Bounded-Error Policy Optimization for Mixed Discrete-Continuous MDPs via Constraint Generation in Nonlinear Programming

Michael Gimelfarb, Ayal Taitler, Scott Sanner

University of Toronto, Toronto, ON, Canada
mike.gimelfarb@mail.utoronto.ca, ataitler@bgu.ac.il, ssanner@mie.utoronto.ca

Abstract

We propose Constraint-Generation Policy Optimization (CGPO) for optimizing policy parameters within compact and interpretable policy classes for mixed discrete-continuous Markov Decision Processes (DC-MDPs). CGPO is not only able to provide bounded policy error guarantees over an infinite range of initial states for many DC-MDPs with expressive nonlinear dynamics, but it can also provably derive optimal policies in cases where it terminates with zero error. Furthermore, CGPO can generate worst-case state trajectories to diagnose policy deficiencies and provide counterfactual explanations of optimal actions. To achieve such results, CGPO proposes a bi-level mixed-integer nonlinear optimization framework for optimizing policies within defined expressivity classes (e.g. piecewise linear) and reduces it to an optimal constraint generation methodology that adversarially generates worst-case state trajectories. Furthermore, leveraging modern nonlinear optimizers, CGPO can obtain solutions with bounded optimality gap guarantees. We handle stochastic transitions through chance-constraints, providing high-probability policy performance guarantees. We also present a road-map for understanding the computational complexities associated with different expressivity classes of policy, reward, and transition dynamics. We experimentally demonstrate the applicability of CGPO in diverse domains, including inventory control, management of a system of water reservoirs, and physics control. In summary, CGPO provides structured, compact, and explainable policies with bounded performance guarantees, enabling worst-case scenario generation and counterfactual policy diagnostics.

An important aim of sequential decision optimization of challenging *Discrete-Continuous Markov Decision Processes* (DC-MDPs) in the artificial intelligence, operations research and control domains is to derive policies that achieve optimal control. A desirable property of such policies is *compactness* of representation, which provides efficient execution on resource-constrained systems such as mobile devices [Wang et al. 2022], as well as the potential for *introspection and explanation* [Topin et al. 2021]. Moreover, while the derived policy is expected to perform well in expectation, in many applications it is desirable to obtain *bounds on maximum policy error* and and the scenarios that induce worst-case policy performance [Corso et al. 2021].

Popular policy optimization approaches used in model-free reinforcement learning such as PPO [Schulman et al. 2017] do not provide bounded error guarantees on policy performance, even if they use compact policy classes like decision trees [Topin et al. 2021]. Model-based extensions that leverage gradient-based policy optimization [Bueno et al. 2019] similarly do not provide bounded error guarantees due to the non-convexity of the underlying optimization. In contrast, there exists a rich tradition of work leveraging mixed integer programming (MIPs) for bounded error policy optimization [Dolgov and Durfee 2005, Ahmed et al. 2017, Vos and Verwer 2023], but these methods only apply to discrete state and action MDPs. More importantly, these policy optimization formulations cannot even be expressed as finite MIPs for continuous state or action MDPs due to the infinite number of constraints in these formulations arising from the infinite state and action space. While some historical work has attempted to circumvent large or infinite constraint sets (from the continuous setting) in MDPs via constraint generation [Schuurmans and Patrascu 2001, Farias and Van Roy 2006, Hauskrecht 2006], none of these works optimize policies. Finally, while there exist some bounded error policy solutions for niche MDP classes with continuous states or actions such as linear–quadratic controllers [Åström 2012], ambulance dispatching [Albert 2022], and the celebrated "(s, S)" threshold policies for Scarf et al. [1960]'s inventory management, these solutions are highly customized to the restrictions of each domain and do not readily generalize.

It remains an open question as to how to derive bounded error policies for the infinite state and action, discrete and continuous MDP (DC-MDP) setting. We address this open question through a novel bi-level MIP formulation. Unfortunately, unlike MIP solutions for discrete MDPs, DC-MDPs with continuous states and/or actions cannot be solved directly since the resulting bi-level MIP is (a) bi-level, (b) has an (uncountably) infinite number of constraints, and (c) will often be nonlinear – a MINLP. Fortuitously, we show how a constraint generation form of this policy optimization – termed CGPO – is able to address (a), (b), and (c) to optimize policies with bounded error guarantees in *many cases*.

In summary, we aim to provide a solution approach to optimize and provide strong performance bound guarantees on structured and compact DC-MDP policies under various expressivity classes of policy, reward, and (nonlinear) transition dynamics. Our specific contributions are as follows:

1. *We propose a novel nonlinear bi-level optimization framework called Constraint-Generation Policy Opti-*

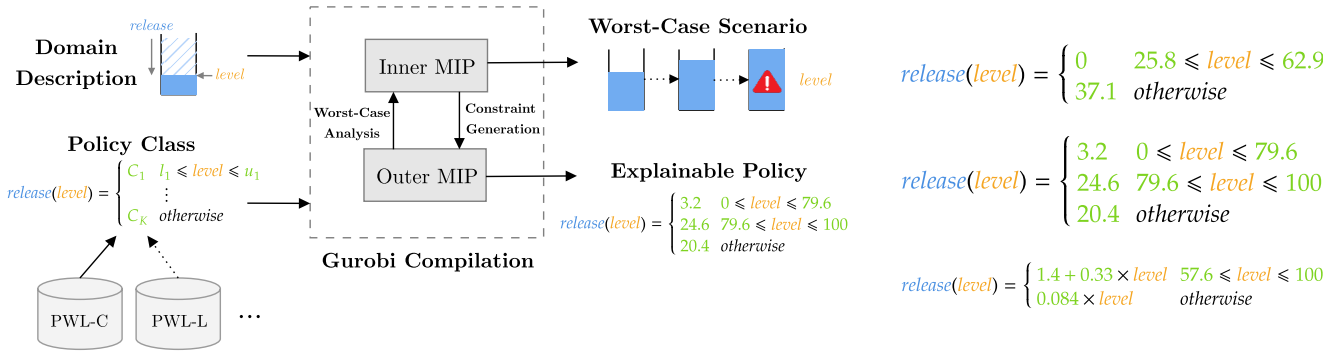


Figure 1: Reservoir control is used as an illustrative example. Left: an overview of CGPO, which consists of a domain description and policy representation compiled to a bilevel mixed-integer program (MIP), in which the inner problem computes the worst-case trajectories for the current policy while the outer problem updates the policy via constraint-generation. The result is a worst-case scenario for the policy (facilitating policy failure analysis), a concrete policy within the expressivity class (for direct policy inspection), and a gap on its performance (error bound). Right: three optimal (i.e. zero-gap) policies produced upon termination across several piecewise policy classes. Crucially, our framework provides the ability to derive highly *compact* (e.g. memory and time-efficient to execute), *intuitive* and *nonlinear* policies, with *strong bound guarantees* on policy performance.

mization (CGPO), that admits a clever reduction to an iterative constraint-generation algorithm that can be readily implemented using standard MIP solvers (cf. Figure 1). State-of-the-art MIP solvers often leverage spatial branch-and-bound techniques, which can provide not only optimal solutions for large mixed integer linear programs (MILP), but also bounded optimality guarantees for mixed integer *nonlinear* programs (MINLP) [Castro 2015]. Chance constraints [Farina, Giulioni, and Scattolini 2016] are used for probabilistic guarantees.

2. *If the constraint generation algorithm terminates, we guarantee that we have found the optimal policy within the given policy expressivity class (Theorem 1).* Further, even when constraint generation does not terminate, our algorithm provides a tight optimality bound on the performance of the computed policy at each iteration and the scenario where the policy performs worst (and as a corollary, for any externally provided policy).
3. *We provide a road map to characterize the optimization problems in (1) above – and their associated expressivity classes – for different expressivity classes of policies and state dynamics, ranging from MILP, to polynomial programming, to nonlinear programs (cf. Table 1).* This information is beneficial for reasoning about which optimization techniques are most effective for different combinations of DC-MDP and policy class expressivity.
4. Finally, *we provide a variety of experiments demonstrating the range of rich applications of CGPO under linear and nonlinear dynamics and various policy classes.* Critically, we derive bounded optimal solutions for a range of problems, including linear inventory management, reservoir control, and a highly nonlinear VTOL control problem. Further, since our policy classes are compact by design, we can also directly inspect and analyze these policies (cf. Figure 2). To facilitate policy interpretation and diagnostics, we can compute the state and exogenous noise trajectory scenario that attains the worst-case error bounds for a policy (cf. Figure 5) as well as a *counterfactual explanation* of what action *should* have been taken in comparison to what the policy prescribes.

Preliminaries

Function Classes

We first define some important classes of functions commonly referred to throughout the paper. Let $\mathcal{X}$ and $\mathcal{Y}$ be arbitrary sets. A function $f : \mathcal{X} \to \mathcal{Y}$ is called *piecewise* (PW) if there exist functions $f_1, \dots f_K, f_{K+1} : \mathcal{X} \to \mathcal{Y}$ and disjoint Boolean predicates $\mathcal{P}_1, \dots \mathcal{P}_K : \mathcal{X} \to \{0, 1\}$, such that for any $x \in \mathcal{X}$, $\forall i, j \neq i$ we have $\mathcal{P}_i(x) = 1 \Rightarrow \mathcal{P}_j(x) = 0$, and $f(x) = f_i(x)$ if $\mathcal{P}_i(x) = 1$ and $f(x) = f_{K+1}(x)$ if $\mathcal{P}_i(x) = 0$, $\forall i$.

Both predicates $\mathcal{P}_i$ and cases f_i can be either linear or nonlinear. In this paper, we consider the following classes:

C *constant*, i.e. $f_i(\mathbf{x}) = C$

D *discrete*, same as restricted values of C

S *simple (axis-aligned) linear functions*, i.e. $f_i(\mathbf{x}) = b_i + w_i \times x_j$, where $j \in \{1, 2 \dots n\}$ and $b_i, w_i \in \mathbb{R}$

L *linear*, i.e. $f_i(\mathbf{x}) = b_i + \mathbf{w}_i^T \mathbf{x}$

B *bilinear*, i.e. $f_i(\mathbf{x}) = x_1 \times x_2 + x_2 \times x_3$

Q *quadratic*, i.e. $f_i(\mathbf{x}) = b_i + \mathbf{w}_i^T \mathbf{x} + \mathbf{x}^T M_i \mathbf{x}$, where $M_i \in \mathbb{R}^{n \times n}$

P *polynomial* of order m, i.e. $f_i(\mathbf{x}) = \sum_{j_1=1}^{m} \cdots \sum_{j_n=1}^{m} w_{i,j_1,\dots j_n} \prod_{k=1}^{n} x_k^{j_k}$

N *general nonlinear*, i.e. $f(\mathbf{x}) = e^{b_i + \mathbf{w}_i^T \mathbf{x}}$.

In a similar way, we consider analogous functions over integer (I) domains $\mathcal{X}$ or $\mathcal{Y}$, or mixed discrete-continuous (M) domains. $\mathcal{P}_i$ can be characterized in a similar way based on whether the constraint, or set of constraints, defining it are constant, linear, bilinear, quadratic, etc.

