Techniques to learn constraints from demonstrations

by

Ashish Gaurav

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Prashant Doshi Professor of Computer Science, School of Computing University of Georgia, Athens
Supervisor(s):	Pascal Poupart Professor, Cheriton School of Computer Science University of Waterloo
Internal Member(s):	Jesse Hoey Professor, Cheriton School of Computer Science University of Waterloo
	Yaoliang Yu Associate Professor, Cheriton School of Computer Science University of Waterloo
Internal-External Member:	Yash Vardhan Pant Assistant Professor, Dept. of Electrical and Computer Engineering University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of contributions

This thesis is based on three works of research authored by myself, my supervisor Pascal Poupart, and other co-authors.

Chapters 4 and 5 consist of material from the first two works on inverse constraint learning and inverse probabilistic constraint learning. I conceived both these ideas, wrote the code to validate these ideas and ran the experiments. For the work on inverse constraint learning, my co-authors include Kasra Rezaee and Guiliang Liu, who provided invaluable feedback and discussion related to the submissions. Pascal was the co-author on both these works and helped me edit several sections.

Chapter 6 consists of material from my third work on feature selection in a high dimensional state action space in the context of constraint learning. The idea was jointly conceived by me and Pascal through the course of several discussions. I wrote the code and conducted experiments for this work.

The first work on inverse constraint learning has been published in ICLR 2023 as a spotlight paper [42]. The other two works are pending submission to suitable conferences.

Abstract

Given demonstrations from an optimal expert, inverse reinforcement learning aims to learn an underlying reward function. However, it is limiting to assume that the reward function fully explains the expert behaviour, since in many real world settings the expert might be acting to satisfy additional behavioural constraints. Recovering these additional constraints falls within the paradigm of constraint learning from demonstrations. Specifically, in this work, we focus on the setting of inverse constraint learning (ICL), where we wish to learn a single but arbitrarily complex constraint from demonstrations assuming the reward is known in advance.

For this setting, we first provide a framework to learn an expected constraint from constrained expert demonstrations. We then show how to translate an expected constraint into a probabilistic constraint and additionally extend the proposed framework to learn a probabilistic constraint from constrained expert demonstrations. Here, an expected constraint refers to a constraint that bounds the cumulative costs averaged over a batch of trajectories to be within a budget. Similarly, a probabilistic constraint upper bounds the probability that cumulative costs are above a certain threshold. Finally, we provide convergence guarantees for the proposed frameworks.

Following these approaches, we consider the complementary challenge of learning a constraint in a high dimensional state-action space. In such a setting, the constraint function may truly depend only on a subset of the input features. We propose using a simple test from the hypothesis testing literature to select this subset of features in order to construct a reduced input space for the constraint function. We also discuss the implications of using this approach in conjunction with an ICL algorithm.

To validate our proposed approaches, we conduct experiments with synthetic, robotics and environments based on real-world driving datasets. For feature selection, we test our approach by considering environments with varying state-action space sizes.

Acknowledgements

First, I would like to extend my deepest appreciation for my supervisor, Pascal, for his invaluable guidance and infinite patience throughout my Ph.D. journey. His mentorship has been instrumental in shaping my research.

Second, I am also exceedingly grateful to my wife Muskan who has been by my side through this journey. Her optimism and belief in my abilities were my pillars of support in this monumental journey.

Third, I am thankful to my family and Muskan's family who encouraged me to start and complete this unconventional journey of research. I am especially grateful to my sister Ankita with whom I shared countless hours of Ph.D. rants.

Finally, I offer my sincere thanks to my numerous friends whose collaboration and discussion made this Ph.D. journey enriching and fruitful.

Dedication

Dedicated to Muskan and Ankita.

Table of contents

List of Figures

List of Tables

List of Abbreviations

CDF Cumulative density function 29, 30, 106, 107

CIFE Conditional infomax feature extraction 104, 105

CMDP Constrained Markov decision process xvi, 15–19, 27, 117

CMIM Conditional mutual information maximization 104

CRL Constrained reinforcement learning 18, 20, 35, 46, 51, 58, 66, 86, 92, 143

DISR Double input symmetrical relevance 104

ICL Inverse constraint learning xiv, xvi, xviii–xxii, 5–8, 20, 21, 26, 27, 35–37, 42, 44, 45, 51, 52, 57–60, 62, 64, 66, 68, 71, 73–78, 83–88, 90, 92–94, 96, 98–103, 105, 108–118, 135–140, 143–146, 148, 149

ICRL Inverse constrained reinforcement learning 20, 26, 57

IPCL Inverse probabilistic constraint learning xiv, xviii, xxi, xxii, 8, 20, 21, 26, 73–77, 82–84, 86–88, 90, 92–94, 96, 110, 111, 117, 139, 143, 148, 149

IRL Inverse reinforcement learning xv, xvi, 2, 6, 19, 26, 45

KS2ST Kolmogorov Smirnov two sample test xix, xxi, xxii, 106, 109, 112–115, 117

MDP Markov decision process xvi, 10, 11, 13–16, 33, 99

MRMR Max relevance min redundancy 105

PCRL Probabilistic constrained reinforcement learning 19, 20, 143

PPO Proximal policy optimization xxii, 22, 51, 59, 83, 144–149

RL Reinforcement learning xv, 1, 2, 4, 10, 15, 18, 19, 31, 34, 35, 86

TRPO Trust region policy optimization 22

Chapter 1

Introduction

1.1 Reinforcement learning (RL)

Reinforcement learning (RL) [127] is a growing subfield within the broad field of machine learning (ML). Typically, RL algorithms aim to learn a policy or behaviour (i.e. how to act) given access to an environment and a reward signal, through the process of repeated interactions with the environment. The overall objective is to maximize some notion of long-term reward.

Figure 1.1: Autonomous driving example: here, the agent must learn how to drive to reach a certain destination. (Source: Forbes)

There are many examples of using RL agents to learn behaviours. In this work, we motivate

decision making through the simple example of an autonomous driving setup where the agent is the vehicle that must learn how to drive, given access to the driving environment and a reward signal that encodes some notion of progress towards the destination.

After each interaction with the environment, the agent (i.e. the vehicle) receives an instantaneous reward signal that indicates how good or bad the action was. For example, if the agent moves towards the final destination, it may receive a positive reward and if it moves away from the destination or stays stopped, it may receive a negative reward. Then, using the feedback from these reward signals, an RL algorithm improves upon a policy repeatedly, until it becomes optimal.

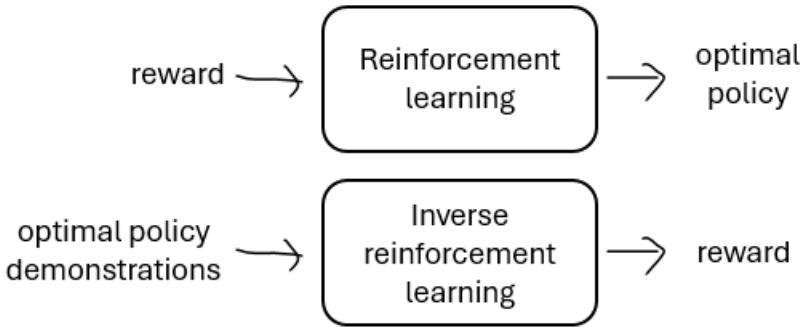


Figure 1.2: RL vs IRL.

Specifying a reward signal for the purpose of learning optimal behaviour is a manual and laborious process. Therefore it is common to collect some expert demonstrations and automate the process of specifying a reward by learning a reward from these collected expert demonstrations. This is the process of inverse reinforcement learning (IRL) [97, 115], which takes in demonstrations from an optimal policy (expert) and outputs a reward function that encodes the underlying motivation behind the agent's behaviour. Here, a reward function is an instantaneous function of the state and action, that encodes the instantaneous reward signal.

1.2 Constrained reinforcement learning (CRL)

In many cases, there exist behavioural constraints that the agent must satisfy, in addition to optimizing the reward function objective. For example, in the context of autonomous

driving, in addition to the standard objective of progress towards the destination which can be encoded through a reward function, there exist other objectives like safety, comfort, no traffic violations, etc. that are naturally specified through additional constraints.

Learning an optimal policy or behaviour (i.e., how to act) such that the long term reward signal is maximized while ensuring that the specified constraints are specified falls within the paradigm of constrained reinforcement learning.

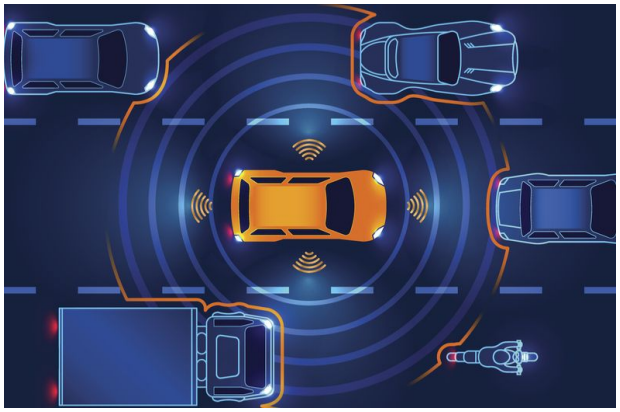


Figure 1.3: Autonomous driving constraint example: here, the agent must learn how to drive to reach a certain destination subject to a constraint on safety, i.e., the agent should avoid collisions with other vehicles by maintaining lateral and longitudinal gaps. (Source: rac.co.uk)

However, defining these constraints is *challenging in practice* since it is impractical to list out all the behavioural constraints that apply to a problem, especially when the constraints are time varying and context-dependent. For example, in an autonomous driving scenario, the constraints may not just depend on safety, comfort and traffic rule obedience, but also depend on contextual factors like time of day and weather conditions. Similarly, in a lane change scenario for autonomous driving, the agent may need to maintain a gap from other vehicles, which is a time varying constraint that depends on the instantaneous traffic density near the agent vehicle and is specified differently depending on the speed limit.

One solution to avoid manually specifying all the constraints that apply to a problem is to perform constraint learning from expert demonstrations. Similar to inverse RL that learns a reward function from expert demonstrations, inverse constrained RL (ICRL) aims to learn both the reward and constraint functions from given expert demonstrations. This means that given expert demonstrations, we learn a reward function as well as all the necessary

constraint functions that produce the expert behaviour. Here, a constraint function defines a corresponding constraint.

Our objective in this work is to provide techniques for learning constraints from expert demonstrations. There are two reasons we prefer to learn constraints over rewards:

- Specifying behaviour through constraints might be more convenient than specifying behaviour through a reward due to interpretability of constraints [25] and their established usefulness in safety-critical applications [26].
- Constraints can be conveniently reused in environments of the same domain, but with different dynamics, and in some cases, can be reused across different domains. To see this, consider a constraint that limits the agent's visit to certain unsafe states and only allows visiting safe states instead. This can be encapsulated by a reward function which is sufficiently negative for unsafe states, so as to discourage visiting such states. While a negative reward may be sufficient to prevent an unsafe state to be visited in some environment, that same negative reward may be insufficient in a different environment when the optimal policy visits other states (due to different dynamics) with positive rewards that may now compensate for this negative reward. These factors may lead to a completely different policy being learned in the target environment. In contrast, specifying this behaviour through a constraint is much simpler since there are no negative rewards to deal with. Due to this transferability problem, it is desirable to work with constraints rather than an equivalent reward.

1.3 Inverse constraint learning (ICL)

As explained in Section 1.2, we can use inverse RL algorithms to extract a reward function, or use inverse constrained RL algorithms to extract both a reward and constraint functions. However, these methods suffer from the problem of *unidentifiability*. In the context of inverse RL, this refers to the phenomenon that several different reward functions can yield the same optimal policy through an RL algorithm. Due to this, there is no unique reward function that the inverse RL process can output. Similarly, for inverse constrained RL, this ambiguity or unidentifiability is exacerbated since now there are several reward-constraint combinations that can be output by the ICRL process.

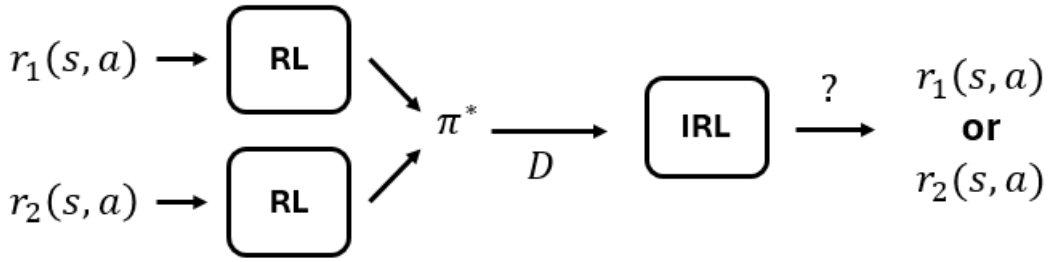


Figure 1.4: The problem of unidentifiability in inverse RL: different rewards may yield the same optimal policy, so there is an inherent ambiguity in the output of an inverse RL algorithm since there are multiple possible valid reward functions that can be output.

