Benchmarking Constraint Inference in Inverse Reinforcement Learning

Abstract

When deploying Reinforcement Learning (RL) agents into a physical system, we must ensure that these agents are well aware of the underlying constraints. In many real-world problems, however, the constraints followed by expert agents (e.g., humans) are often hard to specify mathematically and unknown to the RL agents. To tackle these issues, Constraint Inverse Reinforcement Learning (CIRL) considers the formalism of Constrained Markov Decision Processes (CMDPs) and estimates constraints from expert demonstrations by learning a constraint function. As an emerging research topic, CIRL does not have common benchmarks, and previous works tested their algorithms with hand-crafted environments (e.g., grid worlds). In this paper, we construct a CIRL benchmark in the context of two major application domains: robot control and autonomous driving. We design relevant constraints for each environment and empirically study the ability of different algorithms to recover those constraints based on expert trajectories that respect those constraints. To handle stochastic dynamics, we propose a variational approach that infers constraint distributions, and we demonstrate its performance by comparing it with other CIRL baselines on our benchmark. The benchmark, including the information for reproducing the performance of CIRL algorithms, is publicly available at *temporarily hidden due to the anonymous policy*.

1 Introduction

Constrained Reinforcement Learning (CRL) algorithms [1] typically learn a policy under a Constrained Markov Decision Process (CMDP) by assuming known constraints. This assumption, however, is not realistic in many real-world problems where it is difficult to specify the exact constraints that an agent should follow, especially when these constraints are time-varying, context-dependent and inherent to experts' own experience, and further, such information may not be completely revealed to the agent. For example, human drivers tend to determine an implicit speed limit and a minimum gap to other cars based on the traffic conditions, rules of the road, weather and social norms. To derive a driving policy that matches human performance, an autonomous agent needs to infer these constraints from expert demonstrations.

An important approach to recovering the underlying constraints is Constraint Inverse Reinforcement Learning (CIRL) [2, 3, 4, 5]. CIRL infers a set of legal state-action pairs or a constraint function (in the continuous case) to approximate constraints respected by expert demonstrations. This is often done by alternating between updating an imitating policy and a constraint function (or set). Figure 1 summarizes the main procedure of CIRL. As an emerging

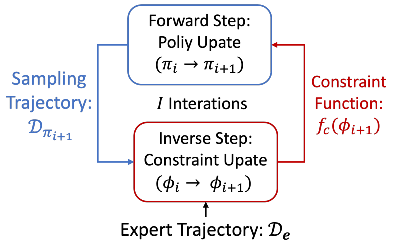
Forward Step:
Poliy Upate
$(\pi_i \rightarrow \pi_{i+1})$
Sampling Trajectory: $\mathcal{D}_{\pi_{i+1}}$
I Interations
Constraint Function: $f_c(\phi_{i+1})$
Inverse Step:
Constraint Upate
$(\phi_i \rightarrow \phi_{i+1})$
Expert Trajectory: $\mathcal{D}_e$

Figure 1: The flowchart of CIRL.

Submitted to the 36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks. Do not distribute.

research topic, CIRL does not have common datasets and benchmarks for evaluation. Existing validation methods heavily depend on the safe-Gym [6] and hand-crafted grid-world environments. Utilizing these environments has some important drawbacks: 1) These environments are designed for control instead of constraint inference. To fill this gap, previous works often pick some environments and add hand-crafted constraints to them. Striving for simplicity, many of the selected environments have discretized state and action spaces of limited size (e.g., grid worlds [2, 4, 5, 7, 8]). Generalizing model performance in these simple environments to practical applications will be difficult. 2) These environments do not include expert demonstrations, and thus previous works manually generate their own trajectories. Since the quality of expert trajectories can significantly influence model performance, it is difficult to compare different CIRL algorithms without a consistent expert dataset.

In this paper, we propose a benchmark for evaluating Constraint Inverse Reinforcement Learning (CIRL) algorithms. This benchmark includes both virtual environments and realistic environments. The virtual environments are based on MuJoCo [9], but we update some of these robot control tasks by modifying reward functions and including specific constraints. The realistic environments are constructed based on a highway vehicle tracking dataset [10], so the environments can suitably reflect what happens in realistic driving scenario. We consider constraints about car velocities and distances in these realistic environments. Each of these environments is accompanied with a demonstration dataset generated by an expert agent trained by the Reward Constrained Policy Optimization (RCPO) [11]. To demonstrate that the added constraints are important for control, we empirically compare the performance of agents trained with and without the constraints.

In the context of stochastic environments, we propose a Variational Constraint Inverse Reinforcement Learning (VCIRL) algorithm. VCIRL infers distributions of constraints induced by the stochasticity in-game dynamics and imitating policies. We demonstrate the performance of VCIRL by comparing its rewards and constraint violation rate with other CIRL baselines in our virtual and realistic environments. Empirical results show that the policies learned with the VCIRL constraints can outperform others in terms of safety and reward accumulation.

2 Background

We introduce some notations about Constrained Reinforcement Learning (CRL) and its inverse problem, CIRL as well as their connections to other related methods.

2.1 Constrained Reinforcement Learning

Constrained Reinforcement Learning (CRL) is based on Constrained Markov Decision Processes (CMDPs) $\mathcal{M}^c$, which can be defined by a tuple $(\mathcal{S}, \mathcal{A}, p_\mathcal{R}, p_\mathcal{T}, \{(p_{\mathcal{C}_i}, \epsilon_i)\}_{\forall i}, \gamma, h)$ where: 1) $\mathcal{S}$ and $\mathcal{A}$ denote the space of states and actions. 2) $p_\mathcal{T}(s'|s,a)$ and $p_\mathcal{R}(r|s,a)$ define the transition and reward distributions. 3) $p_{\mathcal{C}_i}(c|s,a)$ denotes a stochastic constraint function with an associated bound ϵ_i, where i indicates the index of a constraint. 4) $\gamma \in [0,1)$ is the discount factor and h is the planning horizon (usually infinite). The goal of CRL is to find a (deterministic or stochastic) policy π that maximizes expected discounted rewards under a set of cumulative soft constraints:

$$\arg\max_\pi \mathbb{E}_{p_\mathcal{R}, p_\mathcal{T}, \pi}\left[\sum_{t=0}^{h} \gamma^t r_t\right] \text{ s.t. } \mathbb{E}_{p_{\mathcal{C}_i}, p_\mathcal{T}, \pi}\left[\sum_{t=0}^{h} \gamma^t c_{i,t}\right] \leq \epsilon_i \ \forall i \in [0, I] \quad (1)$$

In the above formulation, constraints consist of bounds on the expectation of cumulative constraint values. In practice, it is often more desirable to express bounds on the cumulative constraint value of a single trajectory instead of the expectation of all trajectories. Hence, we consider a more restrictive objective that requires all the sampled trajectories to satisfy the constraints:

$$\arg\max_\pi \mathbb{E}_{p_\mathcal{R}, p_\mathcal{T}, \pi}\left[\sum_{t=0}^{h} \gamma^t r_t\right] \text{ s.t. } \sum_{t=0}^{h} \gamma^t c_{i,t} \leq \epsilon_i \ \forall i \in [0, I] \quad (2)$$

where $c_{i,t} \sim p_{\mathcal{C}_i}(c|s_t, a_t)$ and $(s_t, a_t) \sim p_\mathcal{T}(s_t|s_{t-1}, a_{t-1})\pi(a_t|s_t)$. This formulation effectively describes hard constraints for each trajectory that may be difficult to satisfy in stochastic environments. Nevertheless, hard constraints are natural and often desired by practitioners. Hence, we will report the constraint violation rate (i.e., the probability with which a policy violates a constraint in a trajectory) in the experiments (see Section 3 and Section 5). Ultimately, we can think of this setup as a multi-objective problem where agents seek to maximize expected rewards while minimizing the constraint violation rate. We note that if the transition dynamics, policy and constraint functions are deterministic, the objectives in Equations 1 and 2 are equivalent to each other.

2.2 Constraint Inverse Reinforcement Learning

CRL assumes the constraint signals are observable from the environment, but in real-world problems, such constraint signals $c_{i,t}$ are not readily available. Instead, we have access to expert demonstrations $\mathcal{D}_e$ that follow the underlying constraints. To solve these problems, the agent must recover the constraint models from $\mathcal{D}_e$. This is a challenging task since there might be various equivalent combinations of reward distributions and constraints that can explain the same expert demonstrations [3, 12]. To simplify the problem, CIRL algorithms generally assume that rewards are observable and the goal is to recover only the constraints.