We also introduce a convenient notation to describe general PW functions by concatenating the expressivity classes of their constraints $\mathcal{P}_i$ and case functions f_i. For example, PWS-C describes a piecewise function on $\mathbb{R}^n$ with simple (axis-aligned) linear constraints and constant values, while PWS-L describes a function with similar constraints but linear values. We can also append the number of cases K to the notation, i.e. PWS1-L describes a piecewise linear function with $K = 1$ predicates. More generally, for PWL, PWP or PWN, constraints could also be (logical) conjunctions of other simpler constraints.

Mathematical Programming

Given a function $f : \mathcal{X} \to \mathcal{Y}$ and Boolean predicate $\mathcal{P} : \mathcal{X} \to \{0, 1\}$, the *mathematical program* is defined as:

$$\min_{x \in \mathcal{X}} \quad f(x) \quad \text{s.t.} \qquad \mathcal{P}(x) = 1.$$

Important standard expressivity classes of *mixed-integer program* (MIP) optimization problems, i.e. those with discrete and continuous decision variables, include:

MILP *mixed-integer linear*: f, $\mathcal{P}$ both in L

MIBCP *mixed-integer bilinearly constrained*: f is in L and $\mathcal{P}$ is in B

MIQP *mixed-integer quadratic*: f is in Q and $\mathcal{P}$ is in L

QCQP *quadratically constrained quadratic*: f,$\mathcal{P}$ both in Q

PP *polynomial*: f, $\mathcal{P}$ both in P

MINLP *mixed-integer nonlinear*: f,$\mathcal{P}$ both in N.

Branch-and-bound [Morrison et al. 2016] solvers are commonly used to maintain upper and lower bounds on the minimal objective value of any linear or nonlinear MIP, whose difference is called the *optimality gap*; when this gap is zero, an optimal solution has been found. Moreover, some packages such as Gurobi, which we use to perform the necessary compilations in CGPO in our experiments, also support a variety of nonlinear mathematical operations via piecewise-linear approximation [Castro 2015].

Discrete-Continuous Markov Decision Processes

A *Discrete-Continuous Markov decision process* (DC-MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, P, r \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions or controls. $\mathcal{S}$ and $\mathcal{A}$ may be discrete, continuous, or mixed. $P(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ is the probability of transitioning to state $\mathbf{s}'$ immediately upon choosing action $\mathbf{a}$ in state $\mathbf{s}$, and $r(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ is the corresponding reward received. We assume that r is uniformly bounded, i.e. there exists $B < \infty$ such that $|r(\mathbf{s}, \mathbf{a}, \mathbf{s}')| \leq B$ holds for all $\mathbf{s}, \mathbf{a}, \mathbf{s}'$.

Given a planning horizon of length T (assumed fixed in our setting), the *value* of a (open-loop) *plan* $\alpha = [\mathbf{a}_1, \dots \mathbf{a}_T] \in \mathcal{A}^T$ starting in state $\mathbf{s}_1$ is

$$V(\alpha, \mathbf{s}_1) = \mathbb{E}_{\mathbf{s}_{t+1} \sim P(\mathbf{s}_t, \mathbf{a}_t, \cdot)} \left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \right].$$

A *policy* $\pi = [\pi_1, \dots \pi_T]$ is a sequence of mappings $\pi_t : \mathcal{S} \to \mathcal{A}$, whose value is defined as

$$V(\pi, \mathbf{s}_1) = \mathbb{E}_{\mathbf{s}_{t+1} \sim P(\mathbf{s}_t, \pi_t(\mathbf{s}_t), \cdot)} \left[\sum_{t=1}^{T} r(\mathbf{s}_t, \pi_t(\mathbf{s}_t), \mathbf{s}_{t+1}) \right].$$

Dynamic programming approaches such as value or policy iteration can compute an optimal horizon-dependent policy π^*_H [Puterman 2014], but do not directly apply to DC-MDPs with infinite or continuous state or action spaces.

Our goal is to compute an optimal stationary *reactive policy* π^* [Bueno et al. 2019] that minimizes the error in value (i.e. regret) relative to π^*_H over all initial states of interest $\mathcal{S}_1 \subseteq \mathcal{S}$, and all stationary policies Π, i.e.

$$\pi^* \in \arg\min_{\pi \in \Pi} \max_{\mathbf{s}_1 \in \mathcal{S}_1} \left[V(\pi^*_H, \mathbf{s}_1) - V(\pi, \mathbf{s}_1) \right]. \qquad (1)$$

In practical applications, the policy class is often restricted to function approximations $\tilde{\Pi} \subset \Pi$. A variety of planning approaches can compute the optimal policies $\tilde{\pi}^* \in \tilde{\Pi}$ for this problem, including straight-line planning that scales well in practice but does not learn policies [Wu, Say, and Sanner 2017, Raghavan et al. 2017], and deep reactive policy [Bueno et al. 2019, Low, Kumar, and Sanner 2022], MCTS [Świechowski et al. 2023] and similar approaches that learn neural network policies but cannot provide concrete bounds on performance, nor compactness or ease of interpretation of the inferred policy map.

Methodology

We begin with a one-dimensional "navigation" problem to serve as an intuitive illustration of CGPO. Next, we present the derivation of CGPO for general DC-MDPs, first for the deterministic case, then the general stochastic case. We end with an analysis of problem complexity and convergence.

Worked Example

Domain Description Let the state s_t denote the continuous position of a particle on $\mathbb{R}$ at epoch $t = 1, 2$, and let the action be a continuous unbounded displacement a_t applied to the particle, so that the state updates according to $s_{t+1} = s_t + a_t$. Thus, $\mathcal{S} = \mathcal{A} = \mathbb{R}$ and suppose further that $\mathcal{S}_1 = [0, 5]$. If the target position is designated as 10, we formulate the reward as $r(s_{t+1}) = -|s_{t+1} - 10|$, and thus the value function is $V(a_1, s_1) = -|s_1 + a_1 - 10|$.

The Outer Problem For ease of illustration, we focus our attention on *linear* policies of the form $a_t = b + ws_t$, where (w, b) are real-valued decision variables to be optimized. The goal is to find the optimal policy parameters:

$$\min_{(w,b) \in \mathbb{R}^2} \max_{s_1 \in [0,5]} \max_{a_1 \in \mathbb{R}} [V(a_1, s_1) - V((w, b), s_1)],$$

where $V((w, b), s_1) = -|s_1 + b + ws_1 - 10|$ is the value of the linear policy. However, this problem is not directly solvable due to the inner $\max$, but it can be written as

$$\min_{(w,b) \in \mathbb{R}^2,\, \varepsilon \in [0,\infty)} \varepsilon, \quad \text{s.t.} \quad \varepsilon \geq V(a_1, s_1) - V((w, b), s_1),$$

where constraints are enumerated by all possible (s_1, a_1) pairs. Since the number of constraints is infinite for continuous states or actions, it suggests a constraint generation approach, where a large set of diverse (s_1, a_1) pairs are used to "build up" the constraint set one at a time, decoupling the overall problem into a sequence of solvable MIPs.

However, a key question arises: ***which* state-action pairs (s_1, a_1) should be considered for inclusion into the outer problem?** While a number of different reasonable choices (diversity, coverage) exist, a logical choice is to select the state action pairs on which the policy $\pi_{(w,b)}$ parameterized by (w, b) performs *worst*.

Solve the Outer Problem The outer problem thus maintains a finite subset of constraints from the infinitely-constrained problem. We begin with an arbitrary (s, a)-pair, which forms the first constraint of the outer problem. For illustration, we select $(s_1, a_1) = (0, 0)$, thus with the constraint $\varepsilon \geq V(0,0) - V((w,b),0) = -10 + |b - 10|$, the outer problem becomes:

$$\min_{w,b\in\mathbb{R},\, \varepsilon\in[0,\infty)} \varepsilon, \quad \text{s.t.} \quad \varepsilon \geq -10 + |b - 10|.$$

Recognizing that the minimum of the r.h.s. of the constraint is attained at $b^* = 10$, and w^* arbitrary, we select $(w^*, b^*) = (0, 10)$ as the solution of the outer problem with optimal $\varepsilon = 0$. Next, we will try to improve upon the policy by adding an additional constraint to the outer problem.

Solve the Inner Sub-Problem To accomplish this, we need to add a new scenario (s_1, a_1) in relation to which the policy $(w, b) = (0, 10)$ performs worst. Thus, we solve:

$$\max_{s_1\in[0,5], a_1\in\mathbb{R}} [V(a_1, s_1) - V((w,b), s_1)]$$
$$= \max_{s_1\in[0,5], a_1\in\mathbb{R}} [-|s_1 + a_1 - 10| + s_1],$$

from which we find $(s_1^*, a_1^*) = (5, 5)$. The optimal value of this problem is positive, so it represents a potentially binding constraint in the outer problem. Thus, we could improve upon the policy by adding the corresponding constraint.

Solve the Outer Problem Again Adding the constraint to the outer problem:

$$\min_{(w,b)\in\mathbb{R}^2,\, \varepsilon\in[0,\infty)} \varepsilon$$
$$\text{s.t.} \quad \varepsilon \geq -10 + |b - 10|, \quad \varepsilon \geq |b + 5w - 5|,$$

whose new solution can be found as $b^* = 10, w^* = -1$ with optimal $\varepsilon = 0$. Finally, the worst-case scenario for this new policy can be found from the inner sub-problem; one solution is $(s_1^*, a_1^*) = (0, 10)$ with value zero. This corresponds to a non-binding constraint, and thus we terminate with the optimal linear policy $\pi^*(s_1) = 10 - s_1$. □

Constraint Generation for Deterministic DC-MDPs

We now extend the intuition above to the general deterministic DC-MDP setting. We assume π can be compactly identified by a vector $\mathbf{w} \in \mathcal{W}$ of decision variables, and we use the shorthand $V(\mathbf{w}, \mathbf{s}_1)$ to denote the value $V(\pi_\mathbf{w}, \mathbf{s}_1)$ of the policy $\pi_\mathbf{w}$ parameterized by $\mathbf{w}$. In this paper, we turn our attention to approximate policy sets $\tilde{\Pi}_\mathcal{W} = \{\pi_\mathbf{w} : \mathbf{w} \in \mathcal{W}\}$ compactly described by piecewise functions.

