure 2 and consists of a one hidden layer feedforward network with 128 neurons for FC_1 and FC_2 and a three-layer LSTM cell with a 128-dimensional hidden layer. To maintain parity, the feedforward network is designed with three hidden layers and 128 neurons per layer. For this problem, we further benchmark against a fully connected network where a one-hot encoding of size $\mathbf{R}^{N+1}$ is used to indicate which δ_t is being queried. By using δ_t as the target labels, the time-wise decomposition in PRISM yielded 7 unique target labels as compared to 2687 target labels when predicting the sequence of binary decision variables $\delta_{0:N}$ directly.

The numerical results are shown in Table 1. We begin by comparing the percentage of problems in the test set for which each approach fails to find a feasible solution. We immediately see two broad trends: (1) the classifier outperforms the regressor, and (2) the use of an LSTM allows both the classifier and the regressor to compute more feasible solutions. Concretely, for the classifier, the feedforward network fails for 7.9% of the test set compared to 6.0% for the PRISM classifier. The performance improvement allowed by the LSTM is more significant for the regressor, with 24.7% infeasible solutions with a fully connected regressor compared to only 13.9% infeasible solutions with the PRISM regressor. We see that this performance improvement comes with a slight increase in computation time for the LSTM, which requires on average approximately 20% longer for the

	Infeasibility [%]	Suboptimality [%]	Time [ms]
Gurobi B&B	0.0	0.0 (0.0, 0.0)	611.9 (186, 1248)
Gurobi B&B (20 ms limit)	77.8	2.3 (0.0, 21.2)	20.9 (20.1, 22.9)
Regressor	24.7	1.1 (0.0, 7.3)	10.7 (9.5, 12.2)
Regressor OH Time	27.7	0.7 (0.0, 3.6)	10.7 (9.5, 12.1)
PRISM Regressor (LSTM)	13.9	0.4 (0.0, 1.6)	12.9 (11.6, 14.4)
Classifier	7.9	0.4 (0.0, 1.0)	10.8 (9.6, 12.3)
Classifier OH Time	14.2	0.3 (0.0, 0.6)	10.8 (9.5, 12.2)
PRISM Classifier (LSTM)	6.0	0.1 (0.0, 0.1)	12.9 (11.6, 14.4)

Table 1: Illustration of the benefit of recurrent architectures on a simulation study using the cart-pole with contact system with $N = 50$, and PRISM is highlighted in blue. The values for suboptimality and time include the average and 5%, 95% percentiles between parentheses.

PRISM regressor and classifier to compute a forward pass and solve the convex relaxation. PRISM also improves upon the one-hot encoding (OH Time) approach, which results in more than twice the failure rate of the PRISM regressor and classifier, respectively. We also compare against the Gurobi B&B solver timed out after 20ms and note that doing so leads to 78% infeasibility on the test set. We attribute the improved performance of the LSTM approaches to having only 7 target labels by using δ_t rather than $\delta_{0:N}$ as the target labels, exploiting the temporal aspect of the MIOCP and leading to improved supervision of the LSTM for this benchmark problem.

The rate of infeasibility for the results in Table 1 could be reduced further by generating multiple candidate binary solutions for the classifier architectures as shown in (Bertsimas and Stellato, 2019), i.e., $n_{\text{evals}} > 1$ in Algorithm 2. For example, the rate of infeasibility for our PRISM classifier is 6.0, 4.1, 3.0 or 2.4 % for $n_{\text{evals}} = 1$, 2, 4 or 8, respectively. The performance of the proposed iterative presolve method in Algorithm 3 will be illustrated on the next numerical case study.

4.2. Motion Planning

A fundamental problem in robotics is motion planning in the presence of obstacles and MICPs can be used to model the inherently combinatorial nature of obstacle avoidance (LaValle, 2006). In this work, we study the problem of a planar robot navigating a workspace with axis-aligned obstacles. A popular approach for solving the motion planning problem is through numerical optimization techniques and the inclusion of binary variables to enforce the non-convex $x_t \in \mathcal{X}_{\text{safe}}$ safety constraint (Landry et al., 2016). Following the formulation from (Cauligi et al., 2022), the MIOCP for planning a collision-free trajectory towards a goal state x_g is an MIQP with $4N_{\text{obs}}(N+1)$ binary decision variables. We choose a number of control intervals $N = 20$ and a number of obstacles $N_{\text{obs}} = 4$ for a total of $N_\delta = 336$ binary variables. The parameters $\theta \in \mathbf{R}^{20}$ for this problem are comprised of the initial state x_{init} and the positions of the four obstacles.