To mitigate this unidentifiability, we assume the reward function is known in advance, and only a single constraint needs to be learned[1]. Since we are learning a constraint parameterized by a neural network, this is not limiting as the learned constraint can be arbitrarily complex. This is defined as the inverse constraint learning (ICL) setting. We will further elaborate on this setting in Chapter 2.

Briefly, the kind of constraints that can be learned in the ICL setting are (a) a hard constraint, (b) an expected constraint, or (c) a probabilistic constraint. Note that in this work, we consider trajectory level constraints rather that instantaneous constraints[2].

Prior work has proposed methods to learn a *hard constraint* [25, 27, 26, 120, 88] from expert demonstrations. Such a constraint can either be defined as a constraint set (i.e. learning which state-action pairs are forbidden to visit) or as a cumulative constraint that must be satisfied for all trajectories.

In contrast, an *expected or soft* constraint is satisfied in expectation across a batch of trajectories [42, 134]. This means that some of the trajectories in the batch may violate the constraint, but on average the constraint will be satisfied. Such a constraint can take into account noise in sensor measurements and some possible violations in expert demonstrations. In this work, we propose a novel (and the first) framework called ICL to learn such an expected constraint (for simplicity, the framework is also called ICL). To formulate our method, we adopt the framework of Constrained Markov decision processes (CMDPs) [4], where an agent seeks to maximize expected cumulative rewards subject to

[1]This does not completely solve the unidentifiability problem since different constraints could still yield the same optimal policy. However, for the purposes of this work it will be sufficient to find "a" constraint that can explain the expert demonstrations.

[2]defined for each state-action pair that was visited

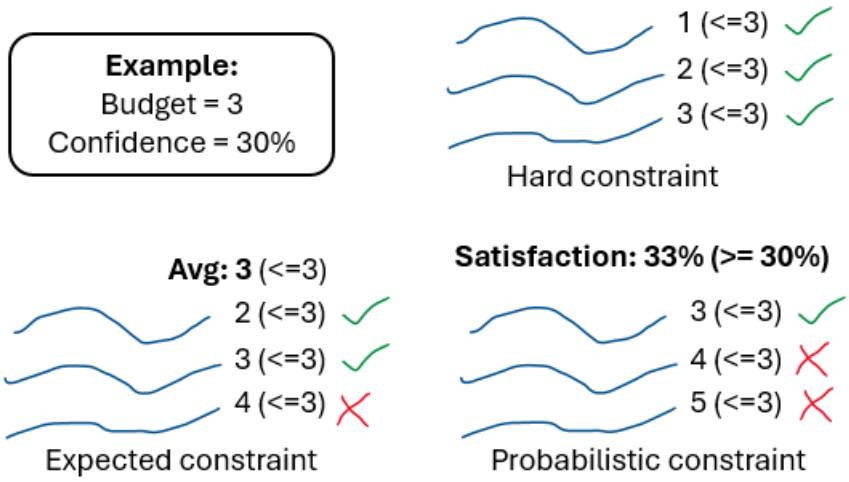


Figure 1.5: An example illustrating the different types of constraints. Here, blue lines correspond to various trajectories, green ticks refer to a constraint being satisfied in a trajectory and red crosses refer to a constraint being violated in a trajectory. The numbers represent the cumulative costs for individual trajectories. A constraint being satisfied for a trajectory refers to the condition being met that the cumulative cost for the trajectory is within the budget. Based on these notions, there are three types of constraints: (a) a hard constraint needs to satisfy the constraint for each trajectory, (b) an expected constraint satisfies the constraint in expectation across a batch of trajectories, and (c) a probabilistic constraint lower bounds the degree of constraint satisfaction across trajectories.

cumulative constraints. In IRL fashion [6], we iteratively adjust a constraint function until the agent's behaviour matches the expert's behaviour.

We can further define a *probabilistic or chance* constraint. Unlike hard constraints where no violations are allowed, and expected constraints where some violations are allowed subject to the average constraint value being within a budget, a probabilistic constraint directly specifies an upper threshold on the amount of violations allowed, or equivalently, a lower bound on the degree of constraint satisfaction across trajectories. We propose two methods to learn such a constraint: one where we reuse ICL to learn an expected constraint that by sufficiency also satisfies a related probabilistic constraint, and another, where we learn the probabilistic constraint directly.

1.4 Challenges in constraint learning

There are some key challenges in learning constraints from demonstrations. Some of these challenges are as follows:

1. As mentioned previously, a set of demonstrations can be generated by two or more equivalent reward-constraint combinations (i.e. *unidentifiability*). Due to this, there may be multiple valid constraints that can be recovered from a single set of expert demonstrations. We consider the setting of learning a single canonical constraint from demonstrations assuming the reward is known in advance, which mitigates some of this ambiguity (explained in Section 2.9.1).

2. The nature of the environment may make learning a constraint more challenging, for example, stochastic environments, multi-agent settings or partially observable environments. We particularly address the challenge of stochastic environments in this work (Appendix A.2).

3. The learned constraint may overfit to the demonstrations. Due to this, it may not generalize well to unseen demonstrations. The methods proposed in this work do not seem to suffer from overfitting, at least empirically.

4. High dimensional input spaces may make a constraint more challenging, when the underlying constraint truly only depends on a subset of the input features. We particularly address the challenge of high dimensional input spaces in Chapter 6.

1.5 Contributions

Our contributions can be stated as follows.

1. We provide a novel formulation and method to learn an *expected or soft* constraint in the ICL setting. Our formulation poses ICL as a two phase alternating procedure, that alternates between constrained policy optimization and constraint adjustment. Our method can learn an arbitrary constraint (e.g. disjunction or conjunction of several simple constraints) since the constraint is parameterized by a neural network.

2. We discuss how to translate an expected constraint into a probabilistic constraint. Specifically, we provide a theorem that shows that an expected constraint with a

modified threshold implies a probabilistic constraint. We then learn a probabilistic constraint using a prior method designed for learning an expected constraint.

3. We propose a novel formulation and method to learn a *probabilistic or chance* constraint in the ICL setting. Similar to the expected constraint formulation, we again pose IPCL as a two phase alternating procedure, however instead of learning an expected constraint, we learn a probabilistic constraint.

4. We provide convergence results for the proposed methods.

5. We propose an approach to select the input features for the proposed ICL algorithms. To do this feature selection, we use a simple statistical test from hypothesis testing literature. We also provide a variant of our algorithm with ICL that mitigates feature overlap.

6. We validate the proposed methods by conducting experiments with synthetic environments, robotics environments and environments based on real world driving datasets.

1.6 Structure of this work

This work is structured as follows.

- In this chapter, i.e. Chapter 1, we first introduce the need for learning various types of constraints and elaborate on the challenges in learning constraints.
- In Chapter 2, we lay out the preliminaries, background and mathematical notion required for the rest of this work.
- In Chapter 3, we discuss related work in constraint learning and feature selection, in addition to applications of constraint learning.
- In Chapter 4, we discuss the ICL framework that can be used to learn an expected constraint from expert demonstrations. We validate the proposed method and discuss our results, also answering related questions.
- In Chapter 5, we discuss the IPCL framework that can be used to learn a probabilistic constraint from expert demonstrations. We also describe how to use existing methods that learn an expected constraint to learn a probabilistic constraint instead. Later, we validate our method and discuss our results.

- In Chapter 6, we discuss the need for learning a constraint with a reduced input space, and later provide a practical algorithm to perform this feature selection using a simple statistical test from hypothesis testing literature.
- In Chapter 7, we conclude this work, discuss limitations and potential future research.

Chapter 2

Preliminaries

2.1 Markov decision process (MDP)

We begin with the concept of MDPs, which formalize the problem of learning from interaction through sequential decision making. The MDP formalism is a popular tool, which is very widely used to represent sequential decision making problems in the RL literature.

The interaction process in an MDP is illustrated in Figure 2.1. The decision maker, or the agent, repeatedly interacts with the environment. Here, the environment consists of everything except the agent, and is represented by an MDP. More concretely, the interaction at time t happens as follows. The agent receives the environment's state $S_t \in \mathcal{S}$ and selects an action $A_t \in \mathcal{A}$. The environment transitions from state S_t to S_{t+1} and produces a reward signal. Then, at the next time step $t+1$, the agent receives the reward signal R_{t+1} and environment transitions into a new state $S_{t+1} \in \mathcal{S}$.

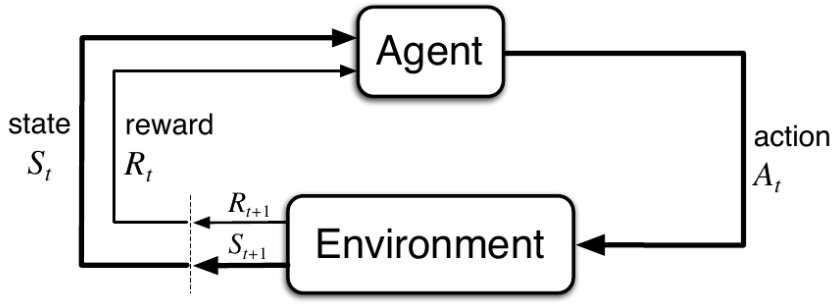


Figure 2.1: Interaction process between the agent and environment. Source: [127]

To understand the MDP formalism better, we can consider the following example of a delivery truck's routine (shown in Figure 2.2). The delivery truck starts from an initial state where it is located in Warehouse 1. From here, it can take 3 actions, A1, A2 and A3 and receive rewards +5, +3, +1 respectively depending on the payment received for the consignment minus the fuel costs. For each of these actions, the truck reaches either Company A, B, or C respectively. Afterwards, it can return to Warehouse 2 by taking the action A4. In this case, it receives a reward of -1 due to the additional fuel cost incurred to reach Warehouse 2. After the truck has reached Warehouse 2, it stays at Warehouse 2.

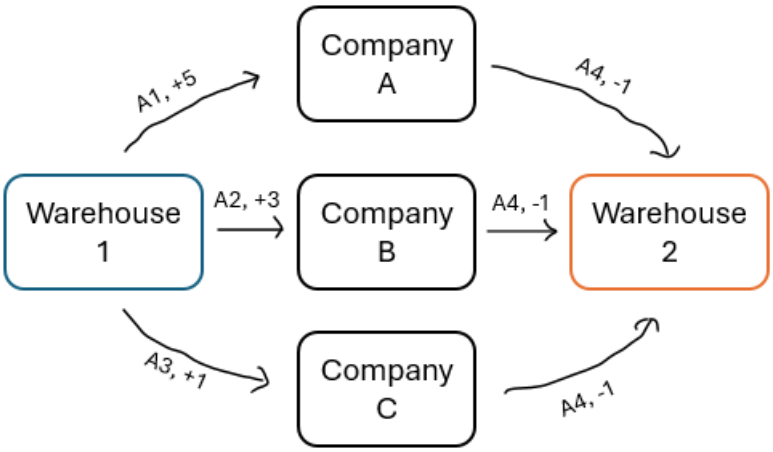


Figure 2.2: MDP example: delivery truck's routine. The possible actions are A1, A2, A3 and A4. The states correspond to the various locations of the truck. Depending on the state and the action taken, the agent receives different rewards.

The objective of the delivery truck, and agents in general, is to maximize the long term reward. Here, we can see that the long term reward is maximized if the delivery truck goes to Company A and then reaches Warehouse 2 (total reward = 5 - 1 = 4). In general, the problem is to maximize the average long term reward across several episodes (or trajectories).

Now, we define the notion of a trajectory. A trajectory or an episode is defined as a sequence of state action pairs (or transitions) until (a) the time horizon or limit h is reached, or (b) the trajectory encounters a terminal state. In the example, this terminal state corresponds to reaching Warehouse 2. Mathematically,

$$\tau := (S_0, A_0, S_1, A_1, S_2, A_2, \cdots) \tag{2.1}$$

With this notation, the interaction process is defined as follows. The agent starts out in a start state determined by the start state distribution $S_0 \sim \mu(S)$. Then it takes an action A_1 and transitions into state S_1 determined by the next state or transition distribution. This process is repeated until the trajectory ends. At every discrete time step $t \geq 0$, the agent observes the state S_t from the environment, takes an action A_t sampled from its stochastic policy $A_t \sim \pi(A|S)$ (this can also be a deterministic policy $A_t = \pi(S_t)$), and transitions into the next state S_{t+1} sampled from the transition distribution, i.e. $S_{t+1} \sim p(S|S_t, A_t)$.