First, observe that for every possible initial state $\mathbf{s}_1$, there exists a fixed optimal plan $\alpha^* = [\mathbf{a}_1^*, \dots \mathbf{a}_T^*]$ such that $V(\alpha^*, \mathbf{s}_1) = V(\pi_H^*, \mathbf{s}_1)$. On the other hand, since we only have access to an expressivity class of *approximate* policies $\tilde{\Pi}_\mathcal{W}$, the error $\varepsilon(\mathbf{w}, \mathbf{s}_1)$ of $\pi_\mathbf{w}$ in state $\mathbf{s}_1$ relative to $V(\pi_H^*, \mathbf{s}_1)$ (according to (1)) must be

$$\varepsilon(\mathbf{w}, \mathbf{s}_1) \geq \max_{\alpha\in\mathcal{A}^T} V(\alpha, \mathbf{s}_1) - V(\mathbf{w}, \mathbf{s}_1), \tag{2}$$

and thus the (global) *worst-case* error $\varepsilon(\mathbf{w})$ of $\mathbf{w}$ is

$$\varepsilon(\mathbf{w}) \geq \max_{\mathbf{s}_1\in\mathcal{S}_1} \max_{\alpha\in\mathcal{A}^T} [V(\alpha, \mathbf{s}_1) - V(\mathbf{w}, \mathbf{s}_1)]. \tag{3}$$

However, since we seek the approximate *optimal* policy $\tilde{\pi}^* \in \tilde{\Pi}_\mathcal{W}$, we can directly minimize (3) over $\mathcal{W}$, obtaining the following *infinitely-constrained mixed-integer program*:

$$\min_{\mathbf{w}\in\mathcal{W},\, \varepsilon\in[0,\infty)} \varepsilon \tag{4}$$
$$\text{s.t.} \quad \varepsilon \geq \max_{\mathbf{s}_1\in\mathcal{S}_1} \max_{\alpha\in\mathcal{A}^T} \big[V(\alpha, \mathbf{s}_1) - V(\mathbf{w}, \mathbf{s}_1)\big].$$

However, the constraint is highly nonlinear, turning (4) into a bi-level program and making analysis of this particular formulation difficult. Instead, since the $\max$'s must hold for all states and actions, we can rewrite the problem (4) as:

$$\min_{\mathbf{w}\in\mathcal{W},\, \varepsilon\in[0,\infty)} \varepsilon \tag{5}$$
$$\text{s.t.} \quad \varepsilon \geq V(\alpha, \mathbf{s}_1) - V(\pi, \mathbf{s}_1), \quad \forall \alpha \in \mathcal{A}^T,\, \mathbf{s}_1 \in \mathcal{S}_1.$$

The goal is therefore to solve (5), which is still infinitely-constrained when $\mathcal{S}$ or $\mathcal{A}$ are infinite spaces. Instead, we solve it by splitting the $\min$ over $\mathcal{W}$ and the $\max$ over $(\alpha, \mathbf{s}_1)$ into two problems, and apply *constraint generation* [Blankenship and Falk 1976, Chembu et al. 2023].

Specifically, starting with a fixed arbitrary scenario $(\mathbf{s}_1, \alpha)$, we form the constraint set $\mathcal{C} = \{(\mathbf{s}_1, \alpha)\}$, and then solve the following two problems:

Outer Solve (5) within the set of finite constraints $\mathcal{C}$ to obtain policy $\mathbf{w}^* \in \mathcal{W}$.

Inner Solve (3) $\operatorname{argmax}_{\mathbf{s}_1\in\mathcal{S}_1, \alpha\in\mathcal{A}^T} [V(\alpha, \mathbf{s}_1) - V(\mathbf{w}^*, \mathbf{s}_1)]$, for the *highest-error scenario* for $\mathbf{w}^*$, and append it to $\mathcal{C}$.

These two steps are repeated until the constraint added to the outer problem is no longer binding, i.e. $V(\alpha^*, \mathbf{s}_1^*) - V(\mathbf{w}^*, \mathbf{s}_1^*) \leq 0$, since the solution of the outer problem will not change with the addition of the new constraint. We name our approach Constraint-Generation Policy Optimization (CGPO) (see Figure 1, and pseudocode in the Appendix). *In retrospect, we can view CGPO as a policy iteration algorithm where the inner problem adversarially critiques the policy with a worst-case trajectory and the outer problem improves the policy w.r.t. all critiques.*

Remarks Upon termination, CGPO's use of constraint generation guarantees an optimal (i.e., lowest error) policy within the specified policy class (see Theorem 1). While we cannot provide a general finite-time guarantee of termination with an optimal policy, CGPO is an *anytime* algorithm that provides a best policy and a bound on performance at each iteration. At each stage, the solution to the inner problem allows us to analyze the worst-case trajectory with respect to π (cf. Figure 5), and generate a counterfactual explanation of what actions *should* have been made.

Extension to Stochastic MDPs

Chance-constrained approaches [Ono et al. 2015, Farina, Giulioni, and Scattolini 2016, Ariu et al. 2017] allow us to derive high probability intervals on stochastic MDP transitions and reduce our solution to a robust optimization problem with probabilistic performance guarantees. To achieve this, we will require that the state transition function $P(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ has a natural reparameterization as a deterministic function g of $(\mathbf{s}, \mathbf{a})$ and some exogenous i.i.d. noise variable ξ with density $q(\cdot)$ on support Ξ, e.g. $\mathbf{s}' = g(\mathbf{s}, \mathbf{a}, \xi)$ [Bueno et al. 2019, Patton et al. 2022]. Given a threshold $p \in (0, 1)$ close to 1, we also assume the existence of a computable interval $\Xi_p = [\xi_l, \xi_u]$ such that $\mathbb{P}(\xi_l \leq \xi \leq \xi_u) \geq p$.

Thus, we can repeat the derivation of (4) by considering the worst case not only over $\mathbf{s}_1$, but also over possible noise variables $\xi_{1:T} = [\xi_1, \dots \xi_T] \in \Xi_p^T$. The final problem is:

$$\min_{\mathbf{w} \in \mathcal{W},\, \varepsilon \in [0,\infty)} \quad \varepsilon \tag{6}$$
$$\text{s.t. } \varepsilon \geq \max_{\mathbf{s}_1 \in \mathcal{S}_1} \max_{\xi_{1:T} \in \Xi_p^T} \max_{\alpha \in \mathcal{A}^T} \left[V(\alpha, \mathbf{s}_1, \xi_{1:T}) - V(\mathbf{w}, \mathbf{s}_1, \xi_{1:T}) \right],$$

where $V(\cdot, \mathbf{s}_1, \xi_{1:T})$ corresponds to the total reward of the policy or plan accumulated over trajectory $\xi_{1:T}$ starting in $\mathbf{s}_1$. In the stochastic setting, ε only holds with probability p^T for a horizon T problem. Experimentally, we found that choosing p such that $p^T = \alpha$, where α is a desired probability bound on the full planning trajectory, was sufficient. Once again, (6) can be reformulated as:

$$\begin{aligned} \min_{\mathbf{w} \in \mathcal{W},\, \varepsilon \in [0,\infty)} \quad & \varepsilon \\ \text{s.t.} \quad & \varepsilon \geq V(\alpha, \mathbf{s}_1, \xi_{1:T}) - V(\mathbf{w}, \mathbf{s}_1, \xi_{1:T}), \\ & \forall \alpha \in \mathcal{A}^T,\, \mathbf{s}_1 \in \mathcal{S}_1,\, \xi_{1:T} \in \Xi_p^T. \end{aligned} \tag{7}$$

Therefore, we can again apply constraint generation to solve this problem in two stages, in which the inner optimization produces not only a worst-case initial state and action sequence, but also the disturbances $\xi_{1:T}^*$ that reproduce *all* future worst-case state realizations $\mathbf{s}_2, \dots \mathbf{s}_T$. A complete description of this algorithm is provided in the Appendix.

Convergence

Under mild conditions, if CGPO terminates at some iteration t, then $\mathbf{w}_t^*$ is optimal in $\mathcal{W}$ (with probability at least p^T in the stochastic case). The proof is adapted from Blankenship and Falk [1976] and deferred to the Appendix.

Theorem 1. *If $\mathcal{S}_1$, $\mathcal{A}$, Ξ_p and $\mathcal{W}$ are non-empty compact subsets of Euclidean space, $V(\alpha, \mathbf{s}, \xi_{1:T})$ and $V(\mathbf{w}, \mathbf{s}, \xi_{1:T})$ are continuous, and CGPO terminates at iteration t, then $\mathbf{w}_t^*$ is optimal for problem (7).*

Remarks Termination of CGPO guarantees that $\mathbf{w}_t^*$ is optimal in $\tilde{\Pi}_{\mathcal{W}}$, and ε_t^* is the corresponding optimality gap. Moreover, if $\varepsilon_t^* = 0$ and the MDP is deterministic, then $\pi_{\mathbf{w}_t^*}$ is the *optimal* policy in Π. This means we can use the gap estimate as a principled way to compare and validate different policy classes in practice.

Policy	L	P	N
PWS-{C, D}	MILP	PP	MINLP
PWL-{C, D}	MILP/MIBCP	PP	MINLP
S, L	MILP/PP	PP	MINLP
PW{S,L}-{S,L}	MILP/PP	PP	MINLP
PW{S,L,P}-P	PP	PP	MINLP
Q	PP	PP	MINLP
PWN-N	MINLP	MINLP	MINLP

Table 1: Problem classes of the inner/outer optimization problems in CGPO for different expressivity classes of dynamics/reward (columns) and policies (rows).

Problem Expressivity Class Analysis

In Table 1, we present the relationship between the expressivity classes of policies and state transition dynamics and the corresponding classes of the inner and outer optimization problems. The policy classes of interest include PWS-C, PWL-C, PWS-L, PWL-L, PWN-N and the different variants of piecewise polynomial policies. Meanwhile, expressivity classes for state dynamics and reward include linear, polynomial and general nonlinear functions (their piecewise counterparts generally fall under the same expressivity classes, and are excluded for brevity). Interestingly, as shown in the Appendix, when the policy and state dynamics are both linear, the outer problem is a PP. In a similar vein, a PWL-C policy and linear dynamics result in a MIBCP outer problem due to the bilinear interaction between successor state decision variables and policy weights in the linear conditions.

Our experiments in the next section empirically evaluate PWS-C and PWS-L policies with linear dynamics, and Q policies with nonlinear dynamics. This requires solving mixed-integer problems with large numbers of decision variables ranging from MILP to PP to MINLP.

Empirical Evaluation

We empirically validate the performance of CGPO on several MDPs, aiming to answer the following questions:

- **Q1** Does CGPO recover exact solutions when the ideal policy class is known? How does it perform if the optimal policy class is not known?
- **Q2** How do different policy expressivity classes perform for each problem?
- **Q3** Does the worst-case analysis provide further insight about a policy?

Domain Descriptions

To answer these questions, we evaluate on linear inventory control, linear reservoir management, and nonlinear VTOL control domains as summarized below (details are provided in the Appendix). Inventory control has provably optimal PWS-S policies, whereas no optimal policy class is known explicitly for reservoir control; this allows us to answer Q1. A public github repository[1] allows reproduction of all results and application of CGPO to arbitrary RDDL domains.

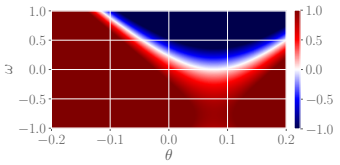
[1] https://github.com/pyrddlgym-project/pyRDDLGym/tree/GurobiCompilerBranch/

$$reorder = 4 - stock$$

$$reorder = \begin{cases} 5 - stock & \text{if } -1 \leq stock \leq 2 \\ 3 - stock & \text{otherwise} \end{cases}$$

$$release_1 = \begin{cases} 33.23 & \text{if } 65.06 \leq level_1 \leq 100 \\ 1.08 & \text{otherwise} \end{cases}$$

$$release_2 = \begin{cases} 72.91 & \text{if } 100 \leq level_2 \leq 200 \\ 0.2 & \text{otherwise} \end{cases}$$



$$F = 0.6 - 15.4\theta - 2.3\omega + 100\theta^2 - 1.5\omega^2$$

Figure 2: Optimal S/PWS-S policies for inventory (left), PWS-C policy for reservoir (middle), Q policy for VTOL (right).