Connections to Inverse RL (IRL). CIRL and IRL are both problems of learning from demonstrations, and thus they have many connections. For example, some previous CIRL works [2, 3] followed the Maximum Entropy IRL framework for constraint inference, and the learned constraints can be turned into penalties added to rewards to solve the control problem (e.g., our baseline GACL in Section 4.2). Besides, IRL algorithms often assume that the expert agents are optimal or near-optimal in terms of maximizing the rewards. CIRL further assumes that the agents must satisfy underlying constraints. How to relax these assumptions will be an important direction of future work (see Section 6).

3 Benchmark

The goal of CIRL is to discover constraints from human demonstrations. However, the underlying constraints in real dataset are often unknown, which makes it difficult to quantify the algorithm's performance. To solve these issues, we develop a CIRL benchmark that enables the incorporation of external constraints to the environments. This benchmark facilitates the development of mature CIRL algorithms by examining whether they can accurately recover the added constraints and measuring their performance with some common RL metrics. For ease of application, our benchmark is based on OpenAI [13]. In this section, we introduce the virtual and realistic environments as well as the datasets in our benchmark.

3.1 Virtual Environment

An important application of RL is robotic control, and our virtual benchmark mainly studies the robot control task with a location constraint. In practice, this type of constraint captures the locations of obstacles in the environment. For example, the agent observes that none of the expert demonstrations visited some places in the environment. Then it is reasonable to infer that these locations must be unsafe. These unsafe locations can be represented by constraints. Although the real-world task might require more complicated constraints, our benchmark, as the first benchmark for constraint inverse reinforcement learning, could serve as a stepping stone for these tasks.

Environment Settings. We implement our virtual environments by utilizing MuJoCo [9], a virtual simulator suited to robotic control tasks. To extend MuJoCo for constraint inference, we modify the MuJoCo environments by incorporating some predefined constraints into each environment and adjusting some reward terms. Table 1 summarizes the environment settings (see Appendix A.1 for more details). The virtual environments have 5 different robotic control environments simulated by MuJoCo. We add constraints on the X-coordinate of these robots: 1) For the environments where it is relatively easier for the robot to move backward rather than forward (e.g., Half-Cheetah, Ant and Walker), our constraints bound the robot in the forward direction (with a large X-coordinate), 2) For the environments were moving forward is easier (e.g., Swimmer), the constraint bounds the robot in the backward direction (with a small X-coordinate). In these environments, the rewards are determined by the distance that a robot moves between the current and the previous time step, so the robot is likely to violate the constraints in order to maximize the magnitude of total rewards (for which we show below). To increase difficulty, we include an additional Biased Pendulum environment that has a larger reward on the left side. We nevertheless enforce a constraint to prevent the agent to go too far on the left side. The agent must resist the influence of high rewards and stay in the safe region.

Table 1: The virtual and realistic environments in our benchmark.

Type	Name	Dynamics	Obs. Dim.	Act. Dim.	Constraints
Virtual	Blocked Half-cheetah	Deterministic	18	6	X-Coordinate $\geq$ -3
	Blocked Ant	Deterministic	113	8	X-Coordinate $\geq$ -3
	Biased Pendulumn	Deterministic	4	1	X-Coordinate $\geq$ -0.015
	Blocked Walker	Deterministic	18	6	X-Coordinate $\geq$ -3
	Blocked Swimmer	Deterministic	10	2	X-Coordinate $\leq$ 0.5
Realistic	HighD Velocity Constraint	Stochastic	76	2	Car Velocity $\leq$ 40 m/s
	HighD Distance Constraint	Stochastic	76	2	Car Distance $\geq$ 20 m

3.2 Realistic Environment

Our realistic environment defines a highway driving task. This HighD environment examines if the agent can drive safely the ego car to the destination by following the constraints learned from human drivers' trajectories (see Figure 2). In practice, many of these constraints are based on driving context and human experience. For example, human drivers tend to keep larger distances from trucks and drive slower on crowded roads. Adding these constraints to an auto-driving system can facilitate a more natural policy that resembles human preferences.

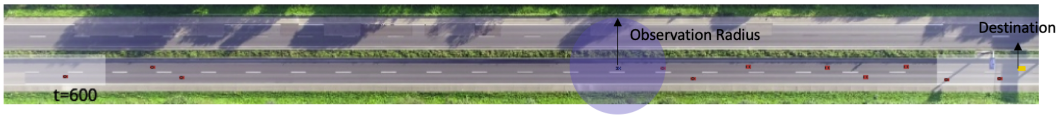

Figure 2: The Highway Driving (HighD) environment. The ego car is in blue, other cars are in red. The ego car can only observe the things within the region around (marked by blue). The goal is to drive the ego car to the destination (in yellow) without going off-road, colliding with other cars, or violating time limits and other constraints (e.g., velocity and distance to other vehicles).

Environment Settings. This environment is constructed by utilizing the HighD dataset [10]. Within each recording, HighD contains information about the static background (e.g., the shape and the length of highways), the vehicles, and their trajectories. We break these recordings into 3,041 scenarios so that each scenario contains less than 1,000 time steps. To create the RL environment, we randomly select a scenario and an ego car for control in this scenario. The game context, which is constructed by following the background and the trajectories of other vehicles, reflects the driving environment in real life. To further imitate what autonomous vehicles can observe on the open road, we ensure the observed features in our environment are commonly used for autonomous driving (e.g., velocity, distances to nearby vehicles). These features reflect only partial information about the game context (Appendix A.4 shows the complete list of input features). To collect these features, we utilize the feature collector from Commonroad RL [14]. Note that the HighD environment is stochastic since 1) we randomly select a scenario for the environment initialization, and 2) the trajectories generated by human drivers are stochastic since depending on the road conditions, people's preferences, and driving skills, these drivers might behave differently under the same context. In this HighD environment, we mainly study a car velocity constraint and a car distance constraint (see Table 1) to ensure the ego car can drive at a safe speed and keep a proper distance from other vehicles.

3.3 Demonstration Dataset

In this work, we generate the expert demonstrations dataset for constraint inference since 1) the virtual environments do not include demonstrations and 2) utilizing the trajectories in the HighD dataset under the realistic environments is problematic (because the underlying constraints in human demonstrations are unknown, which makes it difficult to determine whether the ground-truth constraints are broken). To generate the dataset, we train a PPO-Lagrange (PPO-Lag) under the CMDP with the known constraints and rewards (Table 1 and Appendix A.1) by performing the following steps:

Training Expert Agent with PPO-Lag. We train expert agents based on the ground-truth constraints in both the virtual environments and the realistic environments. The expert agent is trained by maximizing the following PPO-Lag objective based on the Reward Constrained Policy Optimization (RCPO) method [11]:

$$\min_{\lambda}\max_{\pi}\mathbb{E}_{(s,a)\sim\pi}\left(\sum_{t=0}^{h}\gamma^{t}r(s_t,a_t)\right)+\mathcal{H}(\pi)-\lambda\left[\mathbb{E}_{(s,a)\sim\pi}\left(\sum_{t=0}^{h}\gamma^{t}c^{*}(s,a)\right)-\epsilon\right] \tag{3}$$

where $\mathcal{H}$ denotes the entropy term and c^* is the ground-truth constraint function from the environment. Note that PPO-Lag assumes the reward and the constraint functions are deterministic and the agent can learn to avoid most of the constrained state-action pairs with the Lagrangian penalty term. The empirical studies (Sections 3.1 and 3.2) show that PPO-Lag can achieve a satisfactory performance (although not optimal) given the ground-truth constraint function.

Generating a Dataset with Expert Agents. We initialize $\mathcal{D}_e = \{\emptyset\}$ and run the trained expert agents in both the virtual and the realistic environments. While running, we monitor whether the ground-truth constraints are violated until the game ends. If not, we record the corresponding trajectory: $\mathcal{D}_e = \mathcal{D}_e \cup \{\tau_e\}$, otherwise, we abandon this trajectory. We repeat this process until the demonstration dataset has enough trajectories. Note that PPO-Lag is **not** optimal in terms of maximizing the rewards or satisfying constraints. After filtering the generated trajectories with state-action pairs that violate constraints, the recorded data $\mathcal{D}_e$ must satisfy the added constraints. However, there is no guarantee that the maximum number of rewards is collected in $\mathcal{D}_e$. Section 6 discuss this issue. We observe some algorithms can outperform PPO-Lag in the experiment (Section 5)

3.4 Empirical Study about Constraints.

In this section, we demonstration the validity of the constraints defined in table 1.