Further, sampling a trajectory refers to the following shorthand:

$$\tau \sim \pi \iff S_0 \sim \mu; A_t \sim \pi(\cdot|S_t); S_{t+1} \sim p(\cdot|S_t, A_t), \forall t \tag{2.2}$$

Using this notation, we can define the objective of the agent as trying to maximize the long term expected reward (or expected return, where the return is the reward accumulated from a certain timestep till the end of the trajectory).

$$\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} r(S_t, A_t)\right] \tag{2.3}$$

A policy that maximizes this objective is called an *optimal policy*. Here, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ refers to the reward function, which is the instantaneous reward signal that the agent receives after performing an action, at the next timestep. Note that the reward itself may be sampled from a distribution, but for the purposes of this work, we assume it is a deterministic function of the immediate state and action.

We can also consider the notion of discounted rewards. In general, a discounted sum of rewards (or a discounted return) can capture the farsightedness of the agent. For this, we can introduce a discount factor $0 \leq \gamma \leq 1$. In particular, we can define a discounted return as:

$$G_r(\tau; \gamma) = G_r(\tau) := \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \tag{2.4}$$

Then, the discounted objective becomes:

$$\max_{\pi} \mathbb{E}_{\tau\sim\pi}[G_r(\tau)] \tag{2.5}$$

The purpose of the discount factor can be understood as follows. As γ approaches 0, the agent weighs future reward lesser and is therefore myopic or near-sighted. If γ approaches 1, the agent weighs future rewards more and is therefore far-sighted. Moreover, the discount factor γ multiplied in a geometric fashion makes the sum of rewards finite, especially when the trajectory itself has infinite time steps. The discounted nature of the objective also means that the objective satisfies the Bellman equation:

$$\mathbb{E}_{\tau\sim\pi}[G_r(\tau)|S_0 = s] = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[r(s,a) + \gamma \mathbb{E}_{\tau\sim\pi}[G_r(\tau)|S_0 = s']\right] \tag{2.6}$$

We can show that the Bellman equation has a unique solution that coincides with the optimal policy that maximizes the discounted objective.

With these notions explained, we can now formally define an MDP as follows. An MDP is a collection or tuple $(\mathcal{S}, \mathcal{A}, p, \mu, r, \gamma)$, where

- $\mathcal{S}$ is the state space, i.e. the set of all possible states
- $\mathcal{A}$ is the action space, i.e. the set of all possible actions
- $p(s'|s,a)$ is the transition probability of going from state $s \in \mathcal{S}$ to another state $s' \in \mathcal{S}$ when action a is taken
- μ is the initial state distribution over the state space $\mathcal{S}$, i.e. how likely is a state to be an initial state
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function
- $\gamma \in [0, 1]$ is the discount factor

We assume that the behaviour of the agent is represented using a *stochastic* policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ such that $\sum_{A\in\mathcal{A}} \pi(A|S) = 1$. Note that the definition of stochastic policy encompasses the definition of a deterministic policy, hence this assumption is okay in practice. The objective of the agent (control) is to maximize the expected long term discounted reward.

2.1.1 Solution time complexity

Given the components of an MDP (assuming a finite state-action space), including the transition function and the reward function, the optimal policy or value can be computed using policy iteration or value iteration algorithms, which have polynomial time complexity in the number of states and actions.

2.1.2 Sampling datasets

Sampling a dataset refers to the following shorthand:

$$\mathcal{D} \sim \pi \iff \mathcal{D} = \{\tau_i; \tau_i \sim \pi\}_{i=1}^{N} \tag{2.7}$$

2.1.3 Dataset dissimilarity

Before defining the notion of dissimilarity between datasets, we first define a metric. A *metric* is a distance function (that maps two mathematical objects to a positive real number) which follows four properties: (a) the distance between the same two objects is 0, i.e. $d(\mathbf{x}, \mathbf{x}) = 0$, (b) the distance between two different objects is positive, i.e. $\forall \mathbf{x} \neq \mathbf{y}, d(\mathbf{x}, \mathbf{y}) > 0$, (c) the distance between two objects is symmetric, i.e. $\forall \mathbf{x}, \mathbf{y}, d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$, and (d) the distance between any two objects in a set of three objects satisfies the triangle inequality, i.e. $\forall \mathbf{x}, \mathbf{y}, \mathbf{z}, d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \geq d(\mathbf{x}, \mathbf{z})$. A simple example of such a distance metric is the Euclidean distance defined for two vectors:

$$d_{euc}(\mathbf{x}, \mathbf{y}) := \|\mathbf{x} - \mathbf{y}\|_2 \tag{2.8}$$

In the same way, a *premetric* is a distance function that satisfies at least the first two conditions out of the four conditions listed above. By definition, a metric is also a premetric. An example of a premetric is the KL divergence defined for two probability distributions:

$$d_{KL}(p, q) := \int p(x) \log \left[\frac{p(x)}{q(x)}\right] dx \tag{2.9}$$

Using these notions, we can now define a distance function d_{dis} that can measure the dissimilarity between two demonstration datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ as a premetric over the space of dataset objects:

$$d_{dis}(\mathcal{D}_1, \mathcal{D}_2; \mathcal{D}_1 \sim \pi^1, \mathcal{D}_2 \sim \pi^2) \tag{2.10}$$

We do not formally define this distance function, but use an empirical estimation of it later on, called the normalized accrual dissimilarity (see Section 4.3.4 and Section 5.3.4).

2.2 Reinforcement learning (RL)

There are two kinds of problems that we encounter in reinforcement learning: (a) given a policy, if we just wish to obtain an estimate of the expected return quantity, it is called the *prediction* problem; (b) if we wish to maximize the objective to obtain an optimal policy, it is called the *control* problem. In this work, we are concerned with the control problem, i.e. an RL process obtains an optimal policy that maximizes the discounted objective.

$$\text{RL}(r) = \pi^* := \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}[G_r(\tau)] \tag{2.11}$$

A key difference between RL and the MDP objective is that in RL, we learn an optimal policy purely using samples, assuming that the transition dynamics and the reward function are unknown or unavailable, and thus cannot be evaluated. In particular, the samples are obtained through the interaction between the agent and the environment. In this way, RL is a relaxation of the MDP formalism where certain objects (reward and transition dynamics) are not directly available.

2.3 Constrained Markov decision process (CMDP)

As explained in Section 1.2, sometimes it may be more amenable to specify behaviour through additional behavioural constraints. For this purpose, it is useful to work with CMDPs.

The interaction process between the agent and environment (represented by the CMDP) is slightly different compared to the MDP setting. More specifically, the agent still receives a state $S_t \in \mathcal{S}$ and takes an action $A_t \in \mathcal{A}$ according to its policy, but at the next time step it receives both the reward signal R_{t+1} and cost signal C_{t+1} while the environment transitions to the next state S_{t+1} according to its transition dynamics.

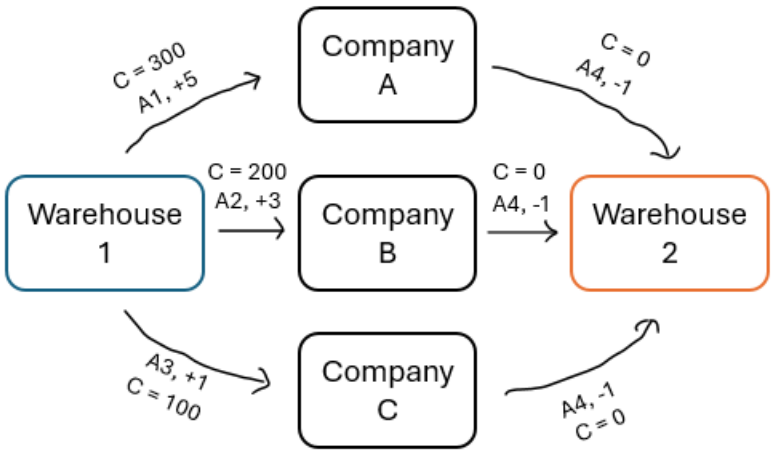


Figure 2.3: CMDP example: delivery truck's routine with payload constraint. Depending on the state and the action taken, the agent receives different rewards and costs. Here, the cost represents the truck weight for the journey.

To understand the CMDP formalism better, we can again consider the example of a delivery truck, as shown in Figure 2.3. Suppose the truck has an operating condition that, at all times, the weight of the truck should not exceed 200 kg. As shown in the figure, the weight of the truck if going to Company A, B, and C are 300 kg, 200 kg and 100 kg respectively. With the specified constraint, the previously optimal policy of going to Company A and then going to Warehouse 2 is no longer optimal since it violates the operating constraint. Instead, now it is optimal to go to Company B and then to Warehouse 2 since that maximizes the long term reward while satisfying the constraint.

As seen in this example, in a CMDP setting, the objective of the agent is to maximize its long term reward subject to satisfying additional constraints (suppose there are K constraints). Formally, a CMDP augments the MDP collection or tuple with one or more constraint functions or costs ($c_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+, \forall i$) [4]. If the constraints are expected, then the episodic constraint thresholds $\beta_i, \forall 1 \leq i \leq K$ are also augmented. That is,

$$\mathcal{M} := (\mathcal{S}, \mathcal{A}, p, \mu, r, \gamma, \{c_i, \beta_i\}_{i=1}^K) \tag{2.12}$$

The corresponding objective for a CMDP is:

$$\max_{\pi} \mathbb{E}_{\tau\sim\pi}[G_r(\tau)] \text{ s.t. } \mathbb{E}_{\tau\sim\pi}[G_{c_i}(\tau)] \leq \beta_i, \forall 1 \leq i \leq K \tag{2.13}$$

In this case, the optimal policy must additionally satisfy the expected constraints.

If the constraints are probabilistic, then both the constraint and probabilistic thresholds $\beta_i, \delta_i, \forall i$ are also augmented to the collection or tuple. That is,

$$\mathcal{M} := (\mathcal{S}, \mathcal{A}, p, \mu, r, \gamma, \{c_i, \beta_i, \delta_i\}_{i=1}^{K}) \tag{2.14}$$

Then, the corresponding objective for a probabilistic CMDP is:

$$\max_{\pi} \mathbb{E}_{\tau\sim\pi}[G_r(\tau)] \text{ s.t. } \mathbb{P}_{\tau\sim\pi}\Big(G_{c_i}(\tau) \leq \beta_i\Big) \geq \delta_i, \forall 1 \leq i \leq K \tag{2.15}$$

In this case, the optimal policy satisfies the additional probabilistic constraints.

We can better understand the difference between expected and probabilistic constraints through Figure 2.4 (repeated from Figure 1.5). An expected constraint satisfies the constraint in expectation across a batch of trajectories, and a probabilistic constraint lower bounds the degree of constraint satisfaction across trajectories. In the given figure, the discounted cumulative costs for 3 trajectories are given. With a budget of 3, the expected constraint is satisfied in the given example since the average discounted cumulative cost is 3. Similarly, with a budget of 3 and a satisfaction threshold of 30%, the probabilistic constraint is also satisfied since the constraint is satisfied in 33.33% of trajectories individually.

For ease of notation, we also define the threshold set to be:

$$\Gamma_\beta := \{\beta_i\}_{i=1}^{K} \tag{2.16}$$

$$\Gamma_{\beta,\delta} := \{\beta_i, \delta_i\}_{i=1}^{K} \tag{2.17}$$

2.3.1 Solution time complexity

Given the components of a CMDP with expected constraints (assuming a finite state-action space), the optimal policy can be expressed in terms of the optimal occupancy measures

Figure 2.4: An example illustrating the difference between expected and probabilistic constraints. Here, blue lines correspond to various trajectories, green ticks refer to a constraint being satisfied in a trajectory and red crosses refer to a constraint being not satisfied in a trajectory. A constraint being satisfied for a trajectory refers to the condition being met that the cumulative cost for the trajectory is within the budget.

that solve the corresponding linear program. Since a linear program can be effectively solved in polynomial time, the optimal policy can also be obtained in polynomial time, polynomial in both the number of variables (size of the finite state action space) and the number of constraints.