Linear Inventory State s_t describes the *discrete* level of stock available for a single good, action a_t is the *discrete* reorder quantity, demand ξ_t is stochastic and distributed as discrete uniform. Backlogging of inventories is allowed and is represented by negative s_t. The reward function consists of *continuous* unit costs for product purchasing, excess and shortage of inventory. A planning horizon of $T = 8$ is used. For this domain and reservoir below, we focus on learning factorized piecewise policies, i.e. C, S, PWS-C and PWS-S.

Linear Reservoir The goal is to manage the water level in a system of interconnected reservoirs. State $s_{t,r}$ represents the *continuous* water level of reservoir r with capacity M_r, action $a_{t,r}$ is the *continuous* amount of water released, and rainfall $\xi_{t,r}$ is a clipped normally-distributed random variable (to allow possibility of no rainfall). Each reservoir r is connected to a set $U(r)$ of upstream reservoirs. Reward linearly penalizes any excess water level above H_r and below L_r. We use $T = 10$.

Nonlinear VTOL Control The goal is to balance two masses on opposing ends of a long pole. The state consists of the angle θ_t and angular velocity ω_t of the pole, and the action is the force $F_t \in [0, 1]$ applied to the mass. Time is discretized into intervals of length $\tau = 0.1$ seconds, and we set $T = 6$. The reward penalizes the difference between the pole angle and a target angle, as well as the magnitude of the force to ensure smoothness. We optimize for *nonlinear* quadratic policies.

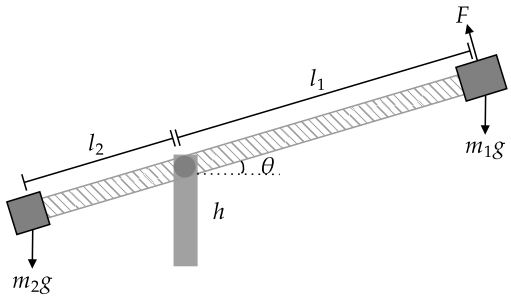


Figure 3: The nonlinear VTOL control problem.

Nonlinear Intercept To demonstrate an example of *discrete* (Boolean) action policy optimization over nonlinear *continuous* state dynamics with independently moving but interacting projectiles, we turn to an intercept problem inspired by Scala et al. (2016). Space restrictions require us to relegate details and results to the Appendix, but we remark here that CGPO is able to derive an optimal policy.

Empirical Results

MIPs are solved to an optimality gap of 5% and we use $p = 0.995$. Fig. 2 illustrates examples of policies learned in a typical run of CGPO. Fig. 4 compares the empirical (simulated) returns and error bounds ε^* (optimal objective value of the inner problem) across different policy classes.

As hypothesized, CGPO can recover exact (s, S) control policies for inventory control in the PWS-S class, which together with S policies achieve the lowest error and best return across all policy classes. Interestingly, as Fig. 4 shows, C and PWS-C policies perform relatively poorly even for $K = 2$, but as expected, the error decreases with the policy expressivity class and the number of cases. This is intuitive, as Fig. 2 shows, the reorder quantity is strongly linearly dependent on the current stock, which is not easily represented as PWS-C. In constrast, for reservoir control the class of PWS-C policies perform comparatively well for $K \geq 1$, and achieve (near)-optimality despite requiring a larger number of iterations. As expected, policies typically release more water as the level approaches the upper target bound. Finally, the optimal policy for VTOL applies large upward force when the angle or angular velocity are negative, with relative importance being placed on angle. As Fig. 2 (right) shows, the force is generally greater when either the angle is below the target angle or the angular velocity is negative, and equilibrium is achieved by applying a modest upward force between the initial angle (0.1) and the target angle (0).

	Inventory		Reservoir	
Policy	Inner	Outer	Inner	Outer
C	8 / 121 / 72	96 / 673 / 385	20 / 0 / 822	300 / 0 / 5823
PWS-S	32 / 129 / 72	384 / 772 / 387	80 / 0 / 842	1200 / 0 / 7133
C	160 / 0 / 64	780 / 0 / 480	540 / 0 / 320	4035 / 0 / 2400
PWS-S	168 / 0 / 112	709 / 168 / 1056	560 / 0 / 440	3797 / 540 / 4200

Table 2: Top half shows the number of binary/integer/real-valued decision variables in the inner and outer optimization problems of CGPO for different benchmark problems and policy classes at the last iteration. Bottom half shows the number of linear/quadratic/general constraints.

Worst-Case Analysis Fig. 5 plots the state trajectory, actions and noise (i.e. rainfall) that lead to worst-case performance of C and S policies after the last iteration of CGPO for reservoir. Here, we observe that the worst-case

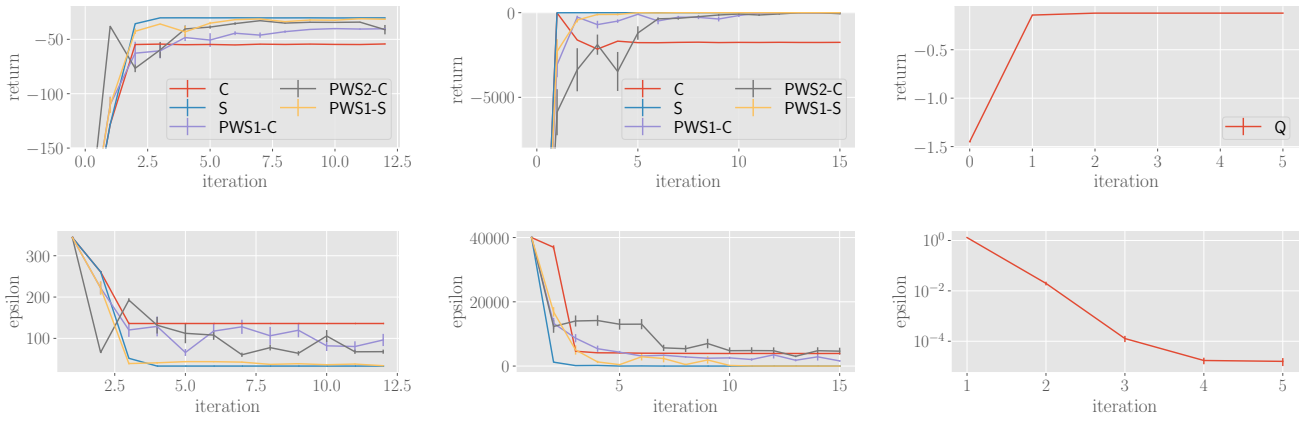


Figure 4: Simulated return (top row) and worst-case error ε^* (bottom row) on inventory (left column), reservoir (centre column) and VTOL (right column) over 100 independent roll-outs as a function of the number of iterations of constraint generation. Bars represent 95% confidence intervals calculated using 10 independent runs of CGPO.

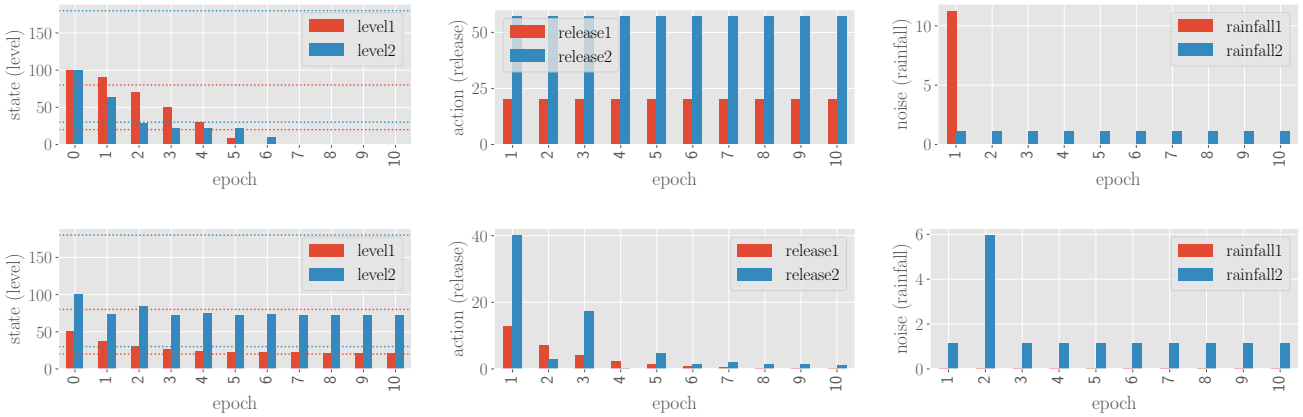


Figure 5: Worst-case state trajectory (left col.), actions (middle col.), and noise (right col.) for C (top row) and S policies (bottom row) computed by CGPO for the reservoir control problem.

performance for the optimal constant-value (C) policy occurs when the rainfall is low and both reservoirs become empty. Given that the cost of overflow exceeds underflow, this is expected as the C policy must release enough water to prevent high-cost overflow events during high rainfall at the risk of water shortages during droughts. In contrast, the optimal linear (S) policy maintains safe water levels even in the worst-case scenario, since it can avoid underflow and overflow events by assigning release amounts that increase linearly with the water levels.

Analysis of Problem Size To understand how the problem sizes in CGPO scale across differing expressivity classes of policies and dynamics, Table 2 shows the number of decision variables and constraints in the MIPs at the last iteration of constraint generation. For VTOL with a quadratic (Q) policy, the number of variables in the inner and outer problems are 0 / 0 / 212 and 0 / 0 / 512, respectively, while the number of constraints are 114 / 18 / 72 and 270 / 95 / 155. The number of decision variables/constraints are fixed in the inner problem and grow linearly with iterations in the outer problem. The number of variables and constraints do not grow significantly with the policy class expressiveness.

Conclusion

We presented constraint generation policy optimization (CGPO), a novel bi-level mixed-integer programming formulation for DC-MDP policy optimization in predefined expressivity classes with *bounded optimality guarantees*. We used constraint generation to decompose CGPO into an inner problem that produces an adversarial worst-case constraint violation (i.e., trajectory), and an outer problem that improves the policy w.r.t. these trajectories. On diverse domains and policy classes, we showed that the learned policies (some provably optimal) and their worst-case performance/failure modes were easy to interpret, while maintaining a manageable number of MIP variables and constraints.

References

Ahmed, A.; Varakantham, P.; Lowalekar, M.; Adulyasak, Y.; and Jaillet, P. 2017. Sampling based approaches for minimizing regret in uncertain markov decision processes (MDPs). *J. Artif. Int. Res.*, 59(1): 229–264.

Albert, L. A. 2022. A mixed-integer programming model for identifying intuitive ambulance dispatching policies. *Journal of the Operational Research Society*, 1–12.

Ariu, K.; Fang, C.; da Silva Arantes, M.; Toledo, C.; and Williams, B. C. 2017. Chance-Constrained Path Planning with Continuous Time Safety Guarantees. In *AAAI Workshop*.

Åström, K. J. 2012. *Introduction to stochastic control theory*. Courier Corporation.

Blankenship, J. W.; and Falk, J. E. 1976. Infinitely constrained optimization problems. *Journal of Optimization Theory and Applications*, 19: 261–281.

Bueno, T. P.; de Barros, L. N.; Mauá, D. D.; and Sanner, S. 2019. Deep reactive policies for planning in stochastic nonlinear domains. In *AAAI*, volume 33, 7530–7537.

