CoCo: Online Mixed-Integer Control Via Supervised Learning

Abhishek Cauligi, Preston Culbertson, Edward Schmerling, Mac Schwager, *Member, IEEE*, Bartolomeo Stellato, and Marco Pavone, *Member, IEEE*

***Abstract*—Many robotics problems, from robot motion planning to object manipulation, can be modeled as mixed-integer convex program (MICPs). However, state-of-the-art algorithms are still unable to solve MICPs for control problems quickly enough for online use and existing heuristics can typically only find suboptimal solutions that might degrade robot performance. In this work, we turn to data-driven methods and present the Combinatorial Offline, Convex Online (CoCo) algorithm for quickly finding high quality solutions for MICPs. CoCo consists of a two-stage approach. In the offline phase, we train a neural network classifier that maps the problem parameters to a *logical strategy*, which we define as the discrete arguments and relaxed big-M constraints associated with the optimal solution for that problem. Online, the classifier is applied to select a candidate logical strategy given new problem parameters; applying this logical strategy allows us to solve the original MICP as a convex optimization problem. We show through numerical experiments how CoCo finds near optimal solutions to MICPs arising in robot planning and control with 1 to 2 orders of magnitude solution speedup compared to other data-driven approaches and solvers.**

***Index Terms*—Optimization and optimal control, data-driven optimization, mixed-integer convex programming.**

I. Introduction

PLANNING and control using MICP has been an extensive area of study within the robotics community. MICPs can be used as a modeling framework to capture the rich set of behaviors and logical constraints that arise in problems such as planning for systems with contact [1], [2], motion planning [3], [4], and dexterous manipulation [5]. Despite their popularity, MICPs have rarely been put into practice for real-world control tasks with demanding performance requirements of 10-100 Hz operational rates due to computational constraints. Indeed, the tremendous strides made in accelerating MICP solution times by several orders of magnitude in the past few decades rely on multithreaded implementations that are inapplicable on embedded systems commonly found on robot hardware. Thus, although algorithms such as branch-and-bound provide certificates of optimality for MICPs, finding the optimal solution can be challenging in practice due to the $\mathcal{NP}$-hard nature of solving MICPs.

Alternate techniques used to make MICPs amenable for real-time control include terminating branch-and-bound when a feasible solution is first found or by simply rounding fractional integer solutions. However, such methods can degrade robot performance if a poor quality feasible solution is returned. A promising approach that has emerged in recent years is to apply techniques from machine learning to accelerate finding solutions for numerical-optimization based robot controllers [6]. Although melding supervised learning techniques and MICP-control has been considered [7], [8], these approaches have yet to demonstrate the ability to scale to the large number of discrete decision variables or problem parameters typically

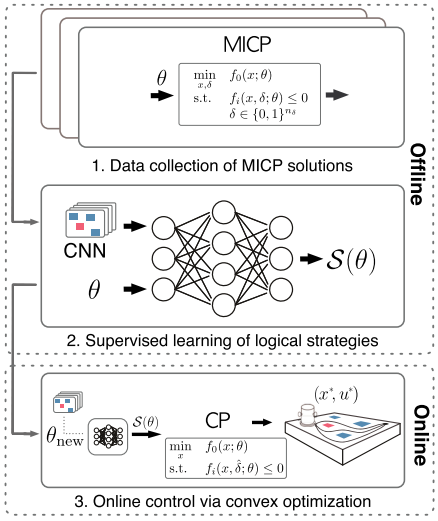


Fig. 1. Our algorithm CoCo is a data-driven approach that seeks to accelerate finding high-quality solutions to parametrized MICPs. The approach proposes a logical strategy $\hat{\mathcal{S}}(\theta)$, which is a candidate discrete solution $\hat{\delta}$ satisfying the logical constraints of the system. Given $\hat{\delta}$, the MICP can be approximately solved as a convex optimization problem. Here, we show CoCo applied to the free-flying spacecraft robot motion planning problem. (1) Offline, CoCo solves a set of MICPs for a representative set of planning problems and constructs the optimal logical strategy $\mathcal{S}^*(\theta)$ using the discrete optimizer δ^* and set of *relaxed* constraints $\mathcal{T}_M(\theta)$. (2) Thereafter, a convolutional neural network classifier is trained to learn a mapping between problem parameters θ and the logical strategy $\mathcal{S}^*(\theta)$. (3) Online, this classifier predicts the strategy $\hat{\mathcal{S}}(\theta)$ associated with new parameters θ and uses the candidate discrete solution $\hat{\delta}$ to solve convex optimization problems until a feasible solution is found.

Manuscript received July 15, 2021; accepted November 27, 2021. Date of publication December 16, 2021; date of current version January 12, 2022. This work was supported in part by the NASA Space Technology Research Fellowship under Grants NNX16AM78H and 80NSSC18K1180, and in part by the NASA University Leadership Initiative under Grant 80NSSC20M0163. This letter was recommended for publication by Associate Editor Daniele Fontanelli and Editor Lucia Pallottino upon evaluation of the reviewers' comments. *(Corresponding author: Abhishek Cauligi.)*

Abhishek Cauligi, Edward Schmerling, Mac Schwager, and Marco Pavone are with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305 USA (e-mail: acauligi@stanford.edu; schmrlng@stanford.edu; schwager@stanford.edu; pavone@stanford.edu).

Preston Culbertson is with the Department of Mechanical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: pculbertson@stanford.edu).

Bartolomeo Stellato is with the Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: bstellato@princeton.edu).

Digital Object Identifier 10.1109/LRA.2021.3135931

found in robotics. Further, these approaches are agnostic to how the discrete decision variables are utilized (e.g., piecewise affine constraints, mixed logical dynamics) and do not take into consideration the logical structure of the problem in their solution approach.

In this work, we seek to develop a learning-based methodology for applying MICP-based control with the following desiderata in mind:

1) *Performant control:* The controller should be able to find high quality solutions with respect to some performance metric for the control task.
2) *Speed:* The online solution procedure should be capable of providing real-time decision making.
3) *Generalizability:* Discrete decision variables are used to encode a rich set of behaviors in robotics and we seek an approach that can leverage the underlying structure present in many robot control tasks.
4) *Scalability:* The approach should be capable of solving MICPs with a high dimensional parameter space and 10s-100 s of discrete decision variables.