Constraints in the Virtual Environments. The thresholds of the constraints are determined experimentally to ensure that these constraints "matter" for solving the control problems. This is shown in Figure 3: 1) without knowing the constraints, a PPO agent tends to violate these constraints in order to collect more rewards within a limited number of time steps. 2) When we inform the agent of the ground-truth constraints (with the Lagrange method in Section 3.3), the PPO-Lagrange (PPO-Lag) agent learns how to stay in the safe region, but the scale of cumulative rewards is likely to be compromised. Based on these observations, we can evaluate whether the CIRL algorithms have learned a satisfying constraint function by checking whether the RL agent *trained with this constraint function (or set)* can gather more rewards and perform feasible actions in the safe states of the environment (i.e., by checking the *cumulative rewards* and the *constraint violation rate*).

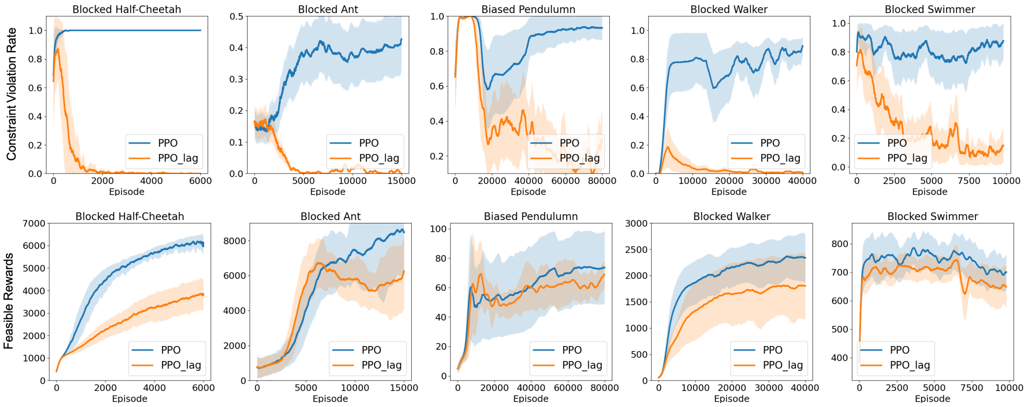

Figure 3: The constraint violation rate (top) and rewards (bottom). Environments from left to right: Blocked Half-cheetah, Blocked Ant, Biased Pendulum, Blocked Walker, and Blocked Swimmer.

Constraints in the Realistic Environments. We determine the thresholds of the velocity and distance constraints by showing the difference of performance between a PPO-Lag agent (Section 3.3) that *knows* the ground-truth constraints and a PPO agent *without knowing* the constraints. Figure 4 reports the violation rate of the speed constraint (top left) and the distance constraint (top right). The bottom graphs report the cumulative rewards in both settings. We find 1) the PPO agent tends to violate the constraints in order to get more rewards and 2) the PPO-Lag agent abandons some of these rewards in order to satisfy the constraints. Their gap demonstrates the significance of these constraints. Appendix A.6 shows the performance of other constraint thresholds.

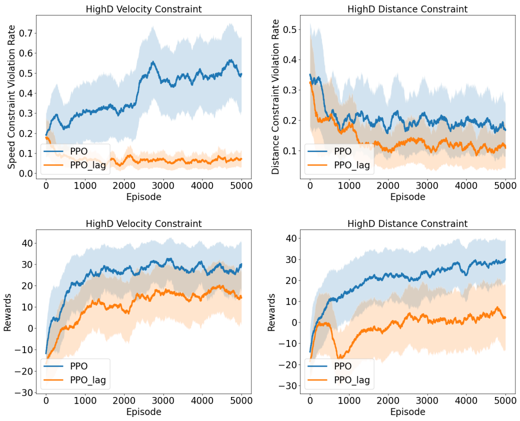

Figure 4: Model performance in the HighD environment with the velocity constraint (*left*) and the distance constraint (*right*) during *training*.

4 Baselines for Constraint Inverse Reinforcement Learning

In this section, we introduce our VCIRL algorithm and other important baselines for CIRL.

4.1 Variational Constraint Inverse Reinforcement Learning

To learn an accurate constraint function from the stochastic policies[1] and environment dynamics, we propose VCIRL that models the induced stochasticity in the generated trajectories. The goal of VCIRL is to infer the distribution of a feasibility variable Φ so that $p(\phi|s,a)$ measures to what extent an action a should be allowed in a particular state s[2]. The instance ϕ can define a soft constraint given by: $\hat{c}_\phi(s,a) = 1-\phi$ where $\phi \sim p(\cdot|s,a)$. Given a CMDP $\mathcal{M}^{\hat{c}_\phi}$ based on the estimated constraint function $\hat{c}_\phi$, we use $p(\mathcal{D}_e|\phi)$ to define the likelihood of generating the demonstration dataset $\mathcal{D}_e$. Under this setting, the true posterior $p(\phi|\mathcal{D}_e)$ is intractable due to high-dimensional input space, so VCIRL learns an approximate posterior $q(\phi|\mathcal{D}_e)$ by minimizing $\mathcal{D}_{kl}\Big[q(\phi|\mathcal{D}_e)\|p(\phi|\mathcal{D}_e)\Big]$. This is equivalent to maximizing an Evidence Lower Bound (ELBo) objective:

$$\mathbb{E}_q\Big[\log p(\mathcal{D}_e|\phi)\Big] - \mathcal{D}_{kl}\Big[q(\phi|\mathcal{D}_e)\|p(\phi)\Big] \tag{4}$$

We introduce our method by defining the key components in this objective.

The Log-Likelihood Term. We define the log-likelihood $\log\left[p(\mathcal{D}_e|\phi)\right]$ by utilizing the Maximum Entropy IRL method on a constrained MDP $\mathcal{M}^{\hat{c}_\phi}$ [3, 12]:

$$\log\left[p(\mathcal{D}_e|\phi)\right] = \log\left[\frac{1}{(Z_{\mathcal{M}^{\hat{c}_\phi}})^N}\prod_{i=1}^{N}\exp\left[r(\tau^{(i)})\right]\mathbb{1}^{\mathcal{M}^{\hat{c}_\phi}}(\tau^{(i)})\right] \tag{5}$$

where 1) N denotes the number of trajectories in the demonstration dataset $\mathcal{D}_e$, 2) the normalizing term $Z_{\mathcal{M}^{\hat{c}_\phi}} = \int \exp\left[r(\tau)\right]\mathbb{1}^{\mathcal{M}^{\hat{c}_\phi}}(\tau)\mathrm{d}\tau$, and 3) the trajectory identifier $\mathbb{1}^{\mathcal{M}^{\hat{c}_\phi}}(\tau^{(i)})$ can be defined by $\phi(\tau^{(i)}) = \prod_{t=1}^{T}\phi_t$ and $\phi_t \sim p(\phi|s_t^i, a_t^i)$, which defines to what extent the trajectory $\tau^{(i)}$ is feasible. Since Φ is a continuous variable with range $[0,1]$, we parameterize $p(\phi|s,a)$ by a Beta distribution:

$$\phi(s,a) \sim p(\phi|s,a) = \text{Beta}(\alpha,\beta) \text{ where } [\alpha,\beta] = \log[1+\exp(f(s,a))] \tag{6}$$

where f is implemented by a multi-layer network with 2-dimensional outputs (for α and β).