Castro, P. M. 2015. Tightening piecewise McCormick relaxations for bilinear problems. *Computers & Chemical Engineering*, 72: 300–311.

Chembu, A.; Sanner, S.; Khurram, H.; and Kumar, A. 2023. Scalable and Globally Optimal Generalized L1 k-center Clustering via Constraint Generation in Mixed Integer Linear Programming. In *AAAI*, volume 37, 7015–7023.

Corso, A.; Moss, R.; Koren, M.; Lee, R.; and Kochenderfer, M. 2021. A survey of algorithms for black-box safety validation of cyber-physical systems. *Journal of Artificial Intelligence Research*, 72: 377–428.

Dolgov, D.; and Durfee, E. 2005. Stationary deterministic policies for constrained MDPs with multiple rewards, costs, and discount factors. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, 1326–1331. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Farias, V. F.; and Van Roy, B. 2006. *Tetris: A Study of Randomized Constraint Sampling*, 189–201. London: Springer London.

Farina, M.; Giulioni, L.; and Scattolini, R. 2016. Stochastic linear model predictive control with chance constraints–a review. *Journal of Process Control*, 44: 53–67.

Hauskrecht, M. 2006. Approximate Linear Programming for Solving Hybrid Factored MDPs. In *International Symposium on Artificial Intelligence and Mathematics, AI&Math 2006, Fort Lauderdale, Florida, USA, January 4-6, 2006*.

Low, S. M.; Kumar, A.; and Sanner, S. 2022. Sample-Efficient Iterative Lower Bound Optimization of Deep Reactive Policies for Planning in Continuous MDPs. In *AAAI*, volume 36, 9840–9848.

Morrison, D. R.; Jacobson, S. H.; Sauppe, J. J.; and Sewell, E. C. 2016. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19: 79–102.

Ono, M.; Pavone, M.; Kuwata, Y.; and Balaram, J. 2015. Chance-constrained dynamic programming with application to risk-aware robotic space exploration. *Autonomous Robots*, 39: 555–571.

Patton, N.; Jeong, J.; Gimelfarb, M.; and Sanner, S. 2022. A Distributional Framework for Risk-Sensitive End-to-End Planning in Continuous MDPs. In *AAAI*, volume 36, 9894–9901.

Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Raghavan, A.; Sanner, S.; Khardon, R.; Tadepalli, P.; and Fern, A. 2017. Hindsight optimization for hybrid state and action MDPs. In *AAAI*, volume 31.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramirez, M. 2016. Interval-based relaxation for general numeric planning. In *ECAI*, 655–663. IOS Press.

Scarf, H.; Arrow, K.; Karlin, S.; and Suppes, P. 1960. The optimality of (S, s) policies in the dynamic inventory problem. *Optimal pricing, inflation, and the cost of price adjustment*, 49–56.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *ArXiv*, abs/1707.06347.

Schuurmans, D.; and Patrascu, R. 2001. Direct value-approximation for factored MDPs. In Dietterich, T.; Becker, S.; and Ghahramani, Z., eds., *Advances in Neural Information Processing Systems*, volume 14. MIT Press.

Świechowski, M.; Godlewski, K.; Sawicki, B.; and Mańdziuk, J. 2023. Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3): 2497–2562.

Topin, N.; Milani, S.; Fang, F.; and Veloso, M. 2021. Iterative bounding mdps: Learning interpretable policies via non-interpretable methods. In *AAAI*, volume 35, 9923–9931.

Vos, D.; and Verwer, S. 2023. Optimal Decision Tree Policies for Markov Decision Processes. *arXiv preprint arXiv:2301.13185*.

Wang, Y.; Wang, J.; Zhang, W.; Zhan, Y.; Guo, S.; Zheng, Q.; and Wang, X. 2022. A survey on deploying mobile deep learning applications: A systemic and technical perspective. *Digital Communications and Networks*, 8(1): 1–17.

Wu, G.; Say, B.; and Sanner, S. 2017. Scalable planning with tensorflow for hybrid nonlinear domains. *NeurIPS*, 30.

Appendix

In this supplement to the main paper, we prove Theorem 1, and provide further insight about termination and practical implementation surrounding Algorithm 1. We also provide mathematical descriptions of the domains outlined in the main text. Next, we discuss experimental details and parameters that were not discussed in the main text, and we provide additional convergence/performance results and worst-case analysis for the domains that was left out of the main text due to space limitations. We also provide RDDL domain description files for the domains benchmarked in the text, and we conclude with a discussion of the related work that was left out of the main text due to space limitations.

Pseudocode of CGPO

Algorithm 1: Constraint-Generation Policy Optimization (CGPO)

Initialize $p \in (0,1)$, Ξ_p
Initialize $\mathbf{s}_1 \in \mathcal{S}_1$, $\xi_{1:T} \in \Xi_p^T$ and $\alpha \in \mathcal{A}^T$ arbitrarily, and constraint set $\mathcal{C}_0 = \{(\mathbf{s}_1, \xi_{1:T}, \alpha)\}$
Set $t = 0$
Step 1: Solve the **outer problem**:

$$(\mathbf{w}_t^*, \varepsilon_t^*) \in \operatorname{argmin}_{\mathbf{w} \in \mathcal{W},\, \varepsilon \in [0,\infty)} \varepsilon$$
$$\text{s.t.} \quad \varepsilon \geq V(\alpha, \mathbf{s}_1, \xi_{1:T}) - V(\mathbf{w}, \mathbf{s}_1), \quad \forall\, (\mathbf{s}_1, \xi_{1:T}, \alpha) \in \mathcal{C}_t$$

Step 2: Solve the **inner problem**:

$$(\mathbf{s}_1^*, \xi_{1:T}^*, \alpha^*) \in \operatorname{argmax}_{\mathbf{s}_1 \in \mathcal{S}_1,\, \xi_{1:T} \in \Xi_p^T,\, \alpha \in \mathcal{A}^T} \left[V(\alpha, \mathbf{s}_1, \xi_{1:T}) - V(\mathbf{w}_t^*, \mathbf{s}_1, \xi_{1:T})\right]$$

Step 3: Check convergence:
if $V(\mathbf{w}_t^*, \mathbf{s}_1^*, \xi_{1:T}^*) \geq V(\alpha^*, \mathbf{s}_1^*, \xi_{1:T}^*)$ **then**
  Terminate with policy $\pi_{\mathbf{w}_t^*}$ and error ε_t^*
else
  Set $\mathcal{C}_{t+1} = \mathcal{C}_t \cup \{(\mathbf{s}_1^*, \xi_{1:T}^*, \alpha^*)\}$, set $t = t+1$, and **go to Step 1**
end if

Linear Policy and Linear Dynamics Result in a Polynomial Class

Suppose both the policy and transition functions are linear in the state. Specifically, consider the transition function $s' = s + a$, and the policy $a = ws$. Then, defining s the initial state, s' the immediate successor state following s, and s'' the immediate successor state following s':

$$\begin{aligned}
s' &= s + a = s + ws = (w+1)s \\
a' &= ws' = w(w+1)s = O(w^2) \\
s'' &= s' + a' = (w+1)s + w(w+1)s = (w+1)^2 s \\
a'' &= ws'' = w(w+1)^2 s = O(w^3),
\end{aligned}$$

which are polynomial in w.

Proof of Theorem 1

In the notation of Blankenship and Falk [1976], we define $X^0 = [0, 2BT]$, $X = \mathcal{W}$, $Y = \mathcal{A}^T \times \mathcal{S}_1 \times \Xi_p^T$. Making the change of variable $\varepsilon = x_0 \in X^0$, $x = \mathbf{w}$, and $y = (\alpha, \mathbf{s}_1, \xi_{1:T}) \in Y$, and defining the continuous function $f(x,y) = \varepsilon(\alpha, \mathbf{s}_1, \xi_{1:T}) = V(\alpha, \mathbf{s}_1, \xi_{1:T}) - V(\mathbf{w}, \mathbf{s}_1, \xi_{1:T})$, the problem (7) is equivalent to the problem:

$$\min_{(x_0, x) \in X^0 \times X} x_0$$
$$\text{s.t. } f(x,y) - x_0 \leq 0, \quad \forall y \in Y$$

as discussed on p. 262 of Blankenship and Falk [1976]. Similarly, our Algorithm 1 is a special case of the Algorithm 2.1 in the aforementioned paper, specialized to the problem state, with $x_t = \mathbf{w}_t^*$. Thus, the assumption of Theorem 2.1 in Blankenship and Falk [1976] holds, and $\mathbf{w}_t^*$ is a solution of (7) as claimed. □

Further Remarks About Theorem 1

In general, Algorithm 1 may not necessarily terminate, unless much stronger convexity requirements are placed on $V(\alpha, \mathbf{s})$ and $V(\mathbf{w}, \mathbf{s})$, and potentially the class of MDPs solved. Thus, in a typical application, one can instead monitor the solution differences $\|\mathbf{w}_t^* - \mathbf{w}_{t-1}^*\|$, and terminate with an *approximate* solution when this difference is sufficiently small. However, our experiments run a fixed number of iterations, showing empirically that Algorithm 1 does indeed terminate on all problems with an exact optimal solution for most policy classes considered, at which point both the return and error curves "plateau" (cf. Figure 4).

Domains

Here we provide more complete mathematical descriptions of the benchmark problems used in the experiments. In what follows, we define $(\cdot)_+ = \max[0, \cdot]$.

Linear Inventory The state transition model can be written as:

$$\begin{aligned} & s_{t+1} = s_t + a_t - \xi_t, \quad \xi_t \sim \lfloor \text{Uniform}(L, U) \rfloor, \\ & \text{s.t.} \quad 0 \leq a_t \leq B, \quad \forall t. \end{aligned}$$

We define costs C, P and S which represent, respectively, the purchase cost, excess inventory cost and shortage cost. The reward function can thus be written as

$$r(\mathbf{s}_{t+1}) = -C \times a_t - P \times (s_{t+1})_+ - S \times (-s_{t+1})_+.$$

We set $B = 10$, $C = 0.5$, $P = S = 2$, $L = 2$, $U = 6$. If $P > C$ and $T = \infty$, then a PWS-S policy is provably optimal (otherwise if $T < \infty$, then it may be non-stationary).

Linear Reservoir Let $s_{t,r}$ denote the water level of reservoir r, $a_{t,r}$ be the amount of water to release from reservoir r, $\xi_{t,r}$ denote the stochastic rainfall amount affecting reservoir r, $U(r)$ denote the set of upstream reservoirs connected to reservoir r, and M_r be the maximum capacity of reservoir r. The state transition model can be written as:

$$\begin{aligned} & s_{t+1,r} = \min \left[M_r, \left(s_{t,r} + (\xi_{t,r})_+ - a_{t,r} + \sum_{d \in U(r)} a_{t,d} \right)_+ \right], \\ & \xi_{t,r} \sim \text{Normal}(m_r, v_r), \\ & \text{s.t.} \quad 0 \leq a_{t,r} \leq s_{t,r}, \quad \forall t, r. \end{aligned}$$

The goal is to maintain the level of water within target range $[L_r, H_r]$. Penalties l_r and h_r are assigned for any quantity of water above or below the required target level. Thus, the reward function can be written as

$$r(\mathbf{s}_{t+1}) = \sum_{r=1}^{N} l_r (L_r - s_{t+1,r})_+ + h_r (s_{t+1,r} - H_r)_+.$$

Our experiment uses a two-reservoir system with the following values for reservoir 1 $L_1 = 20, H_1 = 80, M_1 = 100, m_1 = 5, v_1 = 5, U(1) = \emptyset$, and the following values for reservoir 2 $L_2 = 30, H_2 = 180, M_2 = 200, m_2 = 10, v_2 = 10, U(2) = \{1\}$. Costs are shared and set to $l_r = -10, h_r = -100$.