Related work: The use of data-driven methods for quickly solving numerical optimization-based controllers is a nascent area of research. In [6], supervised learning is used for learning warm starts for a quadratic program (QP)-based controller. Non-convex optimization-based controllers are considered as well, with [9] learning warm starts for a sequential quadratic program (SQP)-based trajectory optimization library. The authors in [10] leverage differentiable convex optimization to develop a learning-based approach for tuning a QP-controller. However, the shortcoming of these preceding approaches is that they are limited to continuous optimization problems.

In comparison, there is a paucity of approaches that have investigated the use of data-driven techniques for accelerating integer program solutions in control. For general discrete optimization problems, a popular approach has been to consider branch-and-bound as a sequential decision making problem and apply reinforcement learning approaches [11]. The shortcoming of such techniques for control is that they still rely on solving branch-and-bound online and often require multiple neural network forward passes at each node.

More recently, both [5], [7] propose a supervised learning approach using a neural network to warm start the binary variable assignments for an mixed-integer quadratic program (MIQP) controller and a k-nearest neighbors approach is proposed in [8]. In these approaches, the supervised framework maps problem parameters to a candidate discrete solution and, by fixing the discrete decision variables to the candidate solution values, the MIQP is solved as a QP. However, these approaches do not address scalability to a high dimensional parameter space or large number of discrete decision variables. Further, their solution strategy of directly proposing a candidate discrete solution without considering how the variables are employed in the MICP limits their applicability to higher dimensional parametric programs often found in robotics. Our proposed approach accommodates a broad set of systems for which MICPs are used as a modeling tool, such as mixed logical dynamical systems and piecewise affine (PWA) systems.

Our work also draws inspiration from the field of explicit model predictive control (MPC) [12]. Unlike standard multi-parametric optimization approaches that construct a solution map corresponding to polyhedral partitions of the parameter space, our approach learns strategies for regions of a nonlinear transformation (learned via a neural network) of the parameter space of the problem.

Statement of Contributions: Towards filling the gaps in existing works, we present the Combinatorial Offline, Convex Online (CoCo) framework. CoCo is a data-driven framework for solving MICPs and entails a two stage approach (as detailed in Fig. 1) consisting of an offline and online phase. We introduce the concept of logical strategies and show how they enable for CoCo to find high quality solutions for a broad class of problems modeled as MICPs, including problems with 100 s of binary decision variables and a large input parameter dimension.

To the best of our knowledge, CoCo uniquely satisfies the four aforementioned desiderata. This paper extends a conference publication [13] and provides the following additional contributions:

1) Demonstration of how the notion of task-specific logical strategies can be exploited to solve problems with a varying number of discrete variables.
2) Additional numerical results comparing CoCo against a commercial MICP solver, alternative data-driven methods used to find feasible solutions for MICPs, and GuSTO, a state-of-the-art local trajectory optimization method.
3) A thorough analysis of how CoCo can be deployed in practical situations and used across a variety of tasks.

II. Technical Background

This section introduces the parametrized MICPs studied in this work, the big-M constraint formulation approach, and the concept of well-posedness of MICP solutions.

A. Parametrized MICPs

This work considers discrete optimization problems of the specific form known as parametrized mixed-integer convex programs. Given problem parameters θ, we can define the following parametrized MICP with continuous decision variables $x \in \mathbf{R}^{n_x}$, and binary decision variables $\delta \in \{0,1\}^{n_\delta}$:

$$\begin{aligned} \underset{x,\delta}{\text{minimize}} \quad & f_0(x;\theta) \\ \text{subject to} \quad & f_i(x,\delta;\theta) \leq 0, \quad i = 1,\dots,m_f \\ & h_i(\delta;\theta) \leq 0, \quad i = 1,\dots,m_I \\ & \delta \in \{0,1\}^{n_\delta}. \end{aligned} \tag{1}$$

The objective function f_0 and inequality constraints f_i are assumed convex with respect to x. The purely integer constraints h_i are assumed linear with respect to δ.

We note here that the binary decision variables δ are the "complicating" variables in the sense that (1) becomes much easier to solve if δ are temporarily held fixed and the resulting convex program solved in terms of x. Indeed, if an optimal discrete solution δ^* for (1) is provided, then the continuous optimizer x^* for the MICP can be easily found by solving a single convex optimization problem,

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x;\theta) \\ \text{subject to} \quad & f_i(x,\delta^*;\theta) \leq 0, \quad i = 1,\dots,m_f. \end{aligned}$$

Thus, we see that identifying an optimizer δ^* for (1) allows a user to quickly find the optimal solution for an MICP and circumvent, e.g., exploring a full branch-and-bound tree. To this end, a host of supervised learning approaches from [7], [8], [14] utilize this insight to generate a candidate $\hat{\delta}$ given problem parameters θ for an MICP.

B. Big-M Formulation

In MICPs, binary variables δ are often introduced in conjunction with what is known as big-M formulation to capture high-level discrete or logical behavior of the system. These big-M constraints then enforce the desired high-level behavior on the continuous variables x for the robot task at hand, e.g., contact task assignment or hybrid control logic [15].

As an example of a desired logical behavior, consider enforcing the constraint,

$$f_1(x;\theta) \leq 0 \ \vee f_2(x;\theta) \leq 0, \tag{2}$$

where $\vee$ indicates an "or" relationship. The big-M method can be used to encapsulate this constraint by introducing an auxiliary term,

$$M_i(\theta) := \sup_x f_i(x;\theta), \quad i = 1, 2$$

where $M_1(\theta)$ is the least upper bound on the value attainable by $f_1(x;\theta)$ and $M_2(\theta)$ the least upper bound for $f_2(x;\theta)$. Then, the logical behavior can be enforced through the introduction of two binary variables δ_1 and δ_2,

$$f_1(x;\theta) \leq M_1(\theta)(1-\delta_1) \tag{3}$$

$$f_2(x;\theta) \leq M_2(\theta)(1-\delta_2) \tag{4}$$

$$\delta_1 + \delta_2 \geq 1 \tag{5}$$

The equivalence between (2) and (3)–(5) can be verified by noting that when $\delta_1 = 1$, then $f_1(x;\theta) \leq 0$ is automatically recovered. Similarly, $\delta_2 = 1$ leads to the constraint $f_2(x;\theta) \leq 0$. Enforcing the behavioral constraint that one of the two constraints must be satisfied is accomplished through the final constraint (5).