2.4 Constrained reinforcement learning (CRL)

Similar to the case of RL, the constraint functions, transition dynamics and rewards are not directly known or accessible in the setting of Constrained reinforcement learning (CRL). In this way, the CRL objective is a relaxation of the CMDP objective specified in Equation (2.13) since we use only samples from the agent-environment interaction. Further, in addition to the reward optimization objective in RL, CRL has constraints that must be satisfied.

2.4.1 Expected CRL

When the constraints are defined using expected constraint functions c_i, we refer to the procedure as expected CRL, or more simply, CRL. The procedure can be defined as:

$$\text{CRL}(r, \{c_i\}_{i=1}^K; \Gamma_\beta) = \pi^* := \arg\max_\pi \mathbb{E}_{\tau\sim\pi}[G_r(\tau)] \text{ s.t. } \mathbb{E}_{\tau\sim\pi}[G_{c_i}(\tau)] \leq \beta_i, \forall 1 \leq i \leq K \tag{2.18}$$

2.4.2 Probabilistic CRL (PCRL)

For a Probabilistic constrained reinforcement learning (PCRL) setting, the bounds are not on the expected constraint function returns, but defined probabilistically in terms of the constraint returns. These constraints are also called *chance constraints* in the CMDP literature. The procedure can be defined as:

$$\begin{aligned} \text{PCRL}(r, \{c_i\}_{i=1}^K; \Gamma_{\beta,\delta}) = \pi^* := \arg\max_\pi \mathbb{E}_{\tau\sim\pi}[G_r(\tau)] \\ \text{s.t. } \mathbb{P}_{\tau\sim\pi}\Big(G_{c_i}(\tau) \leq \beta_i\Big) \geq \delta_i, \forall 1 \leq i \leq K \end{aligned} \tag{2.19}$$

2.5 Inverse learning

Next, we define the various inverse learning procedures relevant to this work.

2.5.1 Inverse reinforcement learning (IRL)

First, we define the process of IRL. IRL does the opposite of RL control objective. In RL, we know the reward function and we wish to obtain the optimal policy that maximizes the expected return. Instead, in IRL, we have access to the optimal (or a near optimal) policy through a demonstration dataset $\mathcal{D} = \{\tau_j\}_{j=1}^N$, and we wish to recover a reward r which best explains the provided dataset, that is, if we perform RL with reward r, the learned optimal policy should have a high probability of generating $\mathcal{D}$.

$$\text{IRL}(\mathcal{D}; \mathcal{D} \sim \pi^*) := r \implies \text{RL}(r) = \pi^* \tag{2.20}$$

This is equivalent to minimizing the dissimilarity between the expert dataset $\mathcal{D}_E$ and the dataset $\mathcal{D}$ generated by the agent's policy:

$$\arg\min_r d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{RL}(r)) \tag{2.21}$$

2.5.2 Inverse CRL (ICRL)

Inverse constrained reinforcement learning (ICRL) similarly tries to obtain a reward function r and the additional constraint functions c_i given access to *constrained* demonstrations $\mathcal{D}$ such that when CRL is performed with $r, c_i \forall i$, the learned optimal *constrained* policy has a high probability of generating $\mathcal{D}$.

$$\text{ICRL}(\mathcal{D}; \mathcal{D} \sim \pi^*) := r, \{c_i\}_i \forall i \implies \text{CRL}(r, \{c_i\}_i; \Gamma_\beta) = \pi^* \tag{2.22}$$

This is equivalent to minimizing the dissimilarity between the expert dataset $\mathcal{D}_E$ and the agent's dataset $\mathcal{D}$, assuming that the agent satisfies additional constraints as a part of the CRL procedure:

$$\arg\min_{r,\{c_i\}_i} d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{CRL}(r, \{c_i\}_i; \Gamma_\beta)) \tag{2.23}$$

Note that the CRL procedure may be replaced with the PCRL procedure if required.

2.5.3 Inverse constraint learning (ICL)

In the ICL setting [42, 88, 120], *we assume that the reward r is known*, and that there is *only one constraint function* that must be recovered. In this work, the recovered constraint function is parameterized by a neural network. These assumptions are not very restrictive as even a single constraint function can represent an arbitrarily complex constraint due to the neural network parameterization.

$$\text{ICL}(r, \mathcal{D}; \mathcal{D} \sim \pi^*) := c \implies \text{CRL}(r, \{c\}; \Gamma_\beta) = \pi^* \tag{2.24}$$

This is equivalent to minimizing the dissimilarity between the expert dataset $\mathcal{D}_E$ and the dataset sampled using the agent's policy, $\mathcal{D}$:

$$\arg\min_{c} d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{CRL}(r, \{c\}; \Gamma_\beta)) \tag{2.25}$$

2.5.4 Inverse probabilistic constraint learning (IPCL)

The difference between the ICL and IPCL setup is that in the case of IPCL, we need to learn a single probabilistic constraint, i.e.

$$\text{IPCL}(r, \mathcal{D}; \mathcal{D} \sim \pi^*) := c \implies \text{PCRL}(r, \{c\}; \Gamma_{\beta,\delta}) = \pi^* \tag{2.26}$$

This is equivalent to:

$$\arg\min_c d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{PCRL}(r, \{c\}; \Gamma_{\beta,\delta})) \tag{2.27}$$

2.6 Problem setup

Suppose we are given access to demonstrations from an expert policy, i.e.

$$\mathcal{D}_E \sim \pi^E \tag{2.28}$$

Then, our objectives in this work are:

1. Provide an ICL procedure that can learn an expected constraint from an expert dataset $\mathcal{D}_E$ (Equation (2.25)):

$$\arg\min_c d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{CRL}(r, \{c\}; \Gamma_{\beta})$$

2. Provide an IPCL procedure that can learn a probabilistic constraint from an expert dataset $\mathcal{D}_E$ (Equation (2.27)):

$$\arg\min_c d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{PCRL}(r, \{c\}; \Gamma_{\beta,\delta}))$$

3. Provide a feature selection procedure that can determine the input features necessary for predicting the constraint function output.

Similar to prior work [27], we learn only the constraint, but not the reward function.

2.7 RL algorithms

2.7.1 Trust region policy optimization (TRPO)

We first discuss TRPO, a family of RL methods that can be used to obtain an optimal policy given a reward function. These Trust region policy optimization (TRPO) algorithms maximize a surrogate objective:

$$\max_{\pi} \mathbb{E}_t \left[\frac{\pi(A_t|S_t;\boldsymbol{\theta})}{\pi(A_t|S_t;\boldsymbol{\theta}_{old})} \hat{A}(S_t, A_t) \right] \tag{2.29}$$

$$\text{subject to } \mathbb{E}_t[\text{KL}[\pi_{\boldsymbol{\theta}_{old}}(\cdot|S_t)\|\pi_{\boldsymbol{\theta}}(\cdot|S_t)]] \leq \delta \tag{2.30}$$

The idea is to maximize a surrogate objective (Equation (2.29)), which is approximately equivalent to the policy improvement quantity (i.e. the change in expected long term reward), but by making a small change to $\boldsymbol{\theta}_{old}$, i.e. the new $\boldsymbol{\theta}$ must lie within a "trust region" as specified by the KL constraint (Equation (2.30)).

Note that here the advantage computation (i.e. computation of $\hat{A}$) corresponds to generalized advantage estimation [118] (r corresponds to the reward function, V corresponds to the value function):

$$\Delta_t = r(S_t, A_t) + \gamma V(S_{t+1}) - V(S_t), t \in [T] \tag{2.31}$$

$$\hat{A}(S_t, A_t) = \Delta_t + (\gamma\lambda)\Delta_{t+1} + \cdots + (\gamma\lambda)^{T-t}\Delta_T \tag{2.32}$$

2.7.2 Proximal policy optimization (PPO)

The PPO objective then builds upon this TRPO objective. In particular, the PPO loss objective is a clipped surrogate objective that takes the minimum of a clipped and an unclipped objective (Equation (2.34)). With this scheme, the objective behaves like a pessimistic bound on the unclipped objective, ignoring the change in the probability ratio (Equation (2.33)) when it makes the objective improve and including it when it makes the objective worse [119]. Finally, minimizing the loss maximizes this objective and maximizes entropy of the policy (Equation (2.35)).

$$\rho_t := \frac{\pi(A_t|S_t;\boldsymbol{\theta})}{\pi(A_t|S_t;\boldsymbol{\theta}_{old})} \tag{2.33}$$

$$\text{PPO-Objective}(\pi) := \mathbb{E}_t\Big[\min(\rho_t, \text{clip}(\rho_t, 1-\epsilon_{\text{PPO}}, 1+\epsilon_{\text{PPO}}))\hat{A}(S_t, A_t)\Big] \tag{2.34}$$

$$\text{PPO-Loss}(\pi) := -\text{PPO-Objective}(\pi) - \lambda_{ent}H(\pi) \tag{2.35}$$

2.8 Solving constrained optimization problems

In this work, we will frequently encounter constrained optimization problems of the following type:

$$\max_y f(y) \text{ such that } g(y) \leq 0$$

There exist several techniques to solve such constrained optimization problems. In this work, we will focus on using either the Lagrangian method or the penalty method.

Note that optimization with constraints is different than traditional optimization which only involves maximization or minimization for an objective. For example, minimizing a potential value (e.g. in robotics) requires us to find the minimizer of this potential, while constrained minimization requires us to find a feasible minimizer. That is, for constrained minimization, the minimizer must lie in the feasible region as dictated by the constraints.

2.8.1 Lagrangian method

The Lagrangian method relaxes the original constrained problem (Section 2.8) into the following min-max problem (λ is the Lagrange multiplier):

$$\min_{\lambda \geq 0} \max_y f(y) - \lambda g(y) \tag{2.36}$$

Here, if we violate the constraint, i.e. $g(y) > 0$, then the overall objective gets penalized. Similarly, if the constraint is satisfied, i.e. $g(y) \leq 0$, then the overall objective value is increased. When the constraint is violated i.e. $g(y) > 0$, λ is increased in order to minimize

the overall objective w.r.t. λ. In such a case, the $g(y)$ term has a larger weight, and the inner optimization chooses a y that has an effect of minimizing $g(y)$ in order to satisfy the constraint better. When the constraint is satisfied i.e. $g(y) \leq 0$, λ is reduced in order to minimize the overall objective w.r.t. λ. In this case, the $g(y)$ term has a lower weight, and the inner optimization chooses a y that has an effect of maximizing $f(y)$. In either case, λ is automatically adjusted by the outer optimization to either satisfy the constraint better, or maximize the original objective $f(y)$.

Many existing algorithms solve such constrained optimization problems from the Lagrangian perspective [17, 11, 128, 16] as a min-max problem that can be handled by gradient ascent-descent type algorithms. In gradient ascent descent, we alternate between the inner maximization step and the outer minimization step, one gradient update at a time. The inner maximization chooses a y to maximize the overall objective given λ:

$$y \leftarrow y + \eta_1 \nabla_y (f(y) - \lambda g(y)) \tag{2.37}$$

The outer optimization chooses a λ to minimize the overall objective, while projecting it to be non negative:

$$\lambda \leftarrow \text{projection}_{\lambda \rightarrow \lambda \geq 0} \{\lambda - \eta_2 \nabla_\lambda (f(y) - \lambda g(y))\} \tag{2.38}$$

Here, the outer optimization runs at a slower rate than the inner optimization i.e. $\eta_2 \ll \eta_1$ [128].

While the Lagrangian method has the benefit of not requiring us to manually set λ, the alternating optimizations cause oscillatory behavior (see Appendix A.3) and difficulty in terms of empirical convergence.

2.8.2 Penalty method

To solve constrained optimization problems, we can also use the less commonly used framework of the penalty method. The penalty method [10, 34, 42] converts the constrained optimization problem[1] in Section 2.8 into an unconstrained problem,

$$\max_y f(y) - \lambda \, \text{ReLU}(g(y))$$

[1] $\text{ReLU}(x)$ is defined as $\max(0, x)$. Additionally, λ is a hyperparameter that controls how strongly the constraint must be satisfied.

Here, λ is a fixed parameter and needs to be set in advance. Note that in this formulation, the unconstrained problem is simply a maximization procedure and hence it doesn't require an alternating gradient ascent descent type algorithm to solve it.