Nonlinear VTOL Control The state dynamics evolve according to:

$$\begin{aligned} \theta_{t+1} &= \max \left[-\sin(h/l_1), \min \left[\sin(h/l_2), \theta_t + \tau \omega_t \right] \right] \\ \omega_{t+1} &= \omega_t + \frac{\tau}{J} \left(9.8(m_2 l_2 - m_1 l_1) \cos \theta_t + 150 l_1 F_t \right) \\ J &= m_1 l_1^2 + m_2 l_2^2 \\ \text{s.t.} \quad & l_1 m_1 \geq l_2 m_2, \quad l_1 > l_2 > h, \quad 0 \leq F_t \leq 1. \end{aligned}$$

The reward is defined as the negative absolute difference between the angle at each epoch and the target angle $\theta_{target} = 0$, i.e.

$$r(F_t, \theta_{t+1}, \omega_{t+1}) = -|\theta_{t+1} - \theta_{target}|.$$

We use values $l_1 = 1, l_2 = 0.5, m_1 = 10, m_2 = 1, h = 0.4, g = 9.8$.

Quadratic policies are of the form

$$F(\theta_t, \omega_t) = w_0 + w_1 \theta_t + w_2 \omega_t + w_3 \theta_t \omega_t + w_4 \theta_t^2 + w_5 \omega_t^2,$$

where w_i are bounded in $[-100, 100]$. Since coefficients are allowed to be both positive and negative, this increases the expressivity of the policy class at the expense of potentially inducing *non-convexity* of the objective.

Nonlinear Intercept The intercept problem is a nonlinear domain with Boolean actions. It involves two objects moving in continuous trajectories in a subset of $\mathbb{R}^2$, in which one object (e.g. missile, bird) flies in a parabolic arc across space towards the ground, and must be intercepted by a second object (e.g. anti-ballistic missile, predator) that is fired from a fixed position on the ground. While the problem is best described in continuous time, we study a discrete-time version of the problem. The state includes the position (x_t, y_t) of the missile at each decision epoch, as well as "work" variables indicating whether the interceptor has already been fired f_t as well as its vertical elevation i_t. Meanwhile, the action a_t is Boolean-valued and indicates whether the interceptor is fired at a given time instant.

The state transition model can be written as:

$$x_{t+1} = x_t + v_x, \quad y_{t+1} = h - \frac{1}{2} g x_t^2$$
$$f_{t+1} = f_t \vee a_t, \quad i_{t+1} = \begin{cases} i_t & \text{if } f_t = 0 \\ i_t + v_y & \text{if } f_t = 1 \end{cases},$$

ignoring the gravitational interaction of the interceptor. Interception happens whenever the absolute differences between the coordinates of the missile and the interceptor are within δ, so the reward is

$$r(s_{t+1}) = \begin{cases} 1 & \text{if } f_{t+1} \wedge |x_{t+1} - i_x| \leq \delta \wedge |y_{t+1} - i_{t+1}| \leq \delta \\ 0 & \text{otherwise} \end{cases}.$$

We set $v_x = 0.1, h = 5, g = 9.8, i_x = 0.7$.

We study Boolean-valued policies with linear constraints (i.e., B, PWL-B) that take into account the position of the missile at each epoch, i.e. PWL1-B policies of the form

$$a_t(x_t, y_t) = \begin{cases} a_1 & \text{if } l \leq w_1 x_t + w_2 y_t \leq u \\ a_2 & \text{otherwise} \end{cases}$$

where $a_1, a_2 \in \{0, 1\}$ and l, u, w_1, w_2 are all tunable parameters.

Initial State Bounds $\mathcal{S}_1$ For reservoir and inventory control, bounds on the initial state have been chosen as intervals centred on the initial state specified in the RDDL domain description. For reservoir control, these bounds are intervals centred around the initial state $(75, 150)$, i.e. $[50, 100]$ for reservoir 1 and $[100, 200]$ for reservoir 2. For inventory control, the bounds are $[0, 2]$, for VTOL they are fixed to the initial state of the system $\theta = 0.1, \omega = 0$, and for intercept they are also fixed to the initial state $x = 0, y = 5, f = 0, i = 0$.

Gurobi and Environment Settings

We use the Python implementation of the Gurobi Optimizer (GurobiPy) version 10.0.1, build v10.0.1rc0 for Windows 64-bit systems, with an academic license. Default optimizer settings have been used, with the exception of NumericFocus set to 2 in order to enforce numerical stability, and the MIPGap parameter set to 0.05 to terminate when the optimality gap reaches 5%. All experiments were conducted on a single machine with an Intel 10875 processor (base at 2.3 GHz, overclocked at 5.1 GHz) and 32 GB of RAM.

Additional Implementation Details

Variable Bounds In order to compute the tightest possible bounds on decision variables in the MIP compilation, we use interval arithmetic [Hickey et al. 2001].

Action Constraints One important issue concerns how to enforce constraints on valid actions. Two valid approaches include (1) explicitly constraining actions in RDDL constraints (state invariants and action preconditions) by compiling them as MIP constraints, or (2) implicitly clipping actions in the state dynamics (conditional probability functions in RDDL). Crucially, we found the latter approach performed better during optimization, since constraining actions inherently limits the policy class to a subset of weights that can only produce legal actions across the initial state space. This not only significantly restricts policies (i.e. for linear valued policies, the weights would be constrained to a tight subset where the output for every possible state would be a valid action) but also poses challenges for the optimization process, which is significantly complicated by these action constraints.

Handling Numerical Errors Mathematical operations, such as strict inequalities, cannot be handled explicitly in Gurobi. To perform accurate translation of such operations in our code-base, we used indicator/binary variables. For instance, to model the comparison $a > b$ for two numerical values a, b, we define a binary variable $y \in \{0, 1\}$ and error parameter $\epsilon > 0$ constrained as follows:

$$y == 1 \implies a \geq b + \epsilon$$
$$y == 0 \implies a \leq b.$$

In practice, ϵ is typically set larger than the smallest positive floating point number in Gurobi (around 10^{-5}), but is often problem-dependent. We derived optimal policies for all domains using $\epsilon = 10^{-5}$, except Intercept where we needed to use a larger value of $\epsilon = 10^{-3}$ to allow Gurobi to correctly distinguish between the two cases above.

Additional Experimental Results

Intercept Fig. 6 illustrates the return and worst-case error of Boolean-valued policies for the intercept problem. Only the policies with at least one case are optimal, with corresponding error of zero. As illustrated in the last plot in Fig. 7, the optimal policies fire whenever a threat is detected in the top right corner of the airspace.

Policy Representations Fig. 8 and Fig. 9 showcase policies learned in other expressivity classes besides those shown in the main text, for inventory and reservoir control, respectively. Fig. 7 illustrates the policies for the intercept problem. As can be seen, policies generally remain intuitive and easy to explain even as the complexity increases. For instance, PWS2-C policies for inventory control order more inventory as the stock level decreases, while the amount of water released is increasing in the water level for reservoir control. Interestingly, the PWS2-C policies for inventory control and the intercept problem contain redundant constraints.

Worst-Case Analysis Fig. 10 and Fig. 11 illustrate worst case scenarios induced by the C and S policies for inventory and VTOL control, respectively. Similar to reservoir control, a non-trivial worst-case scenario is identified by CGPO for inventory control for optimal C policies but not S policies. This scenario corresponds to very high demand that exceeds the constant reorder quantity, causing stock-outs. For the VTOL control problem, there is no worst-case scenario with non-zero cost upon convergence at the final iteration of CGPO (bottom row). We have also shown the worst-case scenario computed at the first iteration prior to achieving policy optimality, where a worst-case scenario corresponds to no force being applied causing the system to eventually lose balance. The ability to interpret the worst-case scenario prior to convergence, or if convergence cannot be achieved, could help to decide whether the policy is trustworthy, and how to prepare for the worst-case if the policy is deployed in practice.

Analysis of Problem Size Fig. 12, Fig. 13 and Fig. 14 summarize the total number of variables and constraints in the outer MIP formulations solved by CGPO at each iteration for inventory, reservoir and VTOL control, respectively. Each iteration of constraint generation requires a roll-out from the dynamics and reward model, which in turn requires instantiate a new set of variables to hold the intermediate state and reward computations, and thus the number of variables and constraints grows linearly with the iterations. Even at the last iteration of CGPO, we see that the number of variables and constraints remains manageable.

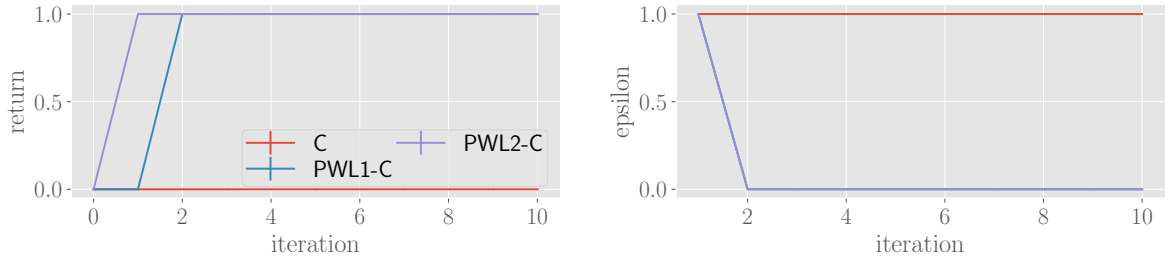


Figure 6: Simulated return (left) and optimal value of ε^* (right) as a function of the number of iterations of constraint generation for the intercept domain.

RDDL Descriptions

The *Relational Dynamic influence Diagram Language* (RDDL) is a structured planning domain description language particularly well-suited for modelling problems with stochastic effects [Sanner 2010]. Our domains are therefore coded in RDDL (as provided below) and we wrote a general-purpose routine for compiling RDDL code into a Gurobi mixed integer program formulation using the `pyRDDLGym` interface [Taitler et al. 2022].