We note that the right hand sides of (3)–(4) are linear in terms of δ_i and thus denote a "big-M" constraint as an affine expression of δ_i:

$$g_i(x;\theta) \leq a_i(\theta)(1-\delta_i), \tag{6}$$

where $a_i(\theta)$ are positive-valued constants precomputed using $M(\theta)$, or any upper bound for the constraint, to impose the desired logical behavior.

In this work, we use big-M constraints exclusively to relate the continuous variables x and binary variables δ. If m_M is the number of big-M constraints used, we can write more specifically the class of MICPs studied as:

$$\begin{aligned}
\underset{x,\delta}{\text{minimize}} \quad & f_0(x;\theta) \\
\text{subject to} \quad & f_i(x;\theta) \leq 0, \quad i = 1,\dots,m_f \\
& g_i(x;\theta) \leq a_i(\theta)(1-\delta_i), \quad i = 1,\dots,m_M \\
& h_i(\delta;\theta) \leq 0, \quad i = 1,\dots,m_I \\
& \delta \in \{0,1\}^{n_\delta}. \qquad\qquad \mathcal{P}(\theta)
\end{aligned}$$

C. Well-Posedness of MICP Solutions

Here, we distinguish between two common classes of MICPs and will later show how this difference is crucial in formulating an effective solution approach. Specifically, as the inclusion of binary variables δ in (1) renders the problem non-convex, an MICP may admit multiple globally optimal solutions (x^*, δ^*). This leads to the distinction between *well-posed* and *completely well-posed* MICPs as introduced in [15]. Well-posed MICPs admit a unique continuous minimizer x^*, but may admit multiple discrete optimizers δ^*. Completely well-posed problems assume that an MICP admits a single global minimizer (x^*, δ^*).

Completely well-posed MICPs can be used to model many systems, such as those with PWA constraints. In the case of PWA constraints, a continuous solution x^* can only be attained by a particular set of mode transitions that are uniquely encoded by a single discrete optimizer δ^*. More broadly however, the assumption of a unique discrete optimizer is a limiting one and cannot accomodate many applications. For example, mixed logical dynamical systems [15] enforce logic rules on discrete-time dynamical systems with both continuous and discrete decision variables, but allow for multiple discrete optimizers $\{\delta^*\}$ that satisfy the propositional logic constraints. Thus, we assume that the problems we treat are well-posed MICPs. Although this entails assuming that the MICP admits a single x^*, we note that this is a mild assumption in practice. For example, if problem parameters θ yield multiple x^* for $\mathcal{P}(\theta)$, a small perturbation in the initial condition typically breaks ties and leads to a single minimizer x^*. Indeed, in many robotics problems, slight regularization terms can be added to the objective function to ensure a unique x^* while still accomplishing the control task [16].

Given a continuous optimizer x^*, the set of binary optimizers $\{\delta^*\}$ for a well-posed MICP can be characterized by identifying which big-M constraints of the form (6) are *relaxed*, where a constraint $g_i(x^*;\theta)$ is denoted relaxed if:

$$g_i(x^*;\theta) > 0. \tag{7}$$

That is, a big-M constraint is considered relaxed if, after plugging in the values from x^*, the constraint (6) is satisfied if and only if the binary variable δ_i^* attains value 0. [1] If δ is subject to purely integer constraints (e.g., a cardinality constraint), then the purely integer constraints are also included in identifying the list of relaxed constraints. In contrast, a big-M constraint is considered *enforced* if the values for x^* automatically lead to constraint satisfaction, i.e., $g_i(x^*;\theta) \leq 0$ which allows for both cases of $\delta_i^* = 0$ or $\delta_i^* = 1$ (subject to purely integer constraints). Thus, the set $\{\delta^*\}$ is then comprised of δ^* that attain value $\delta_i^* = 0$ for the set of relaxed constraints and only differ in the values associated with the enforced constraints.

As an example, we consider a continuous optimizer x^* that satisfies the logical expression from (2) for which $f_1(x^*;\theta) \leq 0$, but $f_2(x^*;\theta) > 0$. In this case, $f_2(x^*;\theta)$ is a relaxed constraint, while $f_1(x^*;\theta)$ is an enforced constraint. This value of x^* then admits two binary optimizers $(\delta_1^*, \delta_2^*) = (0,0)$ and $(\delta_1^*, \delta_2^*) = (1,0)$. We see here that as $f_2(x^*;\theta)$ is the relaxed constraint, δ_2^* attains value 0 for both possible binary optimizers.

III. Technical Approach

This section defines logical strategies, presents the subsequent development of task-specific logical strategies, and demonstrates how they can be used for solving MICPs.

A. Logical Strategies

MICPs are used to encode a rich set of logical constraints and behaviors for robotic systems and existing approaches seek to tackle this broad class of problems by identifying an optimal discrete solution δ^* associated with problem parameters θ. For well-posed MICPs, a solution approach that only proposes a single candidate binary optimizer $\hat{\delta}$ given θ leads to an ill-posed supervised learning problem, as there may exist a set of target values $\{\delta^*\}$ given a θ.

[1] We stress here that a relaxed constraint does not connote a relaxation of the binary variable δ_i over its convex hull, $\delta_i \in [0,1]$.

To address this shortcoming, we present the notion of logical strategies, named as such because they take into consideration how binary variables δ are used to enforce high-level logical behavior in well-posed MICPs. A logical strategy $\mathcal{S}(\theta)$ leverages the idea that, while there may exist multiple discrete optimizers given a continuous optimizer x^*, the problem $\mathcal{P}(\theta)$ can be solved by identifying the set of relaxed big-M constraints.

Given the preceding definition of a relaxed big-M constraint from (7), we now define a logical strategy. Given problem parameters θ and continuous optimizer x^* for the problem $\mathcal{P}(\theta)$, let $\delta^*(\theta)$ be a particular binary optimizer from the set of discrete optimizers $\{\delta^*(\theta)\}$ and $\mathcal{T}_M(\theta)$ be the set of relaxed big-M constraints for the problem,

$$\mathcal{T}_M(\theta) = \{i \mid g_i(x^*; \theta) > 0\}. \tag{8}$$

We then define the logical strategy $\mathcal{S}(\theta)$ as a tuple $(\delta^*(\theta), \mathcal{T}_M(\theta))$.