We elaborate on the choice of λ a bit later. First, we describe an algorithm to do constrained optimization using the penalty objective [34]. More specifically, we alternate between feasibility projection and optimizing the overall objective. In feasibility projection, we modify y in order to directly minimize the ReLU term.

$$y \leftarrow y - \eta_1 \nabla_y \text{ReLU}(g(y)) \tag{2.39}$$

Here,

$$\nabla_y \text{ReLU}(g(y)) = \mathbf{I}(g(y) > 0) \nabla_y g(y) \tag{2.40}$$

Doing this for a few steps ensures that the constraint is satisfied, i.e. $g(y) \leq 0$ without maximizing $f(y)$. Then, when we optimize the overall objective, we ensure that the constraint stays satisfied while maximizing $f(y)$:

$$y \leftarrow y + \eta_2 \nabla_y (f(y) - \lambda \text{ReLU}(g(y))) \tag{2.41}$$

As an alternative to Lagrangian based approaches, the penalty method has the advantage of simpler algorithmic implementation. However, the choice of λ is crucial.

A small λ means that in the overall objective optimization, the gradient update of $\text{ReLU}(g(y))$ is miniscule in comparison to the gradient update of $f(y)$, because of which y may not stay within the feasible region during the soft loss optimization. In this case, feasibility projection is important to ensure the overall requirement that feasibility is met (even if exact feasibility cannot be guaranteed). Conversely, a large λ means that maximizing the overall objective is likely to ensure feasibility, and the feasibility projection step may therefore be omitted.

A large λ also means that the algorithm can get stuck in locally feasible regions. In order to get out of such regions, we need to optimize $f(y)$ more (and therefore adjust λ to be lower). Thus, the disadvantage of penalty based approaches is that they require us to manually adjust λ depending upon the result that we wish to achieve.

2.9 Other topics

2.9.1 Unidentifiability

Now, we discuss the topic of unidentifiability or ambiguity in reward and constraint learning. Specifically, we define the notion of an *alternative reward/constraint* as a reward/constraint that is different from the underlying reward/constraint yet it yields the same policy. This problem is prevalent in IRL, ICRL and ICL problems.

In the context of IRL, if r_0 is the true underlying reward (which generates the expert dataset $\mathcal{D}_E$), then r is an alternative reward function if (in the limit of infinite samples):

$$r \overset{\text{IRL}}{\equiv} r_0 \iff d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{RL}(r), \mathcal{D}_E \sim \text{RL}(r_0)) = 0 \tag{2.42}$$

Similarly, in the context of ICL, if c_0 is the true underlying constraint function (which generates the expert dataset $\mathcal{D}_E$), then c is an alternative constraint function if (in the limit of infinite samples):

$$c; \Gamma_\beta \overset{\text{ICL}}{\equiv} c_0; \Gamma_\beta \iff d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{CRL}(r, c), \mathcal{D}_E \sim \text{CRL}(r, c_0)) = 0 \tag{2.43}$$

This definition can be similarly extended to IPCL and ICRL. In the case of IPCL, the CRL procedure is replaced by the PCRL procedure. Similarly, for ICRL, instead of reward or constraint equivalence, we have equivalence between two different reward-constraint combinations.

Essentially, it is difficult to say when a demonstrated behaviour is obeying a constraint, or maximizing a reward, or doing both. So, for simplicity, we assume the (nominal) reward is given, and we just need to learn a (single) constraint function. Without loss of generality, we fix the threshold β to a predetermined value and learn only a constraint function c.

In the context of ICL, mathematically equivalent constraints can be obtained by multiplying the constraint function and the threshold by the same value.

$$c; \Gamma_\beta \overset{\text{ICL}}{\equiv} \alpha c; \alpha\Gamma_\beta; \alpha > 0, \text{ where } k\Gamma_\beta = \{k\beta_i\}_i \forall i \tag{2.44}$$

In this equivalence relation, we say that two constraint functions c and αc (c multiplied by α for all values of c) are equivalent when the constraint thresholds are also multiplied by the same value. The equivalence holds between the constraint functions where the

thresholds (represented by the threshold set Γ_β) are known or given for both sides of the relation.

Another possible unidentifiability arises when we do constraint function shaping, similar to reward shaping. That is, for a given potential Φ,

$$c'(s, a, s') = c(s, a, s') + \gamma\Phi(s') - \Phi(s) \tag{2.45}$$

It can be shown that modifying β as follows does not change the constraint:

$$\beta' = \beta - \mathbb{E}_{S\sim\mu}[\Phi(S)] \tag{2.46}$$

This is because

$$\mathbb{E}_{\tau\sim\pi}\left[\sum_t \gamma^t c'(S_t, A_t, S_{t+1})\right] = \mathbb{E}_{\tau\sim\pi}\left[\sum_t \gamma^t c(S_t, A_t, S_{t+1})\right] - \mathbb{E}_{S\sim\mu}[\Phi(S)] \tag{2.47}$$

And hence the following constraint is equivalent to the original constraint,

$$\mathbb{E}_{\tau\sim\pi}\left[\sum_t \gamma^t c'(S_t, A_t, S_{t+1})\right] \leq \beta' \tag{2.48}$$

By fixing the reward and β in ICL, we reduce this unidentifiability and learn a certain constraint within the set of equivalent constraints. However, the unidentifiability is not completely resolved. This is because there are other possible sources of unidentifiability that result in constraint ambiguity. For example, changing the constraint function by a differential amount while setting β to a comparably large amount will not change the resulting policy if the constraint satisfaction condition can still be met, for certain specific CMDP settings.

2.9.2 Entropy and mutual information

We will also need to use the concepts of entropy and mutual information for Chapter 6.

The entropy of a discrete random variable X conveys its uncertainty (note that this definition can be extended to continuous variables by replacing the summation with an integral over the domain):

$$H(X) := -\sum_X p(X) \log p(X) \tag{2.49}$$

Similarly, the conditional entropy is defined as:

$$H(X|Y) := -\sum_{X,Y} p(X,Y) \log \left(\frac{p(X,Y)}{p(Y)} \right) \tag{2.50}$$

Then, the mutual information between variables X, Y is defined as:

$$I(X;Y) = H(Y) - H(Y|X) = H(X) - H(X|Y) \tag{2.51}$$

2.9.3 Reparameterization trick

We will also need to understand the concept of reparameterization trick for the material described in Chapter 5. Essentially, the reparameterization trick is a technique that allows us to differentiate through objectives that depend on sampling from a distribution. Traditionally, it is difficult to differentiate through the operation of sampling, but this trick allows us to rewrite these objectives and makes the quantities easily differentiable.

Consider the problem of supervised policy learning, i.e. learning a stochastic policy from a dataset of state action pairs $\mathcal{D} = \{(\boldsymbol{s}_i, \boldsymbol{a}_i)\}_{i=1}^K$. We will consider two cases: one where the policy is parameterized as a Gaussian distribution and another, where the policy is parameterized as a Categorical distribution.

Continuous variables (Gaussian)

Suppose the stochastic policy is parameterized as a Gaussian, i.e. it outputs a mean vector $\boldsymbol{\mu}(s)$ and a variance vector $\boldsymbol{\sigma}(s)$.

The loss can be specified using the mean squared error objective:

$$\mathcal{L}(\pi_\theta) = \mathbb{E}_{\hat{\boldsymbol{a}}_i \sim \pi_\theta(\boldsymbol{s}_i)} \left[||\boldsymbol{a}_i - \hat{\boldsymbol{a}}_i||_2^2 \right] \tag{2.52}$$

Using the definition of the Gaussian policy, the loss function becomes:

$$\mathcal{L}(\pi_\theta) = \mathbb{E}_{\hat{\boldsymbol{a}}_i \sim \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{s}_i), \boldsymbol{\sigma}_\theta(\boldsymbol{s}_i))} \left[||\boldsymbol{a}_i - \hat{\boldsymbol{a}}_i; ||_2^2 \right] \tag{2.53}$$

This quantity is difficult to differentiate, since the parameters are a part of the sampling operation. The reparameterization trick rewrites the sampling using the following sum

(here, $\boldsymbol{\epsilon}_i$ is sampled from a standard normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and $\odot$ refers to the elementwise product) [61]:

$$\hat{\boldsymbol{a}}_i = \boldsymbol{\mu}_\theta(\boldsymbol{s}_i) + \boldsymbol{\epsilon}_i \odot \boldsymbol{\sigma}_\theta(\boldsymbol{s}_i) \tag{2.54}$$

This rewriting makes the error objective easily differentiable, since we can now just sample several $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and compute the gradient of the expectation, as the expectation of the individual gradient terms, which can be computed through basic vector operations and automatic differentiation.

$$\nabla_\theta \mathcal{L}(\pi_\theta) = \nabla_\theta \mathbb{E}_{\hat{\boldsymbol{a}}_i \sim \mathcal{N}(\boldsymbol{\mu}_\theta(\boldsymbol{s}_i), \boldsymbol{\sigma}_\theta(\boldsymbol{s}_i))} \left[\| \boldsymbol{a}_i - \hat{\boldsymbol{a}}_i; \|_2^2 \right] \tag{2.55}$$

$$= \mathbb{E}_{\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})} \left[\nabla_\theta \left\{ \| \boldsymbol{a}_i - (\boldsymbol{\mu}_\theta(\boldsymbol{s}_i) + \boldsymbol{\epsilon}_i \odot \boldsymbol{\sigma}_\theta(\boldsymbol{s}_i)) \|_2^2 \right\} \right] \tag{2.56}$$

Discrete variables (Categorical)

For discrete categorical variables, we can consider the Gumbel max reparameterization trick as follows [57]. This trick is a general way to approximate samples from the Categorical distribution.

Let Z be a Categorical variable with class probabilities $\theta^k, \forall k$. Then, a one hot sample from this Categorical distribution can be obtained through the Gumbel max trick:

$$\boldsymbol{z} = \text{one-hot}[\arg\max_k \{g^k + \log \theta^k\}] \tag{2.57}$$

Here, $g^k, \forall k$ are i.i.d. samples drawn from Gumbel(0, 1) distribution. We refer to a vector of such variables as $\boldsymbol{g} \sim \{\text{Gumbel}(0, 1)\}^K$, where K is the number of individual Gumbel(0, 1) samples. The purpose of the one-hot operation is to convert the sample to a one hot encoding. This is useful when computing objectives like the cross entropy loss:

$$\mathcal{L}(\pi_\theta) = \mathbb{E}_{\hat{\boldsymbol{a}}_i \sim \pi_\theta(\boldsymbol{s}_i)} [\text{Cross-Entropy}(\boldsymbol{a}_i, \hat{\boldsymbol{a}}_i)] = \mathbb{E}_{\hat{\boldsymbol{a}}_i \sim \text{Categorical}_\theta(\boldsymbol{s}_i)} [-\boldsymbol{a}_i \odot \log \hat{\boldsymbol{a}}_i] \tag{2.58}$$

The Gumbel(0, 1) distribution can be sampled by using inverse transform sampling, i.e. drawing $u \sim \text{Uniform}(0, 1)$ and computing $g = -\log(-\log u)$. This is because the Cumulative density function (CDF) of the Gumbel(0, 1) distribution is

$$F(x) = e^{-e^{-x}} \tag{2.59}$$

Thus the inverse CDF is:

$$F^{-1}(u) = -\log(-\log u) \tag{2.60}$$

Since we wish to obtain a differentiable reparameterization, we can consider sampling from the softmax version of the Gumbel max trick. In this version, we can create a *Gumbel softmax* distribution where the class probabilities can be computed as:

$$\hat{\theta}^k = \frac{\exp((\log \theta^k + g^k)/\xi)}{\sum_j \exp((\log \theta^j + g^j)/\xi)}, \forall i \tag{2.61}$$

Here, ξ refers to the temperature parameter. As $\xi \to 0$, the sampling operation represents the original Gumbel max trick since the logit $\hat{\theta}^k$ quantity gets closer to an argmax operation.