`inventory.rddl`

```
// Linear inventory control problem

domain inventory {
```

$$fire = 1 \qquad fire = \begin{cases} 0 & \text{if } -0.136 \leq -0.2x - 0.02y \leq 0 \\ 1 & \text{otherwise} \end{cases} \qquad fire = \begin{cases} 0 & \text{if } -0.136 \leq -0.2x - 0.02y \leq 0 \\ 0 & \text{if } 0.001 \leq -x \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

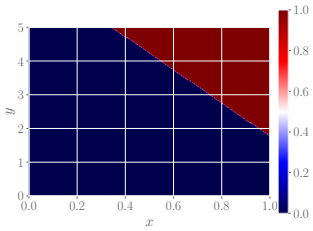


Figure 7: From left to right, examples of learned B, PWS1-B and PWS2-B policies and visualization of PWS-B policy for the intercept problem.

$$order = 3 \qquad order = 4 - stock \qquad order = \begin{cases} 8 & \text{if } -13 \leq stock \leq -2 \\ 2 & \text{otherwise} \end{cases} \qquad order = \begin{cases} 6 & \text{if } -10 \leq stock \leq -1 \\ 1 & \text{if } 2 \leq stock \leq 2 \\ 3 & \text{otherwise} \end{cases}$$

Figure 8: From left to right, examples of learned C, S, PWS1-C and PWS2-C policies for the inventory control problem.

$$release_1 = 19.73 \qquad release_1 = level_1 - 20$$
$$release_2 = 55.87 \qquad release_2 = 0.93 \times level_2 - 29.74$$

$$release_1 = \begin{cases} 1.39 & \text{if } 30.62 \leq level_1 \leq 68.66 \\ 40.71 & \text{if } 70.93 \leq level_1 \leq 100 \\ 10.622 & \text{otherwise} \end{cases}$$
$$release_2 = \begin{cases} 80.78 & \text{if } 159.93 \leq level_2 \leq 200 \\ 38.69 & \text{if } 144.21 \leq level_2 \leq 159.11 \\ 18.25 & \text{otherwise} \end{cases}$$

$$release_1 = \begin{cases} level_1 - 50 & \text{if } 20 \leq level_1 \leq 100 \\ 2 \times level_1 - 100 & \text{otherwise} \end{cases}$$
$$release_2 = \begin{cases} level - 2 - 37.8 & \text{if } 38.9 \leq level_2 \leq 200 \\ 1.17 \times level_2 - 235.5 & \text{otherwise} \end{cases}$$

Figure 9: From left to right, examples of learned C, S, PWS2-C and PS1-S policies for the reservoir control problem.

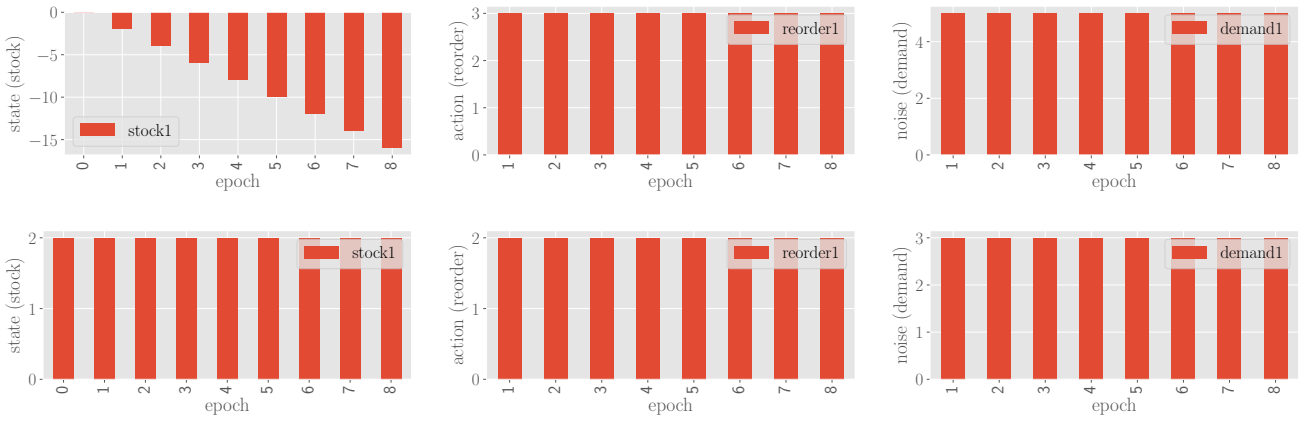


Figure 10: Worst-case state trajectory (left col.), actions (middle col.), and disturbances/noise (right col.) for C (top row) and S policies (bottom row) computed by Algorithm 1 for the inventory control problem.

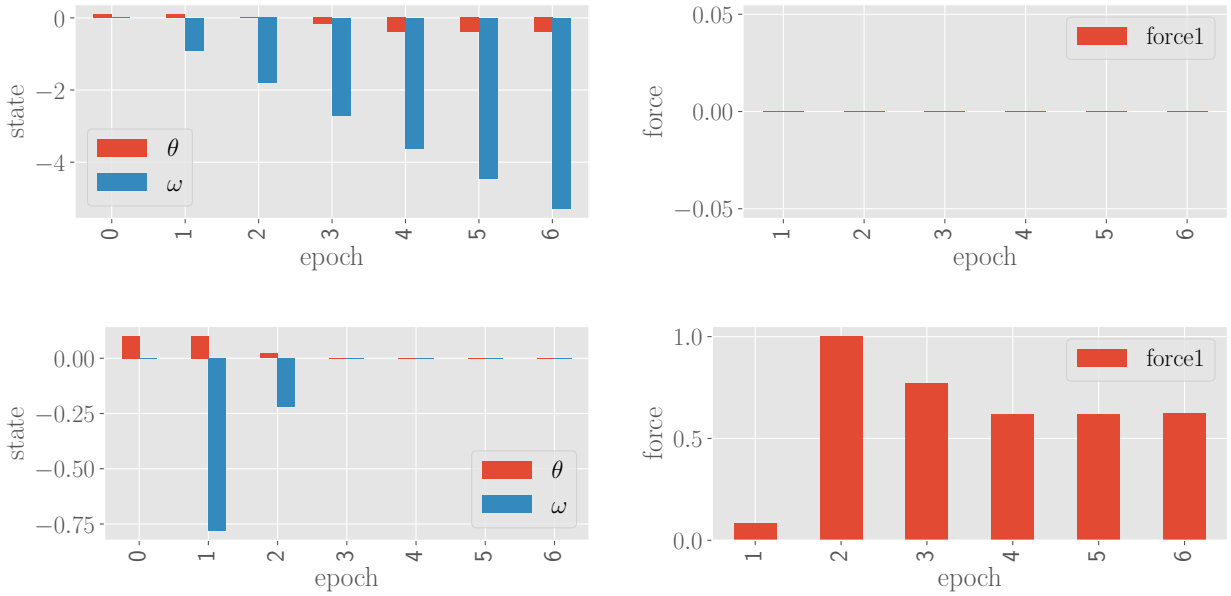


Figure 11: For VTOL control, worst-case state trajectory (top row), and actions $\mathbf{a}_1^*, \ldots \mathbf{a}_T^*$ (bottom row) for Q policies. First (last) column corresponds to the first (last) iterations of Algorithm 1.

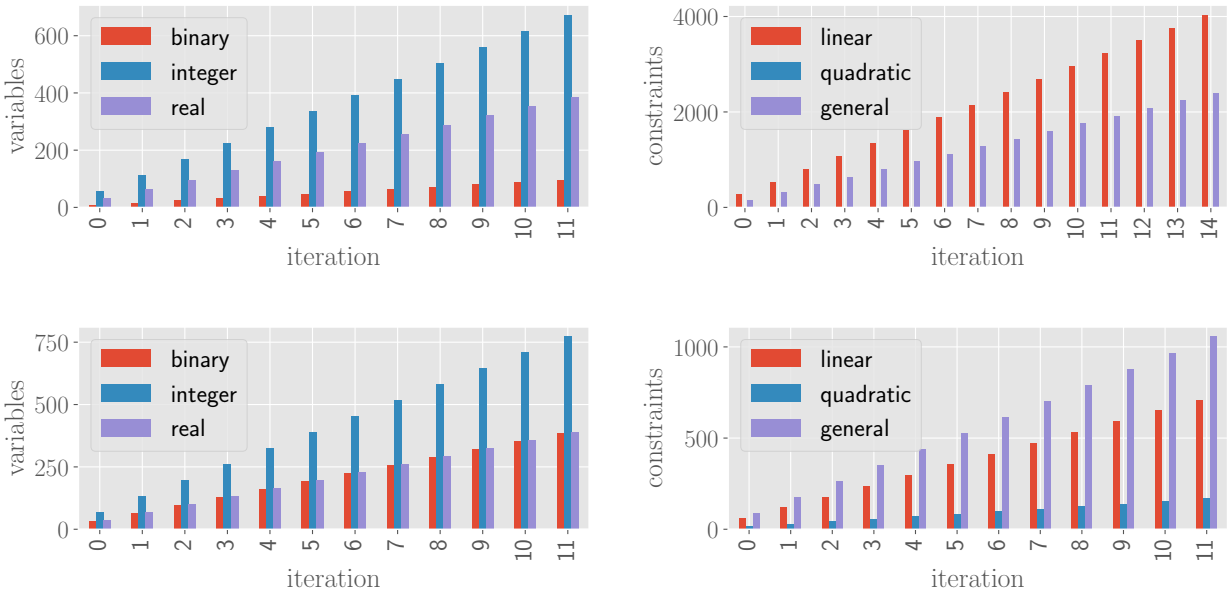


Figure 12: Number of decision variables (left) and constraints (right) for C policy (top row) and PWS1-S policy (bottom row) corresponding to the outer optimization problem at each iteration applied to the inventory problem.

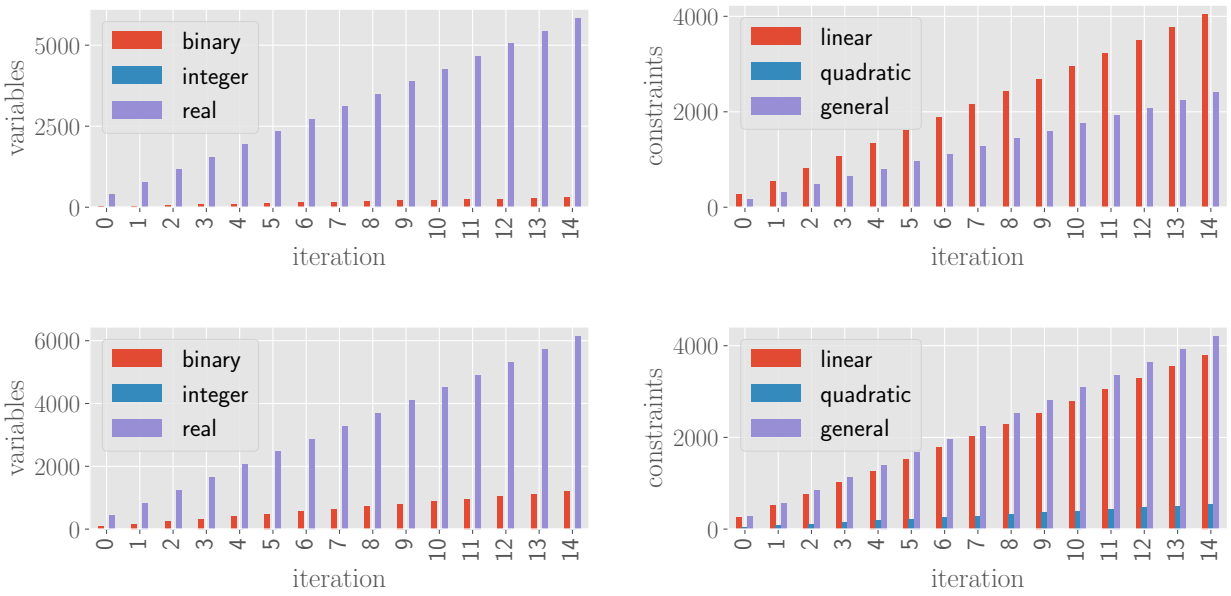


Figure 13: Number of decision variables (left) and constraints (right) for C policy (top row) and PWS1-S policy (bottom row) corresponding to the outer optimization problem at each iteration applied to the reservoir problem.