The utility of the logical strategy definition becomes apparent when revisiting the original MICP in (1). If the optimal logical strategy $\mathcal{S}^*(\theta)$ is provided for an MICP, then the continuous optimizer x^* for the MICP can be found by solving a single convex optimization problem,

$$\begin{aligned} \underset{x}{\text{minimize}} \;& f_0(x, \delta^*; \theta) \\ \text{subject to} \;& f_i(x; \theta) \leq 0, \quad i = 1, \ldots, m_f \\ & g_i(x; \theta) \leq 0, \quad i \in \mathcal{T}_M^c(\theta) \end{aligned}$$

where $\mathcal{T}_M^c(\theta)$ is the set of enforced constraints, i.e., the complement of $\mathcal{T}_M(\theta)$. We note here that if $\mathcal{P}(\theta)$ admits a set of discrete solutions $\{\delta^*\}$ given a particular x^*, then any one of the δ^* can be included in $\mathcal{S}(\theta)$. This leads to the insight that, rather than proposing a candidate $\hat{\delta}(\theta)$, a solution approach should rather propose a candidate logical strategy $\hat{\mathcal{S}}(\theta)$. As we demonstrate later, considering logical strategies improves performance of a supervised learning-based solution approach.

B. Task-Specific Logical Strategies for MICPs

For large problems, proposing a candidate logical strategy starts to become intractable as the number of candidate logical strategies becomes too large. However, underlying problem structure allows us to consider the logical strategy associated with each constraint individually. To that end, we introduce the idea of task-specific logical strategies that consider a common structure arising in many robotics problems known as separability.

To demonstrate an example of separability, we consider the robot motion planning problem shown in Fig. 1. Four binary variables are used to enforce obstacle avoidance, where each binary variable is associated with lying on one side of an axis-aligned rectangular obstacle at a particular time. Thus, the binary variables δ are decoupled on the basis of which obstacle they are associated with.

Task-specific logical strategies seek to exploit this problem structure. Formally, the underlying mixed logical constraints can be written as a conjunction of Boolean formulas:

$$F_1 \wedge F_2 \wedge \cdots \wedge F_\ell,$$

where F_i is a distinct sub-formula of literals involving continuous and binary variables, and each binary variable δ_j is associated with only one sub-formula F_i. When the mixed logical constraint consists of ℓ such Boolean formulas, then the logical strategy $\mathcal{S}(\theta)$ can itself be split into ℓ sub-formula strategies $\mathcal{S}_1(\theta), \ldots, \mathcal{S}_\ell(\theta)$. We note that this decomposition is only possible because of the separability in the problem that allows for each δ_j to be considered only in relation to a single sub-formula $\mathcal{S}_i(\theta)$ and the value of the continuous solution x^*.

This technique for leveraging the separability of the constraint set is applicable to a host of robotics problems, e.g., a walking robot which needs to reason about contact with N surfaces, robots in a multi-agent system that need to reason about interactions with N neighbors, etc. One problem that we focus on here is the MICP formulation of collision avoidance constraints [3]. Consider an axis-aligned, 2D rectangular obstacle m that is parametrized by the coordinates of its lower-left hand corner $(x^m_{\text{min}}, y^m_{\text{min}})$ and upper right-hand corner $(x^m_{\text{max}}, y^m_{\text{max}})$. If the 2D position of the robot is $p = (p^1, p^2) \in \mathbf{R}^2$, then the collision avoidance constraints with respect to obstacle m are:

$$x^m_{\text{max}} - M\delta^{m,1} \leq p^1 \leq x^m_{\text{min}} + M\delta^{m,2} \tag{9}$$

$$y^m_{\text{max}} - M\delta^{m,3} \leq p^2 \leq y^m_{\text{min}} + M\delta^{m,4} \tag{10}$$

where M is chosen to be a sufficiently large number. As written in (9) and (10), $\delta^{m,i} = 1$ indicates that robot is on one side of face i of the obstacle and in violation of that keep-out constraint. To ensure that the robot does not collide with obstacle m, a final constraint

$$\sum_{i=1}^{4} \delta^{m,i} \leq 3, \tag{11}$$

is enforced. Note that each binary variable depends only on the three other variables associated with the same obstacle. Thus, each task-specific logical strategy considers only the binary variables and big-M constraints for obstacle m.

Task-specific logical strategies offer several advantages for a supervised learning approach to solving parametrized MICPs. First, as the number of possible logical strategies can grow exponentially in terms of the number of binary variables n_δ, considering each sub-formula strategy separately $\mathcal{S}_i(\theta)$ reduces the number of binary variables in each task-specific logical strategy, thereby resulting in a smaller number of values that each sub-formula can attain. Second, this reduced number of candidate sub-formulas leads to improved supervision in a learning-based approach as the number of target values is reduced. Finally, in the case when additional Boolean formulas are added (i.e., additional binary variables are added to the MICP), the sub-formula strategies can be queried at inference time for these new formulas.

Thus, task-specific logical strategies can lead to improved performance in problems with separability by reducing the number of class labels and thereby improving supervision for a data-driven approach. However, as we show, the separability of the constraints has performance limits, especially in applications where additional sub-formulas strategies are queried at test time. Practically, in the context of robotics, the performance of the controller can be assessed beforehand and only deployed for tasks that are sufficiently similar to the test set (e.g., a maximum number of obstacles for which task-specific logical strategies are queried).

IV. Combinatorial Offline, Convex Online

We now present our proposed approach CoCo, short for Combinatorial Offline, Convex Online. CoCo consists of a two-stage approach for training and deploying a neural network classifier that maps problem parameters θ to a candidate logical strategy $\hat{\mathcal{S}}(\theta)$.

Algorithm 1 details the offline portion of CoCo. The algorithm takes as input a set of problem parameters $\{\theta_i\}$, where each θ_i is sampled from a parameter distribution $p(\Theta)$ representative

Algorithm 1: CoCo Offline.