For simplicity, we slightly overload the symbols by using $\prod_{t=1}^{T}\phi(s_t^i, a_t^i)$ to denote the trajectory identifier $\mathbb{1}^{\mathcal{M}^c}(\tau^{(i)})$ and substitute it in Equation (5). In this way, we can define:

---

[1] PPO may models the policy distribution with a Gaussian, for which we follow a stable implementation in *temporarily hidden due to the anonymous policy* [15]

[2] We use a uppercase letter and a lowercase letter to define a random variable and an instance of this variable.

$$\log\left[p(\mathcal{D}_e|\phi)\right] = \sum_{i=1}^{N}\left[r(\tau^{(i)}) + \log\prod_{t=0}^{T}\phi(s_t^{(i)}, a_t^{(i)})\right] - N\log\int \exp[r(\hat{\tau})]\prod_{t=0}^{T}\phi(\hat{s}_t, \hat{a}_t)\mathrm{d}\hat{\tau} \tag{7}$$

The gradient of this likelihood function is given by:

$$\nabla_\phi \log\left[p(\mathcal{D}_e|\phi)\right] = \sum_{i=1}^{N}\left[\nabla_\phi\sum_{t=0}^{T}\log[\phi(s_t^{(i)}, a_t^{(i)})]\right] - N\mathbb{E}_{\hat{\tau}\sim\pi_{\mathcal{M}^{\phi}}}\left[\nabla_\phi\sum_{t=0}^{T}\log[\phi(\hat{s}_t, \hat{a}_t)]\right] \tag{8}$$

So maximizing $\log\left[p(\mathcal{D}_e|\phi)\right]$ is equivalent to maximizing the objective:

$$\mathcal{L}(\mathcal{D}) = \sum_{i=1}^{N}\sum_{t=0}^{T}\log[\phi(s_t^{(i)}, a_t^{(i)})] - N\mathbb{E}_{\hat{\tau}\sim\pi_{\mathcal{M}^{\phi}}}\left[\sum_{t=0}^{T}\log[\phi(\hat{s}_t, \hat{a}_t)]\right] \tag{9}$$

where $\hat{\tau}$ is sampled based on executing policy $\pi_{\mathcal{M}^{\hat{\phi}}}(\hat{\tau}) = \frac{\exp[r(\tau)]\phi(\hat{\tau})}{\int\exp[r(\tau)]\phi(\tau)\mathrm{d}\tau}$. This is a maximum entropy policy that can maximize cumulative rewards subject to $\pi_{\mathcal{M}^{\phi}}(\tau) = 0, \forall\sum_{(s,a)\in\tau}\hat{c}_\phi(s,a) > \epsilon$ (note that $\hat{c}_\phi(s,a) = 1 - \phi_t$ as defined above). We follow [11] and learn this policy with a RCPO objective. This is equivalent to substituting c^* with the estimated $\hat{c}_\phi$ in Equation (3).

The KL Divergence. Striving for simplicity and the ease of computing mini-batch gradients, we approximate $\mathcal{D}_{kl}\Big[q(\phi|\mathcal{D})\|p(\phi)\Big]$ with $\sum_{(s,a)\in\mathcal{D}}\mathcal{D}_{kl}\Big[q(\phi|s,a)\|p(\phi)\Big]$. Since both the posterior and the prior are Beta distributed, we define the KL divergence by following the Dirichlet VAE [16]:

$$\begin{aligned}\mathcal{D}_{kl}\Big[q(\phi|s,a)\|p(\phi)\Big] &= \log\left(\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha^0+\beta^0)}\right) + \log\left(\frac{\Gamma(\alpha^0)\Gamma(\beta^0)}{\Gamma(\alpha)\Gamma(\beta)}\right) \\ &\quad + (\alpha-\alpha^0)\Big[\psi(\alpha)-\psi(\alpha+\beta)\Big] + (\beta-\beta^0)\Big[\psi(\beta)-\psi(\alpha+\beta)\Big]\end{aligned} \tag{10}$$

where 1) $[\alpha^0, \beta^0]$ and $[\alpha, \beta]$ are parameters from the prior and the posterior functions. 2) Γ and ψ denote the gamma and the digamma functions.

4.2 Other Baselines for CIRL

Maximum Entropy Constraint Learning (MECL) is based on the Maximum Entropy IRL [12] framework, which proposes to maximize the entropy $H(\pi_E)$ during learning to solve the unidentifiability issues in classic apprenticeship learning. As one of the most generalizable frameworks, the maximum entropy framework acts as the foundation of many recent IRL methods. MECL extends the classic maximum entropy framework to constraint inference: instead of learning reward functions, MECL infers constraint functions or sets. [2] proposed an algorithm to search for constraints that can be added to the MDP in order to most increase the likelihood of observing expert demonstrations. This algorithm focused on only discrete state spaces. A following work [3] expanded MECL to continuous states and actions, which is consistent with the setting of our benchmark, and thus we use their implementation.

Generative Adversarial Constraint Learning (GACL) extends MECL by following the design of Generative Adversarial Imitation Learning (GAIL) [17]. Since the goal of CIRL is to infer constraints given the underlying rewards, we train the discriminator $\zeta(s,a)$ to assign 0s to violating state-action pairs and 1s to satisfying ones. In order to include the learned constraints into the policy update, we follow [3] and construct a new reward $r'(s,a) = r(\cdot) + \log\left[\zeta(\cdot)\right]$. This reward will punish the violating states or actions by assigning them $-\infty$ rewards, so the learned policy tends to satisfy the constraints. By comparing MECL and GACL, we show whether adversarial networks can accelerate training and improve imitation performance.

Binary Classifier Constraint Learning (BC2L) shares a similar policy update method with MECL, but instead of learning a stochastic constraint, it utilizes a deterministic binary classifier to differentiate expert trajectories from the generated ones, which often induces a loss of identifiability. BC2L allows us to study whether a binary classifier is sufficient for capturing the ground-truth constraints.

5 Empirical Evaluation

Experiment Setting. We study the performance of the aforementioned baselines in our benchmark. For a fair comparison, we use the same hyper-parameters to train baseline models. We repeat each experiment with different random seeds, according to which we report the mean ± standard deviation (std) results for each studied baselines and environment. For the details of model parameters and random seeds, please see Appendix A.2. Note that we explored the option of simplifying the realistic environments by inputting only the relevant features (i.e., the velocity of ego car and its distance to other vehicles in the HighD environments) into the constraint functions.

Figures 5 and 6 show the training curves in the virtual and realistic environments. Tables 2 and 3 show the testing performance. The highest success rate among all baselines in our HighD environments is around 68%, whereas the upper bound of success rate is 100%, which shows the room for improvement is sufficient. Appendix B.2 illustrates the causes of failures by showing the collision rate, time-out rate, and off-road rate. Compared to other baseline models, we find VCIRL generally outperforms other baselines in terms of getting lower constraint violation rates and collecting more rewards. This is because VCIRL can learn a more accurate and robust constraint model by considering the stochasticity inherent in the environment dynamics and the policy of PPO agents. More importantly, since VCIRL tries to find an imitation policy that satisfies the constraints in all its trajectories (instead of their expectation), VCIRL develops a more conservative imitation policy, especially when handling the constraints with large (epistemic and aleatoric) uncertainty. Although MECL, GACL, and BC2L achieve better performance in the Blocked Walker, Blocked Swimmer environments, and the simplified HighD environment with distance constraints respectively, none of these algorithms can perform consistently better than the others, and we find that GACL achieves only very limited performance in the HighD environment, which demonstrates that directly augmenting rewards with penalties induced by constraints can yield a sub-optimal control policy. Another important finding is that the performance of imitation policies in CIRL algorithms cannot match that of PPO-Lag (when the ground-truth constraints are known) in the Biased Pendulum and Blocked Walker environments. How to fill in this performance gap will be an important direction for improving CIRL algorithms.

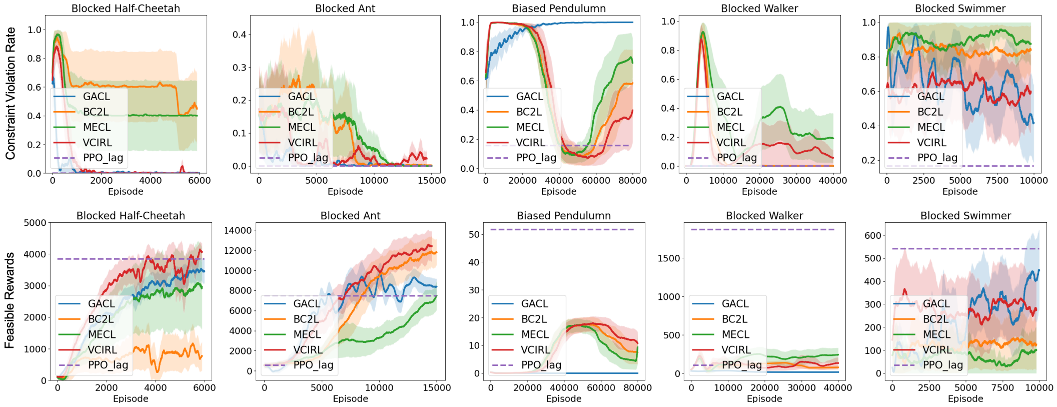

Figure 5: The constraint violation rate (top) and feasible rewards (i.e., the rewards from the trajectories without constraint violation, bottom) during training. From left to right, the environments are Blocked Half-cheetah, Blocked Ant, Bias Pendulum, Blocked Walker, and Blocked Swimmer.