Results: For this system, the feedforward networks consist of three hidden layers with 256 neurons each and the appropriately sized output dimension. The LSTM in PRISM consists of a one hidden layer feedforward network with 256 neurons for FC_1 and FC_2 and a three-layer LSTM cell with a 128-dimensional hidden layer. We additionally include the task-specific logical strategies for this system presented in (Cauligi et al., 2022). In this one-hot obstacle (OHO) approach, PRISM learns the binary variable assignment separately for each of the N_{obs} obstacles and uses an LSTM with input dimension adjusted to include the one-hot encoding of size $\mathbf{R}^{N_{\text{obs}}}$, which denotes the ob-

	Infeasibility [%]		Suboptimality [%]		Time [ms]	
	Standard	Presolve	Standard	Presolve	Standard	Presolve
Gurobi B&B	–	0.0	–	0.0 (0.0, 0.0)	–	102.7 (33, 239)
B&B (20 ms)	–	87.4	–	247.1 (68, 575)	–	20.6 (20.1, 23.2)
Regressor	83.6	1.8	2.3 (0.0, 4.6)	11.4 (0.0, 49.8)	3.0 (2.9, 3.1)	9.4 (6.4, 12.7)
OHO	70.5	1.8	6.1 (0.0, 22.5)	9.3 (0.0, 42.3)	3.1 (3.0, 3.3)	9.5 (6.3, 13.1)
PRISM	72.1	2.3	4.8 (0.0, 5.4)	5.5 (0.0, 22.7)	4.5 (4.3, 4.7)	11.0 (7.8, 14.4)
PRISM OHO	29.5	1.9	1.5 (0.0, 4.5)	2.3 (0.0, 8.9)	3.7 (3.5, 3.9)	10.1 (6.7, 13.6)
Classifier	54.7	2.6	3.1 (0.0, 16.9)	29.2 (0.0, 151)	3.2 (3.0, 3.4)	9.0 (6.2, 12.3)
OHO	44.0	1.9	1.8 (0.0, 6.0)	8.2 (0.0, 45.3)	3.2 (3.0, 3.4)	9.2 (6.1, 12.6)
PRISM	16.0	1.5	0.5 (0.0, 1.7)	2.7 (0.0, 8.9)	4.6 (4.4, 4.8)	10.7 (7.5, 14.1)
PRISM OHO	16.5	1.1	0.5 (0.0, 1.2)	2.2 (0.0, 6.0)	3.7 (3.5, 3.9)	9.7 (6.6, 13.2)

Table 2: Results with and without the iterative presolve method in Alg. 3 for the motion planning example with $N = 20$, including the one-hot obstacle (OHO) approach. The values for suboptimality and time include the average and 5%, 95% percentiles between parentheses.

stacle for which the variables are predicted, and output dimension of 4 for each LSTM output layer. For this problem, the number of unique target labels with PRISM was 244 for δ_t as compared to 42406 target labels for $\delta_{0:N}$ without the time-wise decomposition. We illustrate the computational efficiency of the iterative presolve method in Algorithm 3 based on the embeddable C code implementation of a presolve step (Line 3) from (Hespanhol et al., 2019; Liang et al., 2021).

The results for this benchmark example are shown in Table 2. As the primary point of comparison, we compare the performance of each approach with and without the presolve routine integrated. Without the presolve routine, we see once again that an LSTM architecture leads to performance improvements for both the regressor and classifier, with 11.5% and 38.7% more feasible solutions found for the PRISM regressor and classifier, respectively. With the presolve routine, the percentage of feasible solutions found for all approaches is dramatically improved, with a worst-case infeasibility of less than 3%. However, the quality of the solution found with presolve differs across the approaches and the suboptimality of PRISM is considerably lower than those found by the feedforward networks. Table 2 also demonstrates that PRISM can be effectively integrated with the one-hot obstacle (OHO) approach and yields higher quality solutions than OHO alone. Thus, we see that presolve is crucial in enabling the application of supervised learning techniques for the motion planning problem as it reduces failure rates by an order of magnitude while maintaining the ability to find near optimal solutions and only a slight slow down in computation time.

5. Conclusion

In this work, we showed how leveraging recurrent neural network architectures in conjunction with presolve techniques from numerical optimization leads to dramatic improvements among existing supervised learning approaches in terms of both feasibility and optimality. We demonstrated the efficacy of PRISM on two standard robot planning and control tasks, cart-pole with contact system and motion planning in the presence of obstacles. Future work may focus on exploring end-to-end learning based frameworks where the decisions made by the presolve routine to prune particular binary variables are included in the loss function used to train the neural network.

References

T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473 – 506, 2019.

A. Aydinoglu, V. M. Preciado, and M. Posa. Contact-aware controller design for complementarity systems. In *Proc. IEEE Conf. on Robotics and Automation*, 2020.

Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'Horizon. *European Journal of Operations Research*, 290(2):405–421, 2021.

D. Bertsimas and B. Stellato. Online mixed-integer optimization in milliseconds, 2019. Available at https://arxiv.org/abs/1907.02206.

A. Cauligi, P. Culbertson, B. Stellato, D. Bertsimas, M. Schwager, and M. Pavone. Learning mixed-integer convex optimization strategies for robot planning and control. In *Proc. IEEE Conf. on Decision and Control*, 2020.

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*, 7(2): 1447–1454, 2022.