Require: Batch of training data $\{\theta_i\}_{i=1,\ldots,T}$, problem $\mathcal{P}(\theta)$

1: Initialize strategy dictionary $\mathcal{S} \leftarrow \{\}$, train set $\mathcal{D} \leftarrow \{\}$
2: $k \leftarrow 0$
3: **for** each θ_i **do**
4: Solve $\mathcal{P}(\theta)$
5: **if** $\mathcal{P}(\theta)$ is optimal **then**
6: Construct optimal strategy $\mathcal{S}^*$
7: **if** $\mathcal{S}^*$ **not in** $\mathcal{S}$ **then**
8: Add $\mathcal{S}^*$ to $\mathcal{S}$
9: Assign class label y_k to $\mathcal{S}^*$
10: $k \leftarrow k + 1$
11: **end if**
12: Identify class label y_i for strategy class $\mathcal{S}^*$
13: Add (θ_i, y_i) to $\mathcal{D}$
14: **end if**
15: **end for**
16: Choose network weights ϕ which minimize cross-entropy loss $\mathcal{L}(h_\phi(\theta_i), y_i)_{i=1,\ldots,T}$ via stochastic gradient descent
17: **return** h_ϕ, $\mathcal{S}$

Algorithm 2: CoCo Online.

Require: Problem parameters θ, strategy dictionary $\mathcal{S}$, trained neural network h_ϕ, n_{evals}

1: Compute class scores $h_\phi(\theta)$
2: Identify top n_{evals}-scoring strategies in $\mathcal{S}$
3: **for** $j = 1, \ldots, n_{\text{evals}}$ **do**
4: **if** $\mathcal{P}(\theta)$ is feasible for strategy $\mathcal{S}^{(j)}$ **then**
5: **return**Feasible solution (x^*, δ^*)
6: **end if**
7: **end for**
8: **return** failure

of the problems encountered in practice. We refer to $\mathcal{S}$ as the strategy dictionary and $\mathcal{D}$ as the train set, both of which are initially empty (Line 1). The strategy dictionary $\mathcal{S}$ stores the set of logical strategies $\{\mathcal{S}^{(i)}\}$ constructed during the offline phase and the train set $\mathcal{D}$ stores the set training tuples $\{(\theta_i, y_i)\}$ used for training the neural network classifier, where y_i is the class label associated with logical strategy $\mathcal{S}^{(i)}$. For each θ_i, the MICP $\mathcal{P}(\theta)$ is solved (Line 4). If an optimal solution is found to the MICP, then the primal solution (x^*, δ^*) is used to construct the logical strategy $\mathcal{S}^*$ for this problem (Line 6) and added to the strategy dictionary $\mathcal{S}$ if it is not already included (Lines 7-11). The class label y_i associated with $\mathcal{S}^*$ is identified (Line 12) and the tuple (θ_i, y_i) added to $\mathcal{D}$ (Line 13). Finally, a neural network classifier $\hat{h}_\phi$ with output dimension $|\mathcal{S}|$ is trained using the elements of $\mathcal{D}$ and the weights ϕ are chosen in order to approximately minimize a cross-entropy loss over the training samples (Line 16).

Remark: We note here that, in addition to learning a mapping between θ and $\mathcal{S}^*(\theta)$, a natural extension would seem to be using a supervised learner to predict the continuous optimizer x^*. However, one of the pitfalls of this approach is that predicting the continuous decision variables poses its own set of unique challenges compared to predicting the $\mathcal{S}^*(\theta)$. Specifically, enforcing constraints on the output of a neural network (e.g., dynamics constraints) is in general challenging and indeed its own active area of research [6]. Thus, the continuous prediction $\hat{x}$ would likely require the use of a solver to project $\hat{x}$ onto the constraint set nonetheless.

V. Numerical Experiments

In this section, we compare CoCo with commercial solvers and other data-driven approaches for solving MICPs. We present results on two benchmark problems in robotics that are modeled as MICPs: the control of an underactuated cart-pole with contact and the robot motion planning problem.

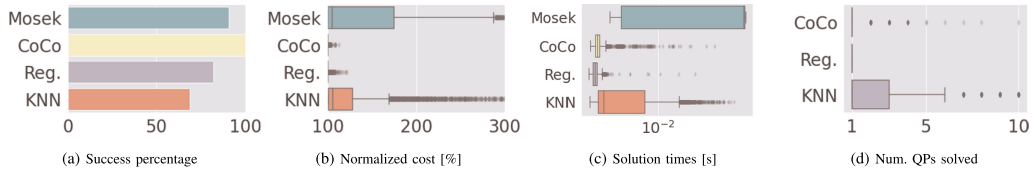
A. Implementation Details

For each system, we first generate a dataset by sampling θ from $p(\Theta)$, until a sufficient number of problems $\mathcal{P}(\theta)$ are solved. For each system, we separate 90% of the problems for training and the remaining 10% for evaluation. The neural network architecture choice was made on the basis of trading off expressivity of the model and the computation time added by a forward pass of the network. Through an architecture search, we empirically found that classifier performance was more sensitive to the network width compared to the depth. For the cart-pole problem, the neural network architecture consists of a standard ReLU feedforward network with three layers and 32 neurons per layer. As the input parameter dimension n_p is larger for the free-flyer system, we chose a neural network with larger width and used a CNN architecture with four convolutional layers followed by a feedforward network with three layers and 128 neurons per layer.

We implemented each example in Python and used the `PyTorch` machine learning library to implement our neural network models with the ADAM optimizer for training. The MICPs were written using the `cvxpy` modeling framework and solved using Mosek. We disable presolve and multithreading to better approximate the computational resources of an embedded processor. The code for our algorithm is available at `https://github.com/StanfordASL/CoCo`.

We further benchmark CoCo against Mosek, the regression framework from [7], and the k-nearest neighbor (KNNs) algorithm from [8]. For the regressor, we train a neural network with identical network architecture as the CoCo classifier, updated with the appropriate number of integer outputs, and use the binary cross-entropy loss after the application of a sigmoid activation. The inference procedure then entails a simple forward pass of the neural network followed by element-wise rounding of the output of the sigmoid activation. We also note that the KNN algorithm with the use of an ℓ_2 distance metric learns a Voronoi partitioning of the solution $\mathcal{S}^*(\theta)$ in the parameter space of the problem. In this sense, the KNN approach is analogous to an explicit MPC approach for QPs wherein a state feedback policy is constructed in the parameter space. Finally, we include Guaranteed Sequential Trajectory Optimization (GuSTO) [17] as an additional benchmark for the free-flyer motion planning problem as a comparison against a state-of-the-art motion planning algorithm that uses local (i.e., sequential convex programming) rather than global (i.e., MICP) optimization for planning in the presence of obstacles.