Table 2: Testing performance in the virtual environments. We report the feasible rewards (i.e., the rewards from the trajectories without constraint violation) computed with 50 runs.

	Half-cheetah	Blocked Ant	Biased Pendulumn	Blocked Walker	Blocked Swimmer
GACL	3477.53 ± 416.54	7213.62 ± 993.12	0.85 ± 0.02	28.35 ± 0.77	**578.27** ± 148.16
BC2L	870.09 ± 499.03	11956.26 ± 1980.88	5.73 ± 5.60	48.73 ± 4.18	141.82 ± 152.14
MECL	3024.88 ± 1364.59	8546.19 ± 1262.03	1.02 ± 1.63	**126.76** ± 52.21	63.66 ± 107.95
VCIRL	**3805.72** ± 511.66	**13670.32** ± 2511.89	**6.64** ± 4.45	93.40 ± 93.97	191.11 ± 154.57

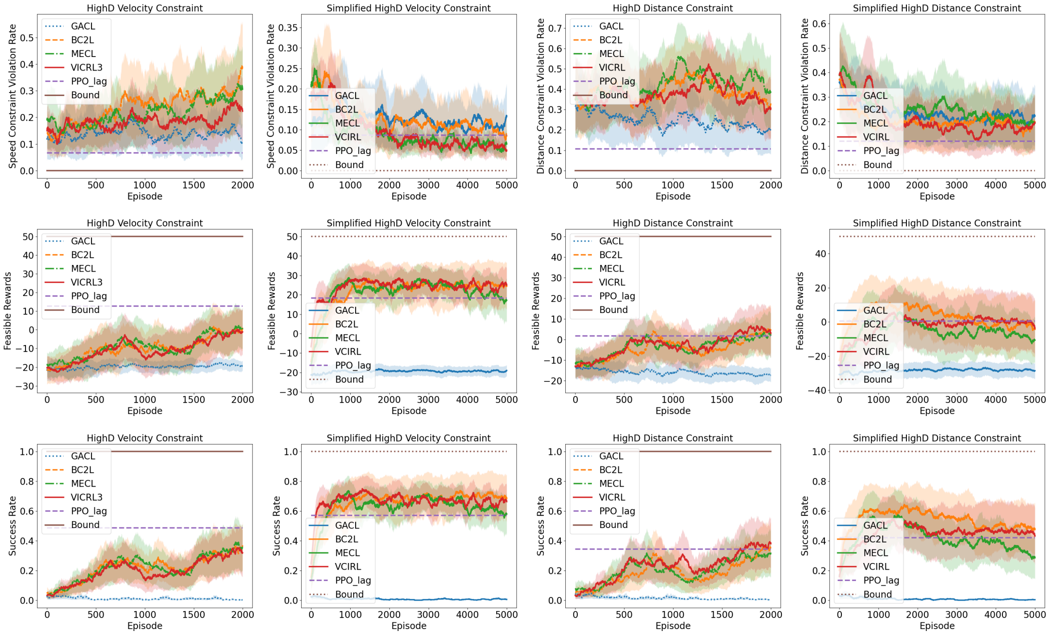

Figure 6: The constraint violation rate (top), feasible rewards (i.e., the rewards from the trajectories without constraint violation, middle) and success rate (bottom) during training. From left to right, the environments are HighD with the velocity, simplified velocity, distance and simplified distance constraints.

Table 3: Testing performance in the realistic environments. We report the feasible rewards (i.e., the rewards from the trajectories without constraint violation) computed with 30 runs.

	HighD Velo.	Simplified HighD Velo.	HighD Dis.	Simplified HighD Dis. dim.6
GACL	-17.33 ± 3.40	-18.06 ± 3.40	-5.33 ± 13.27	-18.0 ± 5.83
BC2L	-0.33 ± 15.73	2.67 ± 19.70	21.75 ± 7.02	**14.0** ± **19.48**
MECL	7.0 ± 18.31	6.5 ± 19.38	8.0 ± 16.55	0.67± 17.56
VCIRL	**10.0** ± **18.03**	**21.75** ± **15.89**	**26.33** ± **15.13**	8.33 ± 16.72

6 Limitations, Challenges and Open Questions

We introduce limitations and challenges in CIRL, as well as open questions for future work.

Constraint Violation. The imitation policies of CIRL agents are updated with RCPO [11], but Lagrange relaxation methods are sensitive to the initialization of the Lagrange multipliers and the learning rate. There is no guarantee that the imitation policies can consistently satisfy the given constraints [1]. As a result, even when a learned constraint function matches the ground-truth constraint, the learned policy may not match the expert policy, causing significant variation in training and sub-optimal model convergence. If we replace the Lagrange relaxation with Constrained Policy Optimization (CPO) [18, 19, 20, 21], CIRL may not finish training within a reasonable amount of time since CPO is computationally more expensive. How to design an efficient policy learning method that matches CIRL's iterative updating paradigm will be an important future direction.

Unrealistic Assumptions about Expert Demonstrations. CIRL algorithms typically assume that the expert demonstrations are optimal in terms of satisfying the constraints and maximizing rewards. There is no guarantee that these assumptions hold in practice since many expert agents (e.g., humans) do not always strive for optimality and constraint satisfaction. Previous works [22, 23, 24, 25, 26, 27], introduced IRL approaches to learn rewards from sub-optimal demonstrations, but how to extend these methods to constraint inference is unclear. A promising direction is to model *stochastic* constraints that assume that expert agents only follow the constraints with a certain probability.

Insufficient Constraint Diversity. CIRL can potentially recover complex constraints, but our benchmark mainly considers linear constraints as the ground-truth constraints (although this information is hidden from the agent). Despite this simplification, our benchmark is still very challenging: a CIRL agent must identify relevant features (e.g., velocity in x and y coordinates) among all input features (78 in total) and recover the exact constraint threshold (e.g., 40 m/s). For future work, we will explore nonlinear constraints and constraints on high-dimensional input spaces (e.g., pixels).

Online versus Offline CIRL. CIRL algorithms commonly learn an imitation policy by interacting with the environment. The online training nevertheless contradicts with the setting of many realistic applications where only the demonstration data instead of the environment is available. Given the recent progress in offline IRL [28, 29, 30, 31], extending CIRL to the offline training setting will be an important future direction.

7 Related Work

Inferring Constraints from Demonstrations. Previous works infer constraints to identify whether a state-action input is allowed. Among these works, [32, 2, 4, 33] assumed *discrete* state and action constraint sets of a limited size. They constructed discrete constraint sets to distinguish feasible state-action pairs from infeasible ones. Regarding *continuous* domains, the problem becomes one of inferring the boundaries between feasible and infeasible state-action pairs: [34, 35, 36] estimated a constraint matrix for observations and its null-space projection matrix. [37] learned geometric constraints by constructing a constraint knowledge base from demonstration. [38] proposed to construct constraint sets that correspond to the convex hull of all observed data. [3, 8] learned parametric non-convex constraint functions from demonstrations. Some previous works [39, 40, 41, 42, 43] focused on learning local trajectory-based constraints from a single trajectory. These works focus on inferring *deterministic* constraints while some recent works have learned *stochastic* constraints: [5] learned probabilistic constraints by assuming the environment constraint follows a logistic distribution. [44, 7] utilized a Bayesian approach to update a belief over constraints.

Testing Environments for CIRL. To the best of our knowledge, there is no common benchmark for CIRL, and thus previous works often define their own environment for evaluating their methods:

- *Grid-World.* Among the studied environments, the most common ones are grid-worlds, where previous works [2, 4, 7, 5, 8] added some obstacles in the grid map and examined whether the algorithms can find the locations and ranges of these obstacles. As simple and interpretable environments, grid-worlds enable a quick check about model performance, but generalizing performance to real applications with high-dimensional and continuous state spaces will be difficult.

- *Robotic Control.* Some previous works have tested their models on real robots, including robot arms [33, 45, 44, 38, 35, 36, 37], quadrotors [45, 44], and a humanoid robot hand [35]. However, there is no consistent type of robots for testing their algorithms, and the corresponding equipment is not commonly available. A recent work [3] used a *simulator* by adding some pre-defined constraints into the simulated environments. Our virtual environments use a similar setting, but we cover more control tasks and include a detailed study about the environments and the added constraints.