A. Chakrabarty, G. T. Buzzard, and S. H. Żak. Output-tracking quantized explicit nonlinear model predictive control using multiclass support vector machines. *IEEE Transactions on Industrial Electronics*, 64(5):4130–4138, 2016.

A. Chakrabarty, R. Quirynen, D. Romeres, and S. Di Cairano. Learning disagreement regions with deep neural networks to reduce practical complexity of mixed-integer MPC. In *Proc. IEEE Conf. on Systems, Man, and Cybernetics*, 2021.

C. Chen, P. Culbertson, M. Lepert, M. Schwager, and J. Bohg. TrajectoTree: Trajectory optimization meets tree search for planning multi-contact dexterous manipulation. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2021.

S. W. Chen, T. Wang, N. Atanasov, V. Kumar, and M. Morari. Large scale model predictive control with neural networks and primal active sets. *Automatica*, 135:109947, 2022.

J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. In *Conf. on Neural Information Processing Systems*, 2015.

S. Di Cairano and I. Kolmanovsky. Real-time optimization and model predictive control for aerospace and automotive applications. In *American Control Conference*, 2018.

C. A. Floudas. *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford Univ. Press, 1995.

Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, 2020. Available at http://www.gurobi.com.

P. Hespanhol, R. Quirynen, and S Di Cairano. A structure exploiting branch-and-bound algorithm for mixed-integer model predictive control. In *European Control Conference*, 2019.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

E. B. Khalil, P. Vaezipoor, and B. Dilkina. Finding backdoors to integer programs: A Monte Carlo Tree Search framework. In *Proc. AAAI Conf. on Artificial Intelligence*, 2022.

B. Landry, R. Deits, P. R. Florence, and R. Tedrake. Aggressive quadrotor flight through cluttered environments using mixed integer programming. In *Proc. IEEE Conf. on Robotics and Automation*, 2016.

B. Landry, H. Dai, and M. Pavone. Seagul: Sample efficient adversarially guided learning of value functions. In *Learning for Dynamics & Control Conference*, 2021.

S. M. LaValle. *Planning Algorithms*. Cambridge Univ. Press, 2006.

J. Liang, S. Di Cairano, and R. Quirynen. Early termination of convex QP solvers in mixed-integer programming for real-time decision making. *IEEE Control Syst. Lett.*, 5(4):1417–1422, 2021.

Y. Löhr, M. Klaučo, M. Fikar, and M. Mönnigmann. Machine learning assisted solutions of mixed integer MPC on embedded platforms. In *IFAC World Congress*, 2020.

T. Marcucci and R. Tedrake. Mixed-integer formulations for optimal control of piecewise-affine systems. In *Hybrid Systems: Computation and Control*, 2019.

T. Marcucci, R. Deits, M. Gabiccini, A. Bicchi, and R. Tedrake. Approximate hybrid model predictive control for multi-contact push recovery in complex environments. In *Proc. IEEE Conf. on Robotics and Automation*, 2017.

D. Masti and A. Bemporad. Learning binary warm starts for multiparametric mixed-integer quadratic programming. In *European Control Conference*, 2019.

J. Mern, A. Yildiz, L. Bush, T. Mukerji, and M. Kochenderfer. Improved POMDP tree search planning with prioritized action branching. In *Proc. AAAI Conf. on Artificial Intelligence*, 2021.

I. Mordatch, E. Todorov, and Z. Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics*, 31(4):1–8, 2012.

J. Morton, T. A. Wheeler, and M. J. Kochenderfer. Analysis of recurrent neural networks for probabilistic modeling of driver behavior. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 18(5):1289–1298, 2017.

Mosek APS. The MOSEK optimization software. Available at `http://www.mosek.com`.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *Conf. on Neural Information Processing Systems - Autodiff Workshop*, 2017.

Y. E. Sahin, R. Quirynen, and S. Di Cairano. Autonomous vehicle decision-making and monitoring based on signal temporal logic and mixed-integer programming. In *American Control Conference*, 2020.

M. Srinivasan, A. Chakrabarty, R. Quirynen, N. Yoshikawa, T. Mariyama, and S. Di Cairano. Fast multi-robot motion planning via imitation learning of mixed-integer programs. In *Modeling, Estimation and Control Conference*, 2021.

A. Walsh, S. Di Cairano, and A. Weiss. MPC for coupled station keeping, attitude control, and momentum management of low-thrust geostationary satellites. In *American Control Conference*, 2016.

Z. Wang, T. Taubner, and M. Schwager. Multi-agent sensitivity enhanced iterative best response: A real-time game theoretic planner for drone racing in 3D environments. *Robotics and Autonomous Systems*, 125, 2020.

X. Zhang, M. Bujarbaruah, and F. Borrelli. Safe and near-optimal policy learning for model predictive control using primal-dual neural networks. In *American Control Conference*, 2019.

J.-J. Zhu and G. Martius. Fast non-parametric learning to accelerate mixed-integer programming for hybrid model predictive control. *IFAC-Papers Online*, 53(2):5239 – 5245, 2020.