For each system, we evaluate the benchmarks on a test set of feasible MICPs and report the percent of problems across this test for which feasible solutions were found. For the feasible solutions found by each approach, we also include the cost of the feasible solution compared to the globally optimal solution found offline using branch-and-bound, the computation time to find the feasible solution, and the number of convex relaxations solved before a feasible solution is found.



(a) Success percentage (b) Normalized cost [%] (c) Solution times [s] (d) Num. QPs solved.

Fig. 2. For the cart-pole system, CoCo finds near-global solutions for a majority of problems. (a) Success percentage. (b) Normalized cost [%] (c) Solution times [s] (d) Num. QPs solved.

B. Cart-Pole With Soft Walls

We first study the cart-pole with wall system, a well-known underactuated, multi-contact problem in robot control [1], [2]. The system consists of a cart and pole and the optimal control problem entails regulating the system from initial state x_0 to a goal x_g. The non-convexity of the problem stems from four binary variables $\delta_t \in \mathbf{R}^4$ introduced at each time step to enforce the logical constraint that the contact force from the wall only becomes active when the tip of the pole makes contact with either wall. The parameter space $\theta \in \mathbf{R}^8$ for this problem is comprised of the initial state $x_0 \in \mathbf{R}^4$ and goal state $x_g \in \mathbf{R}^4$. We refer the reader to [13] for a full derivation of system constraints.

1) Results: Numerical results for the cart-pole system are given in Fig. 2. We set the horizon N to the value 10, resulting in a total of 40 binary variables. The training set consists of 90,000 problems generated using parameters sampled from the parameter distribution $p(\Theta)$ and evaluation metrics presented for a test set of 10,000 problems. Fig. 2(a) reports the percent of feasible solutions found over the test set of 10,000 MICPs. For the commercial solver, branch-and-bound is timed out after 50 ms and, for CoCo, ten candidate logical strategies are evaluated before the algorithm terminates with failure. Computation times shown in Fig. 2(c) include the inference step to generate a candidate binary solution (i.e. the forward pass of the network, nearest neighbor lookup, etc.) plus solution time for solving convex relaxations before a feasible solution was found. For the problems from the test set for which a feasible solution was found, Fig. 2(b) and (d) report the cost of the feasible solution relative to the globally optimal solution and the number of convex relaxations solved per problem, respectively. Note that that Fig. 2(d) does not include the number of convex relaxations solved by Mosek as the `cvxpy` interface does not provide this information.

We see that CoCo outperforms the commercial solver Mosek and the two other benchmarks. As shown in Fig. 2(a), CoCo is able to learn the underlying structure of the discrete solution of the problem, finding feasible solutions for 99% of the problems, compared to 82% and 69% for the regressor and KNN, respectively. For the regressor, we find in practice rounding the neural network output to yield a binary solution often leads to physically implausible predictions, e.g., predicting contact with both walls simultaneously, whereas, by construction, the class labels in CoCo satisfy such constraints. Mosek (timed out at 50 ms) finds feasible solutions for 91% of the test set and only 44% of these solutions correspond to the globally optimal solution. As Fig. 2(b) to (d) show, CoCo finds the globally optimal solution after one QP solve for 98% of its feasible solutions. Finally, we emphasize here that Fig. 2(b) to (d) report results for only the feasible solutions found by each approach. Although the computation time for the regressor is comparable to CoCo, this computation time corresponds to a smaller number of overall feasible solutions found. Thus, CoCo is able to provide a feasible solution for all problems in the test set without sacrificing optimality or computation time compared to the benchmarks.

C. Free-Flying Space Robots

A fundamental problem in robotics that is inherently combinatorial is that of motion planning in the presence of obstacles. Here, we study a free-flying spacecraft robot that must navigate around obstacles on a planar workspace with linear dynamics. We show how the use of task-specific logical strategies allows for (1) this problem to become tractable for application of CoCo and (2) the learned strategy classifier $\hat{h}_\phi$ to be used at test time for MICPs with a different number of binary variables n_δ than from the training set.

The system state $x_t \in \mathbf{R}^4$ consists of the position $p_t \in \mathbf{R}^2$ and velocity $v_t \in \mathbf{R}^2$. The planning problem is to regulate the robot from an initial state x_0 to a goal state x_g while satisfying dynamics and actuator constraints. The crucial constraint that renders the problem non-convex is the safety constraint $x_t \in \mathcal{X}_{\text{safe}}$, as $\mathcal{X}_{\text{safe}}$ is typically a highly non-convex region of the workspace and solving the planning problem requires a global combinatorial search. For the free-flying space robot, a popular approach to solving the motion planning problem has been to pose it as an MICP [3]. In this formulation, given N_{obs} obstacles, the workspace is first decomposed into keep-in and keep-out zones and binary variables δ used to enforce collision avoidance with the keep out regions. Due to the ℓ_2-norm constraints imposed on the thruster forces, this problem is a mixed-integer quadratically constrained quadratic program (MIQCQP) with $4N_{\text{obs}}N$ binary variables. The parameters θ for this problem include the initial state x_0, goal state x_g, and position of obstacles $\{(x^m_{\min}, y^m_{\min}, x^m_{\max}, y^m_{\max})\}_{m=1}^{N_{\text{obs}}}$.

1) Use of Task-Specific Logical Strategies: We show here how task-specific strategy decompositions are necessary to efficiently solve the free-flyer motion planning problem with CoCo and how the underlying structure of the problem can be leveraged in order to do so.

From inspection of (9) and (10), we note that a binary variable $\delta_t^{m,i}$ appears in constraints only with the other three binary variables for obstacle m at time t. Thus, our key insight here is to decompose the logical strategy on a per obstacle basis. That is, a logical strategy $\mathcal{S}(\theta; m)$ is associated with the constraint used to enforce collision avoidance with obstacle m and encodes information about the binary variable assignment for that obstacle $\{\delta_t^m\}_{t=1}^N$. Rather than training N_{obs} separate classifiers for each obstacle, we train a single classifier with θ and append an encoding to specify which obstacle logical strategy is being queried.

Specifically, this is accomplished in this work using the architecture shown in Fig. 1. First, a synthetic image of the obstacles is generated using obstacle coordinates $(x^m_{\min}, y^m_{\min})$ and $(x^m_{\max}, y^m_{\max})$. The strategy $\mathcal{S}(\theta; m)$ being queried is indicated by coloring in obstacle m with a different color in the image.