Similar to the case of Gaussian reparameterization, this technique makes an objective (like the cross entropy loss) differentiable. More particularly,

$$\begin{aligned}
\nabla_\theta \mathcal{L}(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\hat{\boldsymbol{a}}_i \sim \pi_\theta(\boldsymbol{s}_i)} \left[\text{Cross-Entropy}(\boldsymbol{a}_i, \hat{\boldsymbol{a}}_i)\right] && (2.62)\\
&= \nabla_\theta \left\{\mathbb{E}_{\hat{\boldsymbol{a}}_i \sim \text{Categorical}_\theta(\boldsymbol{s}_i)} \left[-\boldsymbol{a}_i \odot \log \hat{\boldsymbol{a}}_i\right]\right\} && (2.63)\\
&\approx \nabla_\theta \left\{\mathbb{E}_{\hat{\boldsymbol{a}}_i \sim \text{Gumbel-Softmax}_{\hat{\theta}}(\boldsymbol{s}_i)} \left[-\boldsymbol{a}_i \odot \log \hat{\boldsymbol{a}}_i\right]\right\} && (2.64)\\
&= \mathbb{E}_{\boldsymbol{g}_i \sim \{\text{Gumbel}(0,\,1)\}^K} \left[-\nabla_\theta \left\{\boldsymbol{a}_i \odot \log \hat{\boldsymbol{\theta}}_i\right\}\right] && (2.65)
\end{aligned}$$

Chapter 3

Related work

3.1 Constraint learning

3.1.1 Learning with known constraints

In this work, we focus on learning a constraint function assuming that the reward function is already known. This is contrary to prior work which fixes the constraints and learns the reward instead [59, 37, 31]. These methods typically perform RL by using only those trajectories which are safe according to the fixed constraints. This can be ensured if the agent is only allowed to choose from a set of safe actions at every timestep. The set of safe actions are those actions which produce an immediate cost that does not cause the trajectory cost to go above the threshold.

$$\mathcal{A}_{safe}(s_t) = \left\{ a_t \in \mathcal{A}; \sum_{j=0}^{t} \gamma^j c(s_j, a_j) \leq \beta \right\} \tag{3.1}$$

This is also in contrast with a recent approach that assumes the reward itself is unknown while learning constraints [74]. In contrast, our method assumes that the reward is known while learning a constraint.

3.1.2 Constraint sets

Early work in constraint learning focused on learning constraint sets, where individual state-action pairs were considered feasible or infeasible [25, 120, 105, 90]. More precisely, a

constraint set is defined as the set of all those state-action pairs which the agent cannot visit. This is essentially equivalent to a hard constraint which must be satisfied per trajectory. That is, if the constraint set is $\mathcal{C}$, then the equivalent constraint is

$$\sum_{t=0}^{\infty} \gamma^t c(S_t, A_t) \leq 0 \tag{3.2}$$

Here the constraint function c is defined to be 1 for all $(S, A) \in \mathcal{C}$ and 0 for all $(S, A) \notin \mathcal{C}$.

In this work, we propose to learn an expected and a probabilistic constraint, unlike prior methods which learn a hard constraint.

3.1.3 Constraints in robotics

When continuous state-action spaces were considered, e.g. in robotics, the learned constraints would usually follow some task-specific parameterization, e.g. geometric constraints [5, 109], task space constraints [72, 73], etc. In these works, the parameterization can be a constraint matrix [5, 72, 73] or a constraint knowledge base with behaviour clusters [109]. Later work focused on learning more general constraint functions, for example, polytopic constraints [27] or neural network constraints [88].

In this work, we propose learning constraints parameterized by neural networks since they can represent constraints with arbitrary complexity.

3.1.4 Learning from limited or offline data

Another parallel direction of work in constraint learning considers the problem of learning constraints from limited data, i.e., a single trajectory or few trajectories [21, 102, 68, 91, 138, 82]. Learning constraints from limited data is challenging since the constraint can overfit to demonstrations. Unlike these works, we find that the methods proposed in this work do not suffer from overfitting to demonstrations.

More work has focused on learning from partial trajectories [104] and even explored the setting of learning from a dataset of trajectories in an offline fashion [112]. In contrast, we learn from complete trajectories and a part of our algorithm involves several interactions with the environment, which mitigates distribution shift common in offline algorithms.

3.1.5 Stochastic environments

Unlike earlier work which required deterministic environments, newer work allows learning constraints that work with stochastic environments (i.e., environments with stochastic dynamics) [90, 7]. In this work, we showcase the efficacy of our approach with stochastic environments as well.

3.1.6 Hard constraints

More recent work in constraint learning focuses on hard constraints, where the learned constraint must be satisfied for each state-action pair, or for each trajectory sampled from the optimal constrained policy [120, 105, 90, 88]. Hard constraints are a special case of expected constraints as well as probabilistic constraints. More particularly, with $\beta = 0$, an expected constraint behaves like a hard constraint (and specifically like a constraint set) since this type of constraint can only hold when the cumulative costs for all trajectories is 0.

$$\forall c(s,a) \geq 0, \mathbb{E}_{\tau\sim\pi}[G_c(\tau)] \leq 0 \iff \forall \tau, G_c(\tau) = 0 \tag{3.3}$$

Similarly, with $\delta = 1$, a probabilistic constraint behaves like a hard constraint:

$$\mathbb{P}_{\tau\sim\pi}(G_c(\tau) \leq \beta) = 1 \iff \forall \tau, G_c(\tau) \leq \beta \tag{3.4}$$

Most of these methods [120, 88, 7, 75] use or extend the maximum entropy IRL formulation [141]. In the maximum entropy IRL formulation, the probability of a trajectory in a deterministic MDP (i.e., deterministic transitions) is proportional to the reward earned by that trajectory:

$$p(\tau) \propto e^{R(\tau)} \tag{3.5}$$

Then, the probability of a trajectory under a hard constraint is additionally dependent on whether the constraint is satisfied ($\mathbf{I}$ is the indicator function subject to the condition in the parentheses):

$$p(\tau) \propto e^{R(\tau)}\mathbf{I}(C(\tau) \leq \beta) \tag{3.6}$$

There are other approaches that formulate the constraint learning problem from a Bayesian perspective [26, 103, 104]. In this formulation, the posterior probability of a constraint (set) C given a trajectory τ can be expressed using the Bayes rule:

$$p(C|\tau) = \frac{p(\tau|C)p(C)}{p(\tau)} \tag{3.7}$$

Here, the likelihood $p(\tau|C)$ can be specified using the maximum entropy formulation, $p(C)$ is the prior over constraint (set) and $p(\tau)$ is the normalizing factor.

In this work, we propose learning expected and probabilistic constraints over hard constraints, and do not use either the maximum entropy or Bayesian formulation, since these formulations have a fundamental assumption (in the indicator function of Equation (3.6)) that a hard constraint is being learned. Instead, we propose using an alternating optimization procedure to learn expected and probabilistic constraints.

3.1.7 Constraints under uncertainty

There is also some new research that aims to learn a distribution over constraints [75]. In this framework, instead of learning a deterministic constraint function $c(s, a)$, the distribution of a feasibility variable $p(\phi|s, a)$ is learned. Then, the posterior can be learned approximately (i.e., $q(\phi|\mathcal{D})$) from an expert dataset $\mathcal{D}$ using evidence lower bound optimization:

$$\max_q \mathbb{E}_q[\log p(\mathcal{D}|\phi)] - d_{KL}(q(\phi|\mathcal{D}), p(\phi)) \tag{3.8}$$

Another line of research models the uncertainty associated with learning constraints [134]. In this work, learning constraints from demonstrations is posed in a risk sensitive RL setting. Mitigating uncertainty in learning these constraints then requires us to mitigate aleatoric uncertainty (i.e., uncertainty due to inherent stochasticity of the specification), and epistemic uncertainty (i.e., uncertainty due to limited training data and reasoning with out of distribution data). Since this work can learn probabilistic constraints, we compare our method for learning a probabilistic constraint to the method proposed in this work.

3.1.8 Learning both rewards and constraints

There is recent work that attempts to learn both reward(s) and constraint(s) simultaneously given access to expert demonstrations. Notably, [105] learn multiple local rewards

and local constraints from expert data using a Bayesian strategy. In this approach, the overall problem is decomposed into smaller local problems with different subgoals, and then constraints are learned that maximize the likelihood of the observed trajectory within each of these local problems. Here, both the constraints and rewards are boolean conditions. Specifically, the reward indicates whether a subgoal is reached or not, and a constraint indicates whether the i-th feature is close to the average value of the i-th feature or not. Moreover, the work assumes a discrete state-action space. In contrast, our method allows learning arbitrarily complex constraints for both discrete and continuous state-action spaces.

Similarly, [81, 82, 83] propose a bi-level optimization problem to learn both the reward and constraint from expert data. Specifically, in the upper level, they maximize the likelihood of the expert data while at the lower level, they ensure that the CRL procedure satisfies the constraints. More concretely, the lower level can be formulated as a maximum entropy constrained RL problem, where the expert and agent features are matched [141]:

$$\max_{\pi} H(\pi) + \mathbb{E}_{\tau\sim\pi}[G_r(\tau)] \text{ s.t. } \mu(\pi) = \mu(\pi^E) \tag{3.9}$$

Here, $\mu(\pi)$ is defined as the expected cumulative cost feature of policy π, similar to the maximum entropy IRL [141].

This optimization problem can be solved through its dual form:

$$\max_{\pi} J(\pi, c) := H(\pi) + \mathbb{E}_{\tau\sim\pi}[G_r(\tau)] - \lambda(\mu(\pi) - \mu(\pi^E)) \tag{3.10}$$

The Lagrange multiplier λ multiplied by the expected cumulative cost feature can be absorbed into the definition of the expected cost:

$$\max_{\pi} J(\pi, c) := H(\pi) + \mathbb{E}_{\tau\sim\pi}[G_r(\tau)] - (\mathbb{E}_{\tau\sim\pi}[G_c(\tau)] - \mathbb{E}_{\tau\sim\pi^E}[G_c(\tau)]) \tag{3.11}$$

The outer level maximizes the likelihood of the expert data, so the overall optimization problem formulated as a bilevel optimization procedure is given as:

$$\max_{c} \mathbb{E}_{\tau\sim\pi^E}[\log \mathbb{P}(\tau|\pi^*(c))] \text{ s.t. } \pi^*(c) = \arg\max_{\pi} J(\pi, c) \tag{3.12}$$

In contrast, our work proposes learning constraints through two alternating procedures. Our method can also be rewritten as a bilevel optimization problem where the inner optimization is the same, but the outer optimization is a constrained max-min procedure. Additionally, most of these methods suffer from the problem of unidentifiability which is considerably lesser in the ICL setting.

3.1.9 Multiple experts

Other recent work focuses on learning constraints where the demonstrations are generated from multiple experts, each of which may follow different or even conflicting constraints [111, 81, 83]. In these settings, typically, each expert is identified by a variable z. Then, we expect the expert's policies to have high entropy when z is unknown, and low entropy when z is more precisely known. The optimization problem then entails feature matching (similar to maximum entropy IRL [141]) while the constraints conditioned on the expert are satisfied.

The main challenge in these works is to identify the various experts reliably. In some of these settings, the number of experts are given [81, 83], while in other works, the expert dataset is clustered into various experts in an unsupervised manner [111]. Additionally, the other challenge is that various experts might be collaborating to satisfy a shared constraint or competing due to conflicting constraints. Due to these challenges, in this work, we consider that the expert dataset is generated by a single optimal expert.

3.1.10 Open problems

Here, we summarize the open questions that still exist in the field of constraint learning, and provide explanation regarding the gaps that this work addresses.

Unidentifiability

While the problem of unidentifiability has been well studied in the IRL literature [124], there is still (almost) no work that studies the problem of unidentifiability in ICRL and ICL from a principled perspective. Unidentifiability is especially a big problem in the setting of learning both rewards and constraints, where it has not been studied theoretically either. Existing methods provide assumptions that mitigate some of this uncertainty, or provide heuristic techniques to limit the space of constraint functions that can be learned [88, 120]. In this work, while we do not theoretically analyze unidentifiability, we assume the ICL setting where this unidentifiability issue is considerably reduced.

Dynamic constraints

Most ICRL and ICL literature focuses on learning static constraints, for example, simple constraints like staying in a safe region, etc. These constraints cannot take into account

the dynamic and changing nature of environments. For example, in autonomous driving, for a lane change scenario, the gap to be maintained w.r.t. other vehicles, the velocity/acceleration to be maintained, depends on the real time traffic density and speed of other vehicles. This is an example of a dynamic constraint, which changes depending on the setup of other acting and non-acting entities in the environment. Our methods can learn such dynamic constraints from demonstrations. More specifically, the example of the ExiD lane change scenario for ICL in our experiments involves learning exactly the kind of dynamic constraints described earlier.

Interpretability and large input spaces

While learning a high dimensional constraint is technically possible with our frameworks, it is not easy to visualize and interpret such constraints. We do this by fixing some of the dimensions and visualizing the rest of the dimensions. For the purposes of interpretability, we also ensure that our constraint functions are bounded between 0 and 1, so that they can be interpreted as safeness values, or probability of trajectory feasibility.

Another challenge is that sometimes the input space (state-action space) of the RL setting may be huge, but the constraint itself may just depend on a few of the features. While we can use dimensionality reduction and feature selection techniques from IRL and the classification literature, the problem of reducing the input space while simultaneously learning a constraint function makes the problem challenging. In this work (specifically, Chapter 6), we provide the first method that can tackle feature selection in the context of ICL.