- *Safety-Gym.* Probably the most similar benchmark to our work is Safety-Gym [6]. However, Safety Gym is used to validate CRL algorithms that solve a *forward* policy-updating problem given the constraints [1], whereas our benchmark is designed for the *inverse* constraint-learning problem.

8 Conclusion

In this work, we introduced a benchmark, including robot control environments and highway driving environments, for evaluating CIRL algorithms. Each environment is aligned with a demonstration dataset generated by expert agents. To take into account the effect of stochastic environment dynamics on hard constraints, we proposed VCIRL to learn a distribution of constraints. We also performed an empirical evaluation of the performance of CIRL baselines on our benchmark. Finally, we discussed our limitations and important future directions.

A More Implementation and Environment Details

A.1 More Information about the Virtual Environments

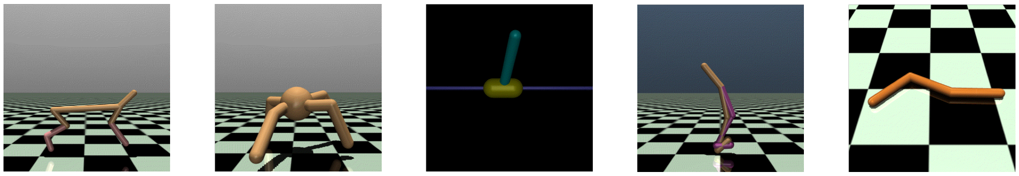

Figure A.1: Mujoco environments. From left to right, the environments are Half-cheetah, Ant, Inverted Pendulum, Walker and Swimmer.

Our virtual environments are based on Mujoco (see Figure A.1). We provide more details about the virtual environments as follows:

- *Blocked Half-Cheetah.* The agent controls a robot with two legs. The reward is determined by the distance it walks between the current and the previous time step and a penalty over the magnitude of the input action. The game ends when a maximum time step (1000) is reached. We define a constraint that blocks the region with X-coordinate ≤ -3, so the robot is only allowed to move in the region with X-coordinate between -3 and ∞.

- *Blocked Ant.* The agent controls a robot with four legs. The rewards are determined by the distance to the origin and a healthy bonus that encourages the robot to stay balanced. The game ends when a maximum time step (500) is reached. Similar to the Blocked Half-Cheetah environment, we define a constraint that blocks the region with X-coordinate ≤ -3, so the robot is only allowed to move in the region with X-coordinate between -3 and ∞.

- *Biased Pendulum.* Similar to the Gym CartPole [46], the agent's goal is to balance a pole on a cart. The game ends when the pole falls or a maximum time step (100) is reached. At each step, the environment provides a reward of 0.1 if the X-coordinate ≥ 0 and a reward of 1 if the X-coordinate ≤ -0.01. The reward monotonically increases from 0.1 to 1 when $-0.01 <$ X-coordinate < 0. We define a constraint that blocks the region with X-coordinate ≤ -0.015, so the reward incentivizes the cart to move left, but the constraint prevents it from moving too far. If the agent can detect the ground-truth constraint threshold, it will drive the cart to move into the region with X-coordinate between -0.015 and -0.01 and stay balanced there.

- *Blocked Walker.* The agent controls a robot with two legs and learns how to make the robot walk. The reward is determined by the distance it walks between the current and the previous time step and a penalty over the magnitude of the input action (this is following the original Walker2d environment). The game ends when the robot loses its balance or reaches a maximum time step (500). Similar to the Blocked Half-Cheetah and Blocked Ant environment, we constrain the region with X-coordinate ≤ -3, so the robot is only allowed to move in the region with X-coordinate between -3 and ∞.

- *Blocked Swimmer.* The agent controls a robot with two rotors (connecting three segments) and learns how to move. The reward is determined by the distance it walks between the current and the previous time step and a penalty over the magnitude of the input action. The game ends when the robot reaches a maximum time step (500). Unlike the Blocked Half-Cheetah and Blocked Ant environment, it is easier for the Swimmer robot to move ahead than move back, and thus we constrain the region with X-coordinate ≥ 0.5, so the robot is only allowed to move in the region with X-coordinate between $-\infty$ and 0.5.

A.2 Hyper-Parameters

We published our benchmarks, including the configurations of the environments and the models, in *temporarily hidden due to the anonymous policy*. Please see the README.MD file for more details. We provide a brief summary of the hyper-parameters.

In order to develop a fair comparison among CIRL algorithms, we use the same setting for all algorithms.

In the **virtual environments**, we set 1) the batch size of PPO-Lag to 64, 2) the size of the hidden layer to 64, and 3) the number of hidden layers for the policy function, the value function, and the cost function to 3. We decide the other parameters, including the learning rate of both PPO-Lag and constraint model, by following some previous work [3] and their implementation [3]. The random seeds of virtual environments are 123, 321, 456, 654, and 666.

In the **realistic environments**, we set 1) the batch size of the constraint model to 1000, 2) the size of the hidden layer to 64 and 3) the number of hidden layers for the policy function, the value function and the cost function to 3. We decide the other parameters, including the learning rate of both PPO-Lag and constraint model, by following CommonRoad RL [14] and their implementation [4]. During our experiment, we received plenty of help from their forum [5]. We will acknowledge their help in the formal version of this paper. The random seeds of realistic environments are 123, 321, and 666.

A.3 Experimental Equipment and Infrastructures

We run the experiment on a cluster operated by the Slurm workload manager. The cluster has multiple kinds of GPUs, including Tesla T4 with 16 GB memory, Tesla P100 with 12 GB memory, and RTX 6000 with 24 GB memory. We used machines with 12 GB of memory for training the CIRL models. The number of running nodes is 1, and the number of CPUs requested per task is 16. Given the aforementioned resources, running one seed in the virtual environments and the realistic environments takes 2-4 hours and 10-12 hours respectively.

A.4 Dataset

We provide a brief summary of the dataset. For more details, please see the *dataset supplementary material*.

A.4.1 Size

We generate the dataset by running the expert agent in the environments (see Section 3.3). The dataset for each virtual environment contains 50 trajectories while the dataset for each realistic environment contains 100 trajectories. We have a total of 5 virtual environments and 4 realistic environments, for which we have collected a total of 650 trajectories. The dataset does not contain any personally identifiable information or other related content.

A.5 Computational Complexity

We provide a brief analysis of the computational complexity. The CIRL algorithms, including GACL, MECL, BC2L, and VCIRL, use an iterative updating paradigm and thus their computational complexities are similar. Let K denote the number of iterations. Within each iteration, the algorithms update both the imitation policy and the constraint model. Let M denote the number of episodes that the PPO-Lag algorithm runs in the environments. Let N denote the number of sampling and expert trajectories. Let L denote the maximum length of each trajectory. During training, we use

---

[3] *temporarily hidden due to the anonymous policy*

[4] *temporarily hidden due to the anonymous policy*

[5] *temporarily hidden due to the anonymous policy*

mini-batch gradient descent. Let B denote the batch size, and then the computational complexity is $O(KL(M+N)/B)$.

A.6 Exploring Other Constraints in the Realistic Environments

The constraint thresholds in our environments are determined empirically according to the performance (constraint violation rate and rewards) of the PPO agent and the PPO-Lag agent. To support this claim, we show the performance of other thresholds and analyze why they are sub-optimal in terms of validating CIRL algorithms.

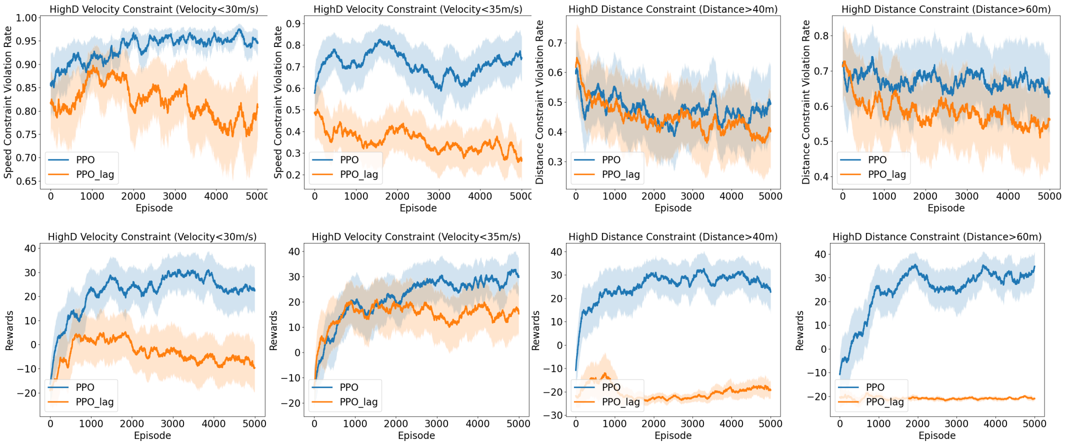

Figure A.2: From left to right, the constraint violation rate (top) and rewards (bottom) of the PPO and PPO-Lag agents in the HighD environments with constraints 1) Ego Car Velocity $<$ 30 m/s, 2) Ego Car Velocity $<$ 35 m/s, 3) Car Distance > 40 m, and 4) Car Distance > 60 m.