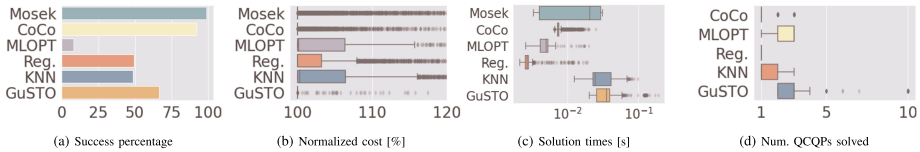


Fig. 3. Simulation results for the free-flyer show how task-specific strategies are necessary for using CoCo. (a) Success percentage. (b) Normalized cost [%] (c) Solution times [s] (d) Num. QCQPs solved.

A convolutional pass is then computed over the synthetic image and the output is flattened and appended with the remaining problem parameters θ before being input to a fully connected feedforward network.

For inference, a batch of input images and problem parameters are constructed, a single forward pass computed, and the strategies for the corresponding sub-formula ranked. One issue with decomposing the strategy queries is that the full binary variable assignment δ must be reconstructed from the individual δ_m. In this work, we consider the n_{evals} highest scoring logical strategy candidates for each $\mathcal{S}(\theta; m)$ and this leads to $n_{\text{evals}}^{N_{\text{obs}}}$ candidates for δ. As this can be a prohibitively large number of solution candidates, we instead randomly sample m_{evals} (where $m_{\text{evals}} << n_{\text{evals}}^{N_{\text{obs}}}$) enumerations from the set of $n_{\text{evals}}^{N_{\text{obs}}}$ candidate assignments for δ. Additionally, we ensure that the δ_m corresponding to the highest scoring candidate $\mathcal{S}(\theta; m)$ for each obstacle is included in this set of m_{evals} candidate binary solutions. We further note that though we focus on axis-aligned obstacles, the use of a CNN architecture would also allow for generalizing to polygonal obstacles that require more complex parameterizations and we leave this to future work.

The results for comparing task-specific strategies used in CoCo are shown in Fig. 3, where we also include Machine Learning Optimizer (MLOPT) [14] and GuSTO [17] as additional benchmarks. We note immediately in Fig. 3(a) that using task-specific logical strategies with CoCo leads to performance gains compared to the approach used for MLOPT, with 92% feasible solutions found for CoCo compared to only 8% for MLOPT. This disparity is attributable to the use of logical strategies $\mathcal{S}^*(\theta)$ as target labels by CoCo rather than the binary optimizer $\delta^*(\theta)$ directly, as CoCo encodes 458 logical strategies versus approximately 67,000 for MLOPT over the 90,000 training problems, leading to a sparser set of training labels per class for MLOPT compared to CoCo. We also see in Fig. 3(c) to (b) that, although Mosek finds a slightly higher number of feasible solutions, CoCo finds on average higher quality solutions compared to Mosek and at twice the speed. Indeed, 90% of the problems for which CoCo finds a feasible solution correspond to the globally optimal solution for that problem. Further, we see in Fig. 3(a) that the regressor and KNN benchmarks fare poorly on the logical constraints and each find feasible solutions for only 49% of the test set. The GuSTO algorithm also fares poorly for this problem, finding feasible solutions for only 67% of problems. This result hints at the inherently combinatorial nature of the motion planning problem, where we see that local trajectory optimization approaches require high quality initializations and suffer from convergence issues while the MICP formulation inherently reasons about multiple homotopy classes.

With regards to computation time, we see in Fig. 3(c) that the regressor on average finds a solution faster than CoCo, but we note once again that these are only for the problems for which it is able to find a feasible solution, which is much fewer than for CoCo. Further, the larger computation times for the classifier based approaches (i.e., CoCo, MLOPT, and the KNN) are attributable to the fact that the regressor can only propose one candidate binary solution at each inference step, whereas the classifier can yield multiple candidate binary solutions that can be assessed until some cutoff time for the problem. Thus, for the motion planning problem, we once again see that CoCo finds a much larger number of feasible solutions compared to the data-driven methods and GuSTO while not sacrificing on optimality or computation time.

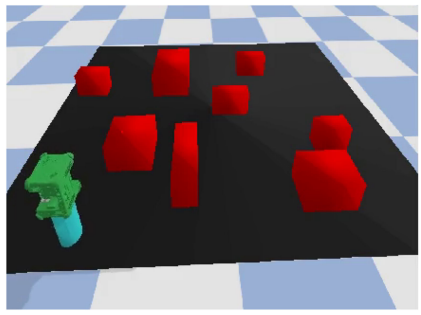


Fig. 4. The simulation environment is modeled after the Stanford Space Robotics Facility testbed and demonstrates a motion planning problem with varying number of obstacles.

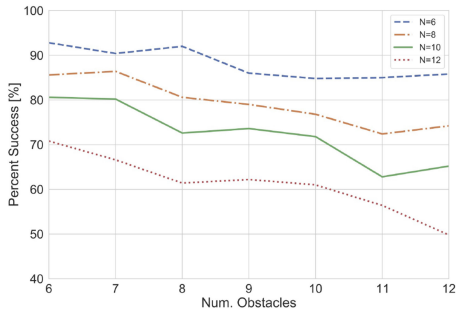


Fig. 5. Generalization results for a CoCo strategy classifier network demonstrates roughly linear dropoff in performance as obstacles are added. The strategy classifier network was trained on an environment with $N_{\text{obs}} = 8$ for various horizons N.