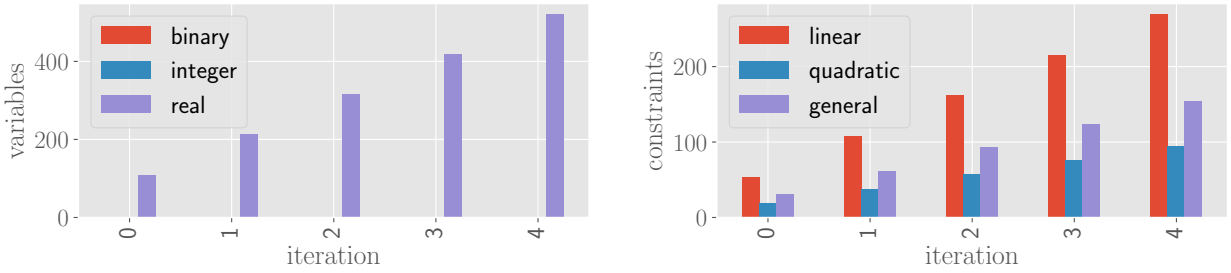


Figure 14: Number of decision variables (left) and constraints (right) for Q policy corresponding to the outer optimization problem at each iteration applied to the VTOL problem.

```
        types {
                item : object;
        };

        pvariables {

                CAP : { non-fluent, int, default = 10 };
                C(item) : { non-fluent, real, default = 0.5 };
                P(item) : { non-fluent, real, default = 2.0 };

                MEAN(item) : { non-fluent, real, default = 4.0 };
                DISP(item) : { non-fluent, real, default = 2.0 };

                demand(item) : { interm-fluent, int };
                order-act(item) : { interm-fluent, int };

                stock(item) : { state-fluent, int, default = 0 };

                order(item) : { action-fluent, int, default = 0 };
        };

        cpfs {
                order-act(?i) = max[0, min[order(?i), CAP]];
                demand(?i) = floor[Uniform(MEAN(?i) - DISP(?i), MEAN(?i) + DISP(?i))];
                stock'(?i) = stock(?i) + order-act(?i) - demand(?i);
        };

        reward = (sum_{?i : item} [(-C(?i)) * order-act(?i) + (-P(?i)) * abs[stock'(?i)]]);

        action-preconditions {
        };

        state-invariants {
        };
}
```

reservoir.rddl

```
domain reservoir_control_linear {

        requirements = {
                reward-deterministic
        };

        types {
                id: object;
        };

        pvariables {

                LOW_BOUND(id) : { non-fluent, real, default = 20.0 };
                HIGH_BOUND(id) : { non-fluent, real, default = 80.0 };
                MAXCAP(id) : { non-fluent, real, default = 100.0 };

                LOW_COST(id) : { non-fluent, real, default = -10.0 };
                HIGH_COST(id) : { non-fluent, real, default = -100.0 };

                RAIN_MEAN(id) : { non-fluent, real, default = 5.0 };
                RAIN_VAR(id) : { non-fluent, real, default = 5.0 };

                DOWNSTREAM(id, id) : { non-fluent, bool, default = false };

                rainfall(id) : { interm-fluent, real };
```

```
                release-act(id) : { interm-fluent, real };

                rlevel(id): { state-fluent, real, default = 50.0 };

                release(id): { action-fluent, real, default = 0.0 };
        };

        cpfs {
                rainfall(?r) = max[0, Normal(RAIN_MEAN(?r), RAIN_VAR(?r))];
                release-act(?r) = max[0, min[release(?r), rlevel(?r)]];
                rlevel'(?r) = max[0, min[MAXCAP(?r),
                        rlevel(?r) + rainfall(?r) - release-act(?r)
                        + (sum_{?r2: id}[DOWNSTREAM(?r2, ?r) * release-act(?r2)])]];
        };

        reward = (sum_{?r: id} [LOW_COST(?r) * max[LOW_BOUND(?r) - rlevel'(?r), 0]
                            + HIGH_COST(?r) * max[rlevel'(?r) - HIGH_BOUND(?r), 0]]);

        action-preconditions {
        };

        state-invariants {
        };
}
```

vtol.rddl

```
////////////////////////////////////////////////////////////////////
// A simple continuous state-action MDP for the classical VTOL system
//
// The goal here is to apply a force by a fan connected to a rod
// with weight, so the rod will stabilize on a goal angle
//
////////////////////////////////////////////////////////////////////
domain VTOL_continuous {

requirements = {
        reward-deterministic
    };

pvariables {

    l1 : { non-fluent, real, default = 1.0 };     // length of rod from pole to fan
    l2 : { non-fluent, real, default = 0.5 };     // length of rod from pole to weight
    m1 : { non-fluent, real, default = 10.0 };   // the mass of the fan
    m2 : { non-fluent, real, default = 1.0 };     // the mass of the weight
    h : { non-fluent, real, default = 0.4 };    // height of the pole
    goal : {non-fluent, real, default = 0.0 };  // the goal angle to stabilize on
    g : { non-fluent, real, default = 9.8 };    // gravity constant
    T : { non-fluent, real, default = 0.1 };    // time sampling constant
    FORCE-MAX : {non-fluent, real, default = 1.0 }; // maximum force applied by the fan

    theta : { state-fluent, real, default = 0.0 };    // angle of rod in relation to x axis
    omega : { state-fluent, real, default = 0.0 };    // the angular velocity of the rod

        F-act : { interm-fluent, real };

    F : { action-fluent, real, default = 0.0 };       // force applied by the fan
};

cpfs {
        F-act = max[-FORCE-MAX, min[FORCE-MAX, F]];
    theta' =  max[-sin[h / l1], min[sin[h / l2], theta + T * omega]];
```

```
    omega' = omega + (T / (m1 * l1 * l1 + m2 * l2 * l2))
                * (g * (m2 * l2 - m1 * l1) * cos[theta] + 150 * l1 * F-act);
};

reward = -abs[theta' - goal];

action-preconditions {
};

state-invariants {
};

}
```

intercept.rddl

```
// Missile interception problem
domain interception {

    pvariables {
                G : { non-fluent, real, default = 9.8 };
                INTERCEPT-X : { non-fluent, real, default = 0.7 };

                missile-x : { state-fluent, real, default = 0 };
                missile-y : { state-fluent, real, default = 5 };
                intercept-fired : { state-fluent, bool, default = false };
                intercept-y : { state-fluent, real, default = 0 };

                fire : { action-fluent, bool, default = false };
        };

        cpfs {
                missile-x' = missile-x + 0.1;
                missile-y' = 5.0 - 0.5 * G * missile-x * missile-x;
                intercept-fired' = intercept-fired | fire;
                intercept-y' = intercept-y + 1.0 * intercept-fired;
        };

        reward = intercept-fired'
                        ^ (abs[missile-x' - INTERCEPT-X] <= 0.2)
                        ^ (abs[missile-y' - intercept-y'] <= 0.2);

        action-preconditions {
        };

        state-invariants {
        };
}
```

Additional Related Work

The scheme of mixed discrete-continuous models can be traced back to the hybrid automata [Henzinger 1996], and how to verify reachability and model checking [Henzinger et al. 1998]. In [Ferretti et al. 2014] a policy iteration to approach has been taken to synthesize a feedback controller to a continuous time system with discrete jumps. Model Predictive Control (MPC) approaches are especially appealing in the hybrid setting for controller synthesis [Borrelli et al. 2017], however they are not employed for worst-case analysis and policy optimization as in this work.

(Chance-) Constrained Optimization in Hybrid Systems: The tool of constrained optimization is popular in policy optimization for hybrid systems [Borrelli 2003]. For instance, [Sato et al. 2021] posed the problem of conjunctive synthesis and system falsification as constrained optimization. In the probabilistic or uncertain case, chance-constraints are popular for finding a robust policy; [Blackmore et al. 2010] used predictive control to synthesis the optimal controller in expectation under chance constraints. [Petsagkourakis et al. 2022] utilized an recurrent neural network as parameterized policy for policy optimization under chance-constraints. Our work differs from the above as we optimize a given policy structure, under the worst case, thus providing guarantees on the bounded policy optimality for the chosen policy class (also providing interpretability of the final optimized policy, due to the compactness of the policy class).

Constrained Policy Optimization in Reinforcement Learning: A different stream of work incorporates safety constraints on parameterized policies [Achiam et al. 2017, Liu et al. 2021, Polosky et al. 2022]. However, they typically only guarantee convergence through (policy) gradient based methods, and not bounded policy optimality as in our work. Furthermore, to accomplish this, they require differentiable – and often non-compact policy classes (i.e. such as neural network) – which makes them unsuitable for directly optimizing piecewise policy classes or other compact policy classes with discrete structure as studied in this work.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *ICML*, pages 22–31. PMLR, 2017.

Lars Blackmore, Masahiro Ono, Askar Bektassov, and Brian C. Williams. A probabilistic particle-control approximation of chance-constrained stochastic predictive control. *IEEE Transactions on Robotics*, 26(3):502–517, 2010. doi: 10.1109/TRO.2010.2044948.

Jerry W Blankenship and James E Falk. Infinitely constrained optimization problems. *Journal of Optimization Theory and Applications*, 19:261–281, 1976.

Francesco Borrelli. *Constrained optimal control for hybrid systems*. Springer, 2003.

Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

Roberto Ferretti, Achille Sassi, and Hasnaa Zidani. Recent advances in the numerical analysis of optimal hybrid control problems. *HAL*, 2014, 2014.

Thomas A Henzinger. The theory of hybrid automata. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292. IEEE, 1996.

Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE transactions on automatic control*, 43(4):540–554, 1998.

Timothy Hickey, Qun Ju, and Maarten H Van Emden. Interval arithmetic: From principles to implementation. *Journal of the ACM*, 48(5):1038–1068, 2001.

Tao Liu, Ruida Zhou, Dileep Kalathil, PR Kumar, and Chao Tian. Policy optimization for constrained mdps with provable fast global convergence. *arXiv preprint arXiv:2111.00552*, 2021.

Panagiotis Petsagkourakis, Ilya Orson Sandoval, Eric Bradford, Federico Galvanin, Dongda Zhang, and Ehecatl Antonio del Rio-Chanona. Chance constrained policy optimization for process control and optimization. *Journal of Process Control*, 111: 35–45, 2022.

Nicholas Polosky, Bruno C Da Silva, Madalina Fiterau, and Jithin Jagannath. Constrained offline policy optimization. In *ICML*, pages 17801–17810. PMLR, 2022.

Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, 32:27, 2010.

Sota Sato, Masaki Waga, and Ichiro Hasuo. Constrained optimization for hybrid system falsification and application to conjunctive synthesis. *IFAC-PapersOnLine*, 54(5):217–222, 2021. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2021.08.501. URL https://www.sciencedirect.com/science/article/pii/S2405896321012763. 7th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2021.

Ayal Taitler, Michael Gimelfarb, Sriram Gopalakrishnan, Martin Mladenov, Xiaotian Liu, and Scott Sanner. pyrddlgym: From rddl to gym environments. *arXiv preprint arXiv:2211.05939*, 2022.