We have explored the option of using a 30m/s velocity constraint (The first column on the left in Figure A.2) and 35m/s velocity constraint (The second column on the left in Figure A.2). Ideally, these constraints should be closer to the realistic speed limit in most countries. However, the HighD dataset comes from German highways where there is no speed limit. Moreover, when building the environment, the ego car is accompanied by an initial speed calculated from the dataset. We observed that the initial speed is already higher than the speed limit (e.g., 35m/s) in many scenarios, and thus the violation rate will always be 1 in these scenarios, leaving no opportunity for improving the policy. This explains why the corresponding violation rates are high for the PPO and the PPO-Lag agents.

We also explored the option of using a 40m distance constraint (third column in Figure A.2) and a 60m distance constraint (fourth column in Figure A.2). Ideally, these constraints should be more consistent with the 2-second gap recommendation (the average speed is around 30m/s in HighD, so the recommended gap is 2*30m/s=60m), but we find the controlling performance of the PPO-Lag agents are very limited, which shows the agent cannot even develop a satisfying control policy when knowing the ground-truth constraints. This is because the ego car learns to frequently go off-road in order to maintain the large gap.

B More Experimental Results

B.1 The Effect of Demonstration Dataset

We study the influence on model performance by utilizing 1) noisy demonstration datasets that record some random actions during data generation. 2) smaller demonstration datasets that include only a subset of expert trajectories in the original data. Figure B.1 shows the experimental results. We find the model performance, especially the rewards collected by the agent, is significantly influenced after replacing some expert actions with random ones. This is because the random actions induce

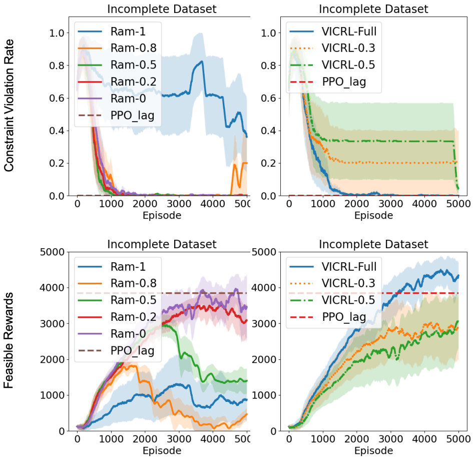

Figure B.1: Model performance of VCIRL in the Blocked Half-Cheetah environment. We use a noisy demonstration dataset with 100%, 80%, 50%, 20% and 0% random actions (*left*) and a dataset with only 30%, 50% and 100% (full) trajectories of expert demonstration (*right*) during *training*.

sub-optimal trajectories, which are less effective in terms of reflecting the preference of expert agents and thus leads to some inaccurate constraints. When it comes to the influence of reducing the size of demonstration data, we find the constraint violation rate becomes higher. This is because the expert agent trained by PPO-Lag applies a stochastic policy. The diversity of recorded actions and states is influenced if we reduce the roll-out numbers (i.e., number of trajectories).

B.2 Complementary Results in the Realistic Environment

Figure B.2 reports the average velocity, collision rate, off-road rate, time-out rate and goal-reaching rate during training. We find the off-road rate of GACL is significantly higher than other methods. It explains why GACL cannot achieve a satisfying performance. Another main limitation of current baselines is their incapability of preventing the collision events, especially under the car distance constraints.

C Societal Impact

Positive Societal Impacts The ability to discover what can be done and what cannot be done is an important function of modern AI systems, especially for systems that have frequent interactions with humans (e.g., house keeping robots and smart home systems). As an important stepping stone towards the design of effective systems, constraint models can help develop human-friendly AI systems and facilitate their deployments in real applications.

Negative Societal Impacts Possible real-world applications of constraint models include autonomous driving systems. Since constraint models are often represented by black-box deep models, there is no guarantee that the models are trustworthy and interpretable. When an autonomous vehicle is involved into an accident, it is difficult to identify the cause of this accident, which might cause a loss of confidence in autonomous systems while negatively impacting society.

References

[1] Yongshuai Liu, Avishai Halev, and Xin Liu. Policy learning with constraints in model-free reinforcement learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4508–4515. ijcai.org, 2021.

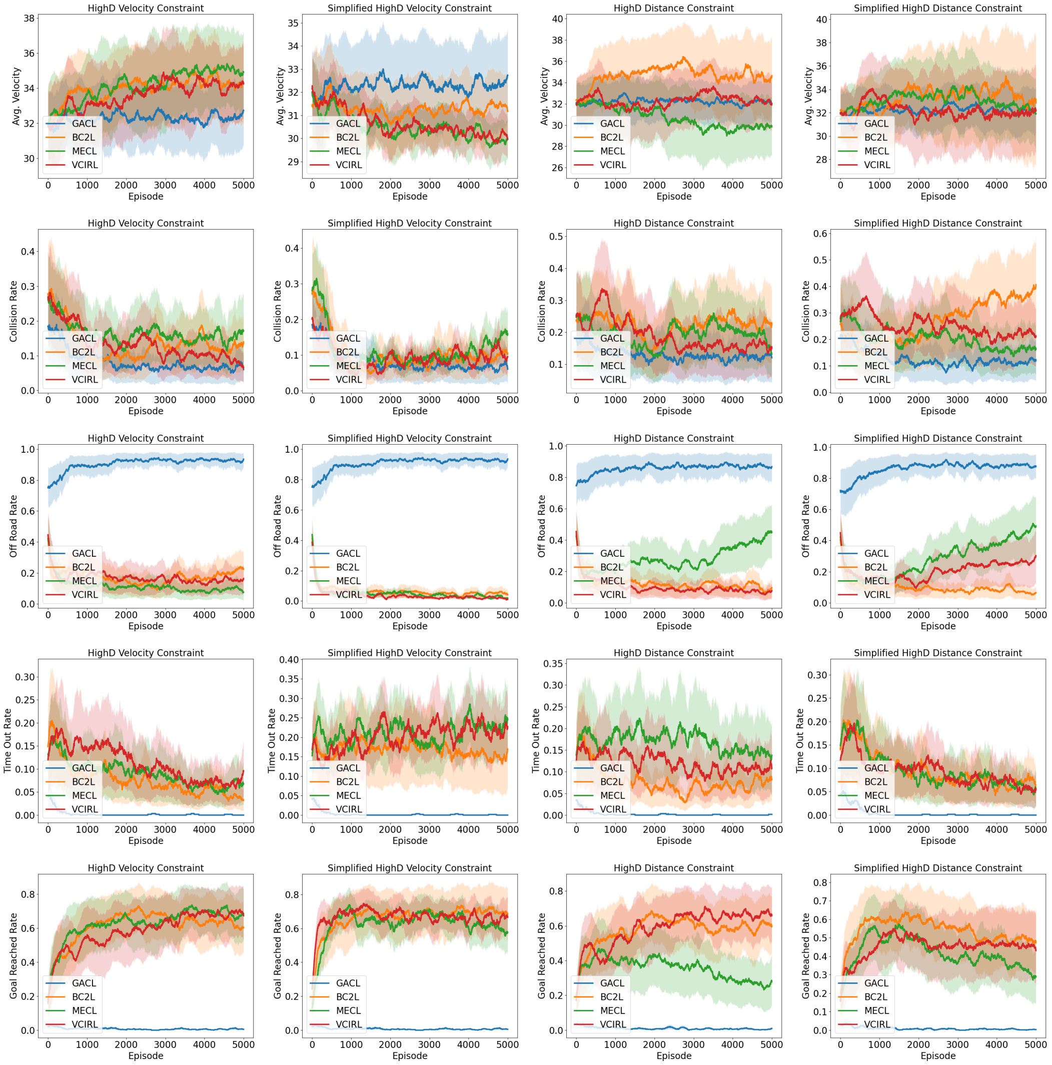

Figure B.2: The average velocity (first column), collision rate (second column), off road rate (third column), time out rate (fourth column) and goal reaching rate (last column) during training. From left to right, the environments are HighD with the velocity, simplified velocity, distance and simplified distance constraints.