2) Generalization: One important consequence of using a convolutional pass to query the strategy sub-formula $\mathcal{S}(\theta; m)$ is that it can be used for inference in problems with a different number of N_{obs} than from the training set. Here, we evaluate the ability of CoCo to generalize to a distribution of problems with a varying number of binary obstacles. We train multiple strategy classifiers corresponding to horizons of $N = \{5, 7, 9, 11\}$ with environments of eight obstacles. Fig. 5 shows the performance of applying these networks in solving problems with an increasing number of obstacles $N_{\text{obs}} = \{6, \ldots, 12\}$. As shown, we see that the efficacy of the strategy classifier diminishes with an increasing horizon length N due to a corresponding increase in the number of strategies. Intuitively, we also see that performance decreases with an increase in obstacles simply due to

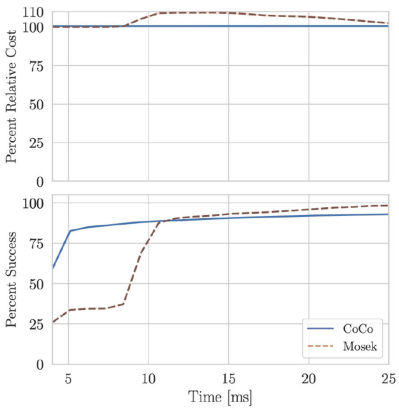


Fig. 6. We compare the performance of CoCo (blue) and Mosek (brown) after timing out both on a test set of MIQCQPs. We see in the bottom plot that timing out Mosek leads to a reduced number of feasible solutions found compared with CoCo. As the cutoff time increases, we see in the top plot that if Mosek finds a feasible solution, then these feasible solutions are suboptimal compared to CoCo, which generally finds the globally optimal solution even when timed out early.

the increased difficulty of the planning problem. However, we note that the performance dropoff remains roughly linear rather than an exponential decrease of performance stemming from including additional binary variables. Thus, given knowledge of the robot operating environment, the use of a trained classifier can be limited to scenarios in which the number of obstacles does not lead to a dramatic dropoff in performance.

3) Timeout Performance: Here, we compare CoCo with a commercial solver that is timed out with a prespecified termination time. For CoCo, we terminate CoCo either when a feasible solution is found or after the termination time has been exceeded. Fig. 6 compares the percent of feasible solutions found between CoCo and Mosek. We see that CoCo finds a feasible solution for the majority of problems within about five milliseconds. However, Mosek requires twice the computation time before it finds a feasible solution for the majority of problems.

Moreover, we see that the solutions found by CoCo are effectively the globally optimal solution for that problem, whereas the Mosek's incumbent solution from branch-and-bound is often suboptimal until branch-and-bound terminates. Thus, in tasks where Mosek is allowed to run its full course, Mosek will indeed find the globally optimal solution, but a designer can weigh the tradeoffs between quickly finding a high quality feasible solution using CoCo or allowing Mosek to terminate. Finally, we further note that in applications requiring a certificate of optimality, a feasible solution found by CoCo can be used as the incumbent and provide a tighter upper bound for branch-and-bound.

4) Simulation Results: We also implemented CoCo in a simulation environment for the Stanford Space Robotics Facility. The Stanford free-flying space robot is a three-DoF system and is equipped with eight thrusters and a reaction wheel to maneuver on a frictionless surface. Our simulation results shown in Fig. 4 demonstrate the efficacy of using CoCo for real-time motion planning in the presence of obstacles and a video of our simulation can be found at https://youtu.be/dXUDekUOGWc.

VI. Conclusion

In this work, we presented CoCo, a data-driven framework to find high quality feasible solutions for MICPs used in robot planning and control problems. We demonstrated how problem structure arising in robot tasks can be utilized effectively in a supervised learning framework. Specifically, we introduced the notion of task-relevant logical strategies to exploit such problem structure and showed how they improve the performance of the trained strategy classifier. We showed through numerical experiments that CoCo improves solution speeds by 1-2 orders of magnitude with only a slight loss of optimality, compared to low-quality feasible solutions found by the commercial solver and alternative data-driven approaches. Finally, we showed how CoCo can uniquely be used to solve problems with a varying number of discrete decision variables and how this allows for solving a new set of tasks online. To this end, we believe that a promising direction of work is to improve the classifier performance given new tasks online. One future approach could be to explore meta-learning to allow for CoCo to adapt network parameters online for improved performance using information gained from solving problems online.

References

[1] R. Deits, T. Koolen, and R. Tedrake, "LVIS: Learning from value function intervals for contact-aware robot controllers," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 7762–7768.
[2] T. Marcucci and R. Tedrake, "Warm start of mixed-integer programs for model predictive control of hybrid systems," *IEEE Trans. Autom. Control*, vol. 66, no. 6, pp. 2433–2448, Jun. 2021.
[3] B. Landry, R. Deits, P. R. Florence, and R. Tedrake, "Aggressive quadrotor flight through cluttered environments using mixed integer programming," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2016, pp. 1469–1475.
[4] P. Culbertson, S. Bandyopadhyay, and M. Schwager, "Multi-robot assembly sequencing via discrete optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 6502–6509.
[5] F. R. Hogan, E. R. Grau, and A. Rodriguez, "Reactive planar manipulation with convex hybrid MPC," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 247–253.
[6] X. Zhang, M. Bujarbaruah, and F. Borrelli, "Safe and near-optimal policy learning for model predictive control using primal-dual neural networks," in *Proc. Amer. Control Conf.*, 2019, pp. 354–359.
[7] D. Masti and A. Bemporad, "Learning binary warm starts for multiparametric mixed-integer quadratic programming," in *Proc. 18th Eur. Control Conf.*, 2019, pp. 1494–1499.
[8] J.-J. Zhu and G. Martius, "Fast non-parametric learning to accelerate mixed-integer programming for hybrid model predictive control," *IFAC-Papers Online*, vol. 53, no. 2, pp. 5239–5245, 2020.
[9] 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," *Robot. Auton. Syst.*, vol. 125, 2020, Art. no. 103410.
[10] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato, "Learning convex optimization control policies," in *Proc. 2nd Conf. Learn. Dyn. Control*, 2019, pp. 361–373.
[11] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6351–6361.
[12] D. Malyuta and B. Açikmeşe, "Approximate multiparametric mixed-integer convex programming," *IEEE Control Syst. Lett.*, vol. 4, no. 1, pp. 157–162, Jan. 2020.
[13] 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. 59th IEEE Conf. Decis. Control*, 2020, pp. 1698–1705.
[14] D. Bertsimas and B. Stellato, "Online mixed-integer optimization in milliseconds," 2019. [Online]. Available: https://arxiv.org/abs/1907.02206
[15] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, pp. 407–427, 1999.
[16] N. M. Boffi and J.-J. E. Slotine, "Implicit regularization and momentum algorithms in nonlinearly parameterized adaptive control and prediction," *Neural Comput.*, vol. 33, no. 3, pp. 590–673, 2021.
[17] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "GuSTO: Guaranteed sequential trajectory optimization via sequential convex programming," in *Proc. Int. Conf. Robot. Autom.*, 2019, pp. 6741–6747.