Generalizability of constraints

While constraints are more generalizable than rewards (Chapter 1), they might still have issues transferring to real world scenarios. In real world settings, small disturbances in the environment may cause the learned constraint to transfer incorrectly. This means that the algorithm may produce constraint functions that are not reliable in practice, or sensitive to sensor noise. For example, in the context of autonomous driving, in various scenarios, the constraint function may not transfer correctly due to factors like time of the day and weather conditions. There is some work that has incorporated the advances in robust optimization to learning constraints [135], however, this work still does not account for reward mismatch during transfer and how it affects the occupancy measure [76]. In this work, we do not directly address the problem of generalizability for constraints. However, we propose learning expected constraints over hard constraints prevalent in the literature,

which can take into account noise in sensor measurements as long as the cumulative cost is within a threshold. Similarly, our method to learn a probabilistic constraint allows for specifying the degree of constraint violations.

Multi agent setting, stochasticity or partially observable setting

The nature of the environment is in itself a huge factor that affects the viability of constraint learning from demonstrations. Multi agent environments [81] model real world settings like autonomous driving better than traditional single agent settings since in the real world, there are several agents interacting with each other in a collaborative or competitive manner. Similarly, stochastic environments and partially observable environments capture the real world better.

While there is some work [81, 83] on constraint learning in a multi agent setting, these methods need to be extended to general sum games and need to cover various kinds of equilibria that may exist in these settings. There is also recent work that considers and models the stochasticity of environments [90, 7, 42]. And lastly, there is almost no work that considers partially observable settings for constraint inference.

In this work, we consider the single agent setting, unlike other works that consider the multi agent setting. We also conduct experiments to validate the effect of environment stochasticity for our proposed methods (Appendix A.2).

Learning from preference data

Preference data (comparison data amongst a set of trajectories) is yet another ubiquitous source of data from which we can learn behaviour specification. While there has been a lot of work in learning rewards from preference data [114, 56, 142, 101, 62, 116], we are not aware of any work that considers the problem of learning a constraint given preference data from an expert who is implicitly satisfying both a trivial reward and constraint(s).

3.2 Applications

3.2.1 Expected constraints

Inventory planning

For products in an inventory, the goal could be to minimize holding and backlog cost for the products while satisfying the constraint that expected resource consumption of the products is within a budget [24, Section 6]. Here, a holding cost refers to the cost of holding an item per unit per time period in the inventory. If more items are held, more space and operating costs are incurred, and hence this cost is higher. Similarly, if an item is not sent to fulfill a demand, it may be carried over to the next day. These items incur a backlog cost per unit per time period, similar to the holding cost.

Queue scheduling in network routing

Given pools of servers with different skillsets and customers who wish to use the various servers, the customers typically wait in queue until they are allocated a server. The objective is to find a scheduling policy such that the cost of holding customers in the queue and routing costs are minimized subject to network capacity constraints, which is an expected constraint formulation [24, Section 7].

Robotics

Many robotics settings have safety constraints expressed through expected constraint formulations [70, 117, 128, 2, 49, 29]. An example of such a setting is when a robot is supposed to navigate in an environment while avoiding obstacles [128, 29]. Here, the expected constraint can be on risk of failure [128], or the agent's collision impact [29]. With a lower threshold, the agent learns a conservative and safer policy.

Autonomous driving

In motion planning, the goal could be to reach a destination in shortest time, while having an expected constraint on the number of obstacles encountered [28, 29]. More practically, we could also bound the expected amount of time that the agent spends really close to an obstacle.

Video compression

For online streaming services, it could be useful to optimize the video quality subject to expected constraints on the bitrate [89]. The bitrate refers to the data processed per unit time during streaming. Higher bitrates lead to better quality but also increase the total data transferred during streaming.

Other applications

Apart from these applications, [48] provide an extensive survey on applications of expected constraints.

3.2.2 Probabilistic constraints

Autonomous driving

Probabilistic constraints have been used for the following applications in autonomous driving: ramp and highway merging [87, 54], lane changing [54], path planning [100, 12, 85] and car following [107].

Robotics

Robotics is another domain where probabilistic constraints are used frequently in problem specification, specifically in collision avoidance with mobile robots [86, 106, 12], pedestrian collision avoidance [22], drone simulations [86], Mars rover simulations [100], safe state reachability [55] and unmanned vehicles [121].

Optimal control and model predictive control

Probabilistic constraints are also used with optimal control techniques for the following applications: feed tank (distillation) control [43], water supply management [47], reactors [44], temperature control systems [99, 85].

Power systems

This is another area where probabilistic constraints are used heavily to specify grid behaviour. Some applications include power flow under uncertainty [44] and optimal power flow [123, 126, 139, 45].

Other applications

Apart from these broad applications, chance constrained formulations are also used in applications like travel itinerary recommendation [32], demand estimation and autonomous fleet management [94], war assessment [58] and network analysis [35].

While it is common to formulate real world problems through chance constrained setups, learning the chance constraint from real demonstrations is not common, and in such cases, our method provides a principled way to recover a constraint from such demonstrations.

3.3 Feature selection for constraint learning

3.3.1 Feature selection in classification

Feature selection is a well-studied problem in machine learning. It involves selecting an optimal subset of features for a prediction task [23]. The approaches to feature selection can be divided into several categories:

1. Filter based approaches use techniques like information gain, mutual information based feature selection or hypothesis tests to select features [137, 40, 132, 39, 108, 93, 20, 98, 122, 41]. Correlation based techniques filter out features using correlation analysis [140]. These techniques are usually pretty fast to apply.

2. Evolutionary techniques select feature subsets using genetic algorithms [136, 60]. These methods do not scale well to large input spaces.

3. Branch and bound algorithms provide optimal feature selection without conducting exhaustive searches [96]. Embedding based methods have an objective which encapsulates feature selection through finding an optimal embedding [129, 125]. These methods need lots of training data.

4. Reinforcement learning methods use the framework of MDPs to do feature selection [78, 79, 77, 36, 80]. These methods use a reward defined by prediction accuracy which works well with classification problems.

We validate our proposed method against mutual information based approaches. We choose this family of algorithms since these are theoretically sound and well-tested in general. Additionally, all these techniques focus on feature selection for classification, and cannot be directly used to do feature selection for ICL, since the problem involves learning a constraint and a smaller feature space, compared to classification, where the problem only entails learning a smaller feature space.

3.3.2 Feature selection for inverse reinforcement learning

Feature selection for reward learning has been a known problem since early work in IRL [1]. Reward formulation for high dimensional and/or continuous state-action spaces often involves representing the reward function as a linear combination of relevant and hand-crafted features [110, 64, 8, 131]. Features can be linear or nonlinear functions of states and/or actions.

Levine et al. (2010) [67] propose an algorithm for feature construction (for IRL) that constructs reward features from a large set of component features, many of which can be irrelevant for IRL.

[14, 15] propose a method for learning relevant features using a divide and conquer approach: focusing human effort to learn relevant features using feature traces and later using an algorithm to construct complex and relevant features from the learned base features.

Baimukashev et al. (2024) [9] propose using polynomials as a candidate set for features, and further select a subset of relevant features using correlation based methods in order to reduce reward function complexity. It is important to note that the chosen feature space may not be expressive enough to represent everything required for learning the reward function [50, 13].

There can also be information overlap or correlation between features in the input space which can lead to causal confusion, a phenomenon which can occur when a learned reward achieves low test error but results in poor performance when RL is performed with this learned reward function [30]. We discuss this phenomenon of feature overlap later in the context of inverse constraint learning.

Since all these methods construct composite features from simpler features, they may not select the right subset of features necessary for feature selection. As a result, the learned models do not generalize well to unseen data [15].

Chapter 4

Inverse constraint learning

In this chapter, we will propose a method to learn an expected constraint from expert demonstrations. For clarity, we first highlight the difference (through an example) between hard constraints prevalent in the literature [120, 88] and expected constraints that we wish to learn.

This difference can be explained as follows. Suppose in an environment, we need to obey the constraint "do not use more than 3 units of energy". As a hard constraint, we typically wish to ensure that this constraint is always satisfied for any individual trajectory. The difference between this "hard" constraint and proposed "expected" constraint is that expected constraints are not necessarily satisfied in every trajectory, but rather only satisfied in expectation. This is equivalent to the specification "on average across all trajectories, do not use more than 3 units of energy". In the case of expected constraints, there may be certain trajectories when the constraint is violated, but in expectation, it is satisfied.

Now, we describe the ICL (*Inverse Constraint Learning*) approach [42] that learns an expected constraint from constrained expert demonstrations. Note that both the problem setting and the proposed method are called ICL for simplicity.

4.1 Framework

In order to learn an expected constraint, our objective is to minimize the dissimilarity between the expert dataset and the agent dataset, where the expert dataset is generated by the expert policy and the agent dataset is generated by the policy learned through an algorithm for the ICL setting (Equation (2.25)):

$$\arg\min_c d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{CRL}(r, \{c\}; \Gamma_\beta))$$

We do this minimization empirically by alternating between policy optimization and constraint adjustment until this distance is below a threshold (described later in Algorithm 1).

4.1.1 Alternating procedures and convergence

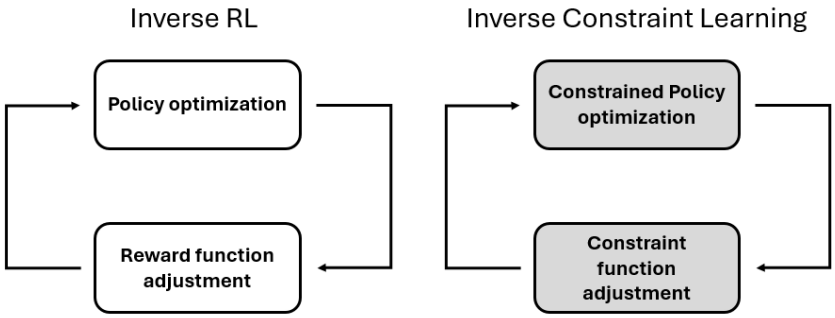


Figure 4.1: IRL vs ICL

The proposed method alternates between a policy optimization phase and a constraint adjustment phase, similar to IRL which alternates between a policy learning phase and a reward learning phase [6]. We first describe a theoretical procedure that captures the essence of this approach and then later adapt it into a practical algorithm. The theoretical approach starts with an empty set of policies $\Pi = \emptyset$ and then grows this set by alternating between the phases of policy optimization and constraint adjustment, adding the learned optimal policy to Π at each iteration. Note that β is a fixed and given parameter.

ICL-Policy-Optimization:

$$\theta^* := \arg\max_\theta \mathbb{E}_{\tau\sim\pi_\theta}[G_r(\tau)] \text{ s.t. } \mathbb{E}_{\tau\sim\pi_\theta}[G_{c_\phi}(\tau)] \leq \beta \tag{4.1}$$

$$\Pi \leftarrow \Pi \cup \{\pi_{\theta^*}\}$$

ICL-Constraint-Adjustment:

$$\phi^* := \arg\max_\phi \min_{\pi_\theta\in\Pi} \mathbb{E}_{\tau\sim\pi_\theta}[G_{c_\phi}(\tau)] \text{ s.t. } \mathbb{E}_{\tau\sim\pi_E}[G_{c_\phi}(\tau)] \leq \beta \tag{4.2}$$

The optimization procedure in Equation (4.1) essentially performs CRL (see Equation (2.4) for a definition of G) to find the parameters of an optimal policy π_{θ^*} given a reward r and a constraint c_ϕ. This optimal policy is added to Π. This is followed by the optimization procedure in Equation (4.2), which adjusts the constraint function c to increase the constraint values of the policies in Π while keeping the constraint value of the expert policy π^E bounded by β. This is achieved by maximizing the accumulated constraint value for the most feasible (lowest total constraint value) policy in Π. This selectively increases the constraint function until the most feasible optimal policies become infeasible. Note that we can also directly perform constraint adjustment by minimizing over the set of all policies, but in practice it is sufficient to minimize over the set Π.

We can show that this approach will converge to a policy equivalent to the expert policy as shown below.

Theorem 1. *Assuming a finite space of policies and a non-negative reward function, the alternation of optimization procedures in Equation (4.1) and Equation (4.2) converges to a set of policies Π such that the last policy π^* added to Π is equivalent to the expert policy π^E in the sense that π^* and π^E generate the same trajectories.*

Proof. First, we define $J^\pi(f)$ as follows:

$$J^\pi(f) := \mathbb{E}_{\tau\sim\pi}[G_f(\tau)] \tag{4.3}$$

We give a proof by contradiction under 3 assumptions. The 3 assumptions are:

1. The reward function is non-negative.
2. The space of policies is finite.

Note that the first assumption is not restrictive since we can always shift a reward function by adding a sufficiently positive constant to the reward of all state-action pairs such that the new reward function is always non-negative. This new reward function is equivalent to the original one since it does not change the value ordering of policies.