[2] Dexter R. R. Scobee and S. Shankar Sastry. Maximum likelihood constraint inference for inverse reinforcement learning. In *8th International Conference on Learning Representations, (ICLR)*. OpenReview.net, 2020.

[3] Shehryar Malik, Usman Anwar, Alireza Aghasi, and Ali Ahmed. Inverse constrained reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139, pages 7390–7399, 2021.

[4] David Livingston McPherson, Kaylene C. Stocking, and S. Shankar Sastry. Maximum likelihood constraint inference from stochastic demonstrations. In *IEEE Conference on Control Technology and Applications, (CCTA)*, pages 1208–1213, 2021.

[5] Arie Glazier, Andrea Loreggia, Nicholas Mattei, Taher Rahgooy, Francesca Rossi, and Kristen Brent Venable. Making human-like trade-offs in constrained environments by learning from demonstrations. *CoRR*, abs/2109.11018, 2021.

[6] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7:1, 2019.

[7] Dimitris Papadimitriou, Usman Anwar, and Daniel S Brown. Bayesian inverse constrained reinforcement learning. In *Workshop on Safe and Robust Control of Uncertain Systems (NeurIPS)*, 2021.

[8] Ashish Gaurav, Kasra Rezaee, Guiliang Liu, and Pascal Poupart. Learning soft constraints from constrained expert demonstrations. *CoRR*, abs/2206.01311, 2022.

[9] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 5026–5033. IEEE, 2012.

[10] Robert Krajewski, Julian Bock, Laurent Kloeker, and Lutz Eckstein. The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2118–2125, 2018.

[11] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *International Conference on Learning Representations ICLR*. OpenReview.net, 2019.

[12] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, pages 1433–1438. AAAI Press, 2008.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[14] Xiao Wang, Hanna Krasowski, and Matthias Althoff. Commonroad-rl: A configurable reinforcement learning environment for motion planning of autonomous vehicles. In *24th IEEE International Intelligent Transportation Systems Conference, ITSC 2021, Indianapolis, IN, USA, September 19-22, 2021*, pages 466–472. IEEE, 2021.

[15] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[16] Weonyoung Joo, Wonsung Lee, Sungrae Park, and Il-Chul Moon. Dirichlet variational autoencoder. *Pattern Recognit.*, 107:107514, 2020.

[17] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Neural Information Processing Systems (Neurips)*, pages 4565–4573, 2016.

[18] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning (ICML)*, volume 70, pages 22–31. PMLR, 2017.

[19] Yinlam Chow, Ofir Nachum, Aleksandra Faust, Mohammad Ghavamzadeh, and Edgar A. Duéñez-Guzmán. Lyapunov-based safe policy optimization for continuous control. *CoRR*, abs/1901.10031, 2019.

[20] Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J. Ramadge. Projection-based constrained policy optimization. In *International Conference on Learning Representations (ICLR)*, 2020.

[21] Yongshuai Liu, Jiaxin Ding, and Xin Liu. IPO: interior-point policy optimization under constraints. In *AAAI Conference on Artificial Intelligence*, pages 4940–4947, 2020.

[22] Daniel S. Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning*, volume 97, pages 783–792, 2019.

[23] Daniel S. Brown, Wonjoon Goo, and Scott Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Conference on Robot Learning (CoRL)*, volume 100, pages 330–359, 2019.

[24] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In *International Conference on Machine Learning (ICML)*, volume 97, pages 6818–6827, 2019.

[25] Letian Chen, Rohan R. Paleja, and Matthew C. Gombolay. Learning from suboptimal demonstration via self-supervised reward regression. In *Conference on Robot Learning (CoRL)*, volume 155 of *Proceedings of Machine Learning Research*, pages 1262–1277, 2020.

[26] Voot Tangkaratt, Bo Han, Mohammad Emtiyaz Khan, and Masashi Sugiyama. Variational imitation learning with diverse-quality demonstrations. In *International Conference on Machine Learning (ICML)*, volume 119, pages 9407–9417, 2020.

[27] Voot Tangkaratt, Nontawat Charoenphakdee, and Masashi Sugiyama. Robust imitation learning from noisy demonstrations. In *Artificial Intelligence and Statistics (AISTATS)*, volume 130, pages 298–306, 2021.

[28] Vinamra Jain, Prashant Doshi, and Bikramjit Banerjee. Model-free IRL using maximum likelihood estimation. In *AAAI Conference on Artificial Intelligence*, pages 3951–3958, 2019.

[29] Donghun Lee, Srivatsan Srinivasan, and Finale Doshi-Velez. Truly batch apprenticeship learning with deep successor features. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5909–5915, 2019.

[30] Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. Imitation learning via off-policy distribution matching. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2020.

[31] Divyansh Garg, Shuvam Chakraborty, Chris Cundy, Jiaming Song, and Stefano Ermon. Iq-learn: Inverse soft-q learning for imitation. In *Neural Information Processing Systems (NeurIPS)*, pages 4028–4039, 2021.

[32] Glen Chou, Dmitry Berenson, and Necmiye Ozay. Learning constraints from demonstrations. In *Workshop on the Algorithmic Foundations of Robotics, WAFR 2018*, volume 14, pages 228–245. Springer, 2018.

[33] Daehyung Park, Michael Noseworthy, Rohan Paul, Subhro Roy, and Nicholas Roy. Inferring task goals and constraints using bayesian nonparametric inverse reinforcement learning. In *Conference on Robot Learning (CoRL)*, volume 100, pages 1005–1014, 2019.

[34] Hsiu-Chin Lin, Matthew Howard, and Sethu Vijayakumar. Learning null space projections. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2613–2619. IEEE, 2015.

[35] Hsiu-Chin Lin, Prabhakar Ray, and Matthew Howard. Learning task constraints in operational space formulation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 309–315. IEEE, 2017.

[36] Leopoldo Armesto, Jorren Bosga, Vladimir Ivan, and Sethu Vijayakumar. Efficient learning of constraints and generic null space policies. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1520–1526, 2017.

[37] Claudia Pérez-D'Arpino and Julie A. Shah. C-LEARN: learning geometric constraints from demonstrations for multi-step manipulation in shared autonomy. In *2017 IEEE International Conference on Robotics and Automation, ICRA*, pages 4058–4065. IEEE, 2017.

[38] Marcel Menner, Peter Worsnop, and Melanie N. Zeilinger. Constrained inverse optimal control with application to a human manipulation task. *IEEE Trans. Control. Syst. Technol.*, 29(2):826–834, 2021.

[39] Sylvain Calinon and Aude Billard. A probabilistic programming by demonstration framework handling constraints in joint space and task space. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 367–372. IEEE, 2008.

[40] Gu Ye and Ron Alterovitz. Demonstration-guided motion planning. In *Robotics Research - The 15th International Symposium ISRR*, volume 100, pages 291–307, 2011.

[41] Lucia Pais, Keisuke Umezawa, Yoshihiko Nakamura, and Aude Billard. Learning robot skills through motion segmentation and constraints extraction. In *HRI Workshop on Collaborative Manipulation*, page 5. Citeseer, 2013.

[42] Changshuo Li and Dmitry Berenson. Learning object orientation constraints and guiding constraints for narrow passages from one demonstration. In *International Symposium on Experimental Robotics (ISER)*, volume 1, pages 197–210, 2016.

[43] Negar Mehr, Roberto Horowitz, and Anca D. Dragan. Inferring and assisting with constraints in shared autonomy. In *IEEE Conference on Decision and Control (CDC)*, pages 6689–6696. IEEE, 2016.

[44] Glen Chou, Dmitry Berenson, and Necmiye Ozay. Uncertainty-aware constraint learning for adaptive safe motion planning from demonstrations. In *Conference on Robot Learning (CoRL)*, volume 155, pages 1612–1639, 2020.

[45] Glen Chou, Necmiye Ozay, and Dmitry Berenson. Learning parametric constraints in high dimensions from demonstrations. In *Conference on Robot Learning (CoRL)*, volume 100, pages 1211–1230, 2019.

[46] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Checklist