We argue that the second assumption is reasonable in practice. Even if we consider what appears to be a continuous policy space because the parameterization of the policy space is continuous, in practice, parameters have a finite precision and therefore the space of policies is necessarily finite (albeit possibly very large). This means that the algorithm must terminate after enumerating all finitely many policies.

Based on assumption (ii), it should be clear that the alternation of the optimization procedures in Equation (4.1) and Equation (4.2) will converge to a fixed point. Since there are finitely many policies, and we add a new policy to Π at each iteration (until we reach the fixed point), in the worst case, the alternation will terminate once all policies have been added to Π.

When the algorithm converges to a fixed point, let π^* be the optimal policy found by policy optimization in Equation (4.1) and c^* be the optimal constraint function found by constraint adjustment in Equation (4.2). According to Equation (4.1), we know that $J^{\pi^*}(c^*) \leq \beta$. Similarly, we show that the objective in Equation (4.2) is less than or equal to β:

$$\max_{c} \min_{\pi \in \Pi} J^{\pi}(c) \text{ subject to } J^{\pi^E}(c) \leq \beta \tag{4.4}$$

$$= \min_{\pi \in \Pi} J^{\pi}(c^*) \qquad \text{since } c^* \text{ is the optimal solution of (4.2)} \tag{4.5}$$

$$\leq \beta \qquad \text{since } \pi^* \in \Pi \text{ and } J^{\pi^*}(c^*) \leq \beta \tag{4.6}$$

Next, if we assume that π^* is not equivalent to π^E, then we can show that the objective in Equation (4.2) is greater than β.

$$\max_{c} \min_{\pi \in \Pi} J^{\pi}(c) \text{ such that } J^{\pi^E}(c) \leq \beta \tag{4.7}$$

$$\geq \min_{\pi \in \Pi} J^{\pi}(\hat{c}) \tag{4.8}$$

$$\text{where we choose } \hat{c}(s,a) = \frac{r(s,a)\beta}{J^{\pi^E}(r)}$$

$$= \min_{\pi \in \Pi} J^{\pi}(r)\beta / J^{\pi^E}(r) \tag{4.9}$$

$$\text{by substituting } \hat{c} \text{ by its definition}$$

$$> J^{\pi^E}(r)\beta / J^{\pi^E}(r) = \beta \tag{4.10}$$

$$\text{since } J^{\pi}(r) \geq J^{\pi^E}(r)\ \forall \pi \in \Pi \text{ according to (2.13)}$$

$$\text{and } J^{\pi}(r) \neq J^{\pi^E}(r) \text{ since } \pi^* \text{ is not equivalent to } \pi^E \text{ (i.e., } J^{\pi^*}(r) \neq J^{\pi^E}(r))$$

$$\text{and } J^{\pi}(r) \geq J^{\pi^*}(r)\ \forall \pi \in \Pi$$

The key step in the above derivation is Equation (4.8), where we choose a specific constraint function $\hat{c}$. Intuitively, this constraint function is selected to be parallel to the reward function while making sure that $J^{\pi^E}(c) \leq \beta$. We know that all policies in Π achieve higher

expected cumulative rewards than π^E. This follows from the fact that the optimization procedure in Equation (2.13) finds an optimal policy π^* at each iteration and this optimal policy is not equivalent to π^E based on our earlier assumption. So the policies in Π must earn higher expected cumulative rewards than π^E otherwise π^E would have been chosen during policy optimization. So if we select a constraint function parallel to the reward function then the policies in Π will have constraint values greater than the constraint value of π^E and therefore they should be ruled out in the constrained policy optimization Equation (2.13).

Since the inequalities in Equation (4.4) - Equation (4.6) contradict the inequalities in Equation (4.7) - Equation (4.10), we conclude that π^* must be equivalent to π^E.

□

4.1.2 Practical algorithm

In practice, there are several challenges in implementing the optimization procedures proposed in Equation (4.2). First, we do not have the expert policy π^E , but rather trajectories generated from the expert policy. Also, the set Π of policies can grow to become very large before convergence is achieved. Furthermore, convergence may not occur depending on whether the parameterized policy space contains the expert policy. The optimization procedure in Equation (4.2) is a constrained max-min optimization, and a straightforward implementation of the same may not easily converge. Therefore, we approximate the theoretical procedure in Equation (4.2) with the practical procedure described in Equation (4.11).

ICL-Simplified-Constraint-Adjustment:

$$\phi^* := \arg\max_{\phi} \mathbb{E}_{\tau\sim\pi_{\text{mix}}}[G_{c_\phi}(\tau)] \text{ s.t. } \mathbb{E}_{\tau\sim\pi^E}[G_{c_\phi}(\tau)] \leq \beta \tag{4.11}$$

Maximizing the constraint values of a mixture of policies over Π (obtained as per the process outlined in Section 4.1.3) tends to increase the constraint values for all policies in Π most of the time, and when a policy's constraint value is not increased beyond β it will be a policy close to the expert policy. The different policies in π_{mix} are weighted by their dissimilarity to the expert behaviour. The practical algorithm, then alternates between Equation (4.1) and Equation (4.11).

4.1.3 Mixture policy weight computation

We can compute the mixture policy as outlined in Algorithm 3. Here, we provide a high level description of this process. More specifically, suppose we are given an expert dataset $\mathcal{D}$. We can then train a normalizing flow f on the state-action pairs in $\mathcal{D}$ (suppose there are $n_{\mathcal{D}}$ number of trajectories, and n_E number of total state-action pairs in the dataset). Then, the normalizing flow can be used to evaluate the negative log likelihood (NLL) $-\log p_f(s,a)$ of any test state-action pair (s,a).

For the expert policy, we can calculate the NLL mean and std. deviation as (here, p_f is the probability of a state-action pair according to the normalizing flow f),

$$\mu_E = \frac{1}{n_E} \sum_{\tau \in \mathcal{D}} \sum_{(s,a) \in \tau} -\log p_f(s,a) \tag{4.12}$$

$$\sigma_E = \sqrt{\frac{1}{n_E} \sum_{\tau \in \mathcal{D}} \sum_{(s,a) \in \tau} (-\log p_f(s,a) - \mu_E)^2} \tag{4.13}$$

The similarity of a state action pair w.r.t the dataset (i.e. does it represent behaviour captured by the dataset) can be computed by checking if the log likelihood is significant, i.e.

$$\text{Sim}(s,a;f) := \mathbf{I}(\log p_f(s,a) > -(\mu_E + \sigma_E)) = \mathbf{I}(-\log p_f(s,a) < \mu_E + \sigma_E) \tag{4.14}$$

We can then define the "dissimilarity" of a test trajectory τ (length n_τ) w.r.t. the expert as the average of individual dissimilarities,

$$w(\tau) := \frac{1}{n_\tau} \sum_{(s,a) \in \tau} (1 - \text{Sim}(s,a;f))$$

Given policies $\pi_{1:i}$ obtained in i rounds of policy optimization, where $\mathcal{D}_{\pi_j}$ is a dataset of $n_{\mathcal{D}}$ trajectories sampled using the policy π_j, we can construct the weights of the mixture policy as,

$$\tilde{w}_i \propto \frac{1}{n_{\mathcal{D}}} \sum_{\tau \in \mathcal{D}_{\pi_i}} w(\tau)$$

Then, a dataset from the mixture policy can be sampled according to weights $\tilde{w}_i$, post normalization (i.e. weights should sum to 1). This means that for this dataset, we can sample individual trajectories such that for every trajectory, the probability of that trajectory being sampled from policy π_i is $\tilde{w}_i / (\sum_j \tilde{w}_j)$.

4.2 Implementation

For our implementation, we use both the Lagrangian approach as well the penalty approach (described in Section 2.8). More specifically, we use both the approaches for the constrained policy optimization step (Equation (4.1)), using PPO Penalty (described a bit later) for the simpler environments and highly optimized PPO Lagrangian (provided by OpenAI [113]) for more complex environments. For the constraint adjustment procedure (Equation (4.11)), we find that the Lagrangian approach suffers from oscillatory behaviour (Appendix A.3), hence we use the simpler (to implement) penalty approach.

For the penalty approach based implementations, we use different λ for different procedures. Specifically, we perform constrained policy optimization (Equation (4.1)) using $\lambda = 0$, with the feasibility projection subprocedure, and constraint adjustment (Equation (4.11)) using a moderate/high λ and without the feasibility projection subprocedure. We set $\lambda = 0$ for constrained policy optimization because we found that the algorithm with a moderate/large λ was getting stuck in locally feasible regions and not finding better policies. Similarly, we found that an appropriately chosen moderate/large λ worked well for constraint adjustment.

4.2.1 Constrained policy optimization (PPO Penalty)

The constrained policy optimization implementation alternates between a feasibility projection step and a PPO update step (PPO is described in Section 2.7.2) as described below.

$$\textsc{Feasibility Projection:}\ \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_1 \nabla_{\boldsymbol{\theta}} \textsc{ReLU}(\mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}[G_{c_{\boldsymbol{\phi}}}(\tau)] - \beta) \tag{4.15}$$

$$\textsc{PPO Update:}\ \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_2 \nabla_{\boldsymbol{\theta}} \textsc{PPO-Loss}(\pi_{\boldsymbol{\theta}}) \tag{4.16}$$

The gradient for the feasibility projection step can be computed just like policy gradient:

$$\nabla_{\boldsymbol{\theta}} \textsc{ReLU}(\mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}[G_{c_{\boldsymbol{\phi}}}(\tau)] - \beta) = \mathbf{I}(\mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}[G^0_{c_{\boldsymbol{\phi}}}(\tau)] > \beta) \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}[G^0_{c_{\boldsymbol{\phi}}}(\tau)] \tag{4.17}$$

$$G^{t_0}_{c_{\boldsymbol{\phi}}}(\tau) := \sum_{t=t_0}^{\infty} \gamma^{t-t_0} c_{\boldsymbol{\phi}}(S_t, A_t) \tag{4.18}$$

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}[G^0_{c_{\boldsymbol{\phi}}}(\tau)] = \mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{\infty} G^t_{c_{\boldsymbol{\phi}}}(\tau) \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t) \right] \tag{4.19}$$

Note that the constrained policy optimization procedure can also be replaced by any other CRL implementation that finds an optimal constrained policy given a constraint.

4.2.2 Constraint adjustment

The constraint adjustment omits the feasibility projection step since λ is large. The update can be written as follows.

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \eta_3 \nabla_{\boldsymbol{\phi}} \left\{ -\mathbb{E}_{\tau \sim \pi_{\text{mix}}}[G_{c_{\boldsymbol{\phi}}}(\tau)] + \lambda \text{ReLU}(\mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[G_{c_{\boldsymbol{\phi}}}(\tau)] - \beta) \right\} \tag{4.20}$$

$$\nabla_{\boldsymbol{\phi}} \{ -\mathbb{E}_{\tau \sim \pi_{\text{mix}}}[G_{c_{\boldsymbol{\phi}}}(\tau)] \} = -\mathbb{E}_{\tau \sim \pi_{\text{mix}}}[\nabla_{\boldsymbol{\phi}} G^0_{c_{\boldsymbol{\phi}}}(\tau)] \tag{4.21}$$

$$\nabla_{\boldsymbol{\phi}} \text{ReLU}(\mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[G_{c_{\boldsymbol{\phi}}}(\tau)] - \beta) = \mathbf{I}(\mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[G^0_{c_{\boldsymbol{\phi}}}(\tau)] > \beta) \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[\nabla_{\boldsymbol{\phi}} G^0_{c_{\boldsymbol{\phi}}}(\tau)] \tag{4.22}$$

The gradient $\nabla_{\boldsymbol{\phi}} G^0_{c_{\boldsymbol{\phi}}}(\tau)$ can be easily computed through automatic differentiation as $G^0_{c_{\boldsymbol{\phi}}}(\tau)$ is the discounted sum of individual $c_{\boldsymbol{\phi}}$ terms.

4.2.3 Overall algorithm

The overall algorithm for ICL is provided in Algorithm 1. This is composed of two alternating procedures. The CRL (PPO Penalty) procedure is provided in Algorithm 2 and the constraint adjustment procedure is provided in Algorithm 3.

We now describe these algorithms in more detail.

Algorithm 1 first trains a normalizing flow on the expert dataset (lines 1-2). After initializing the constraint function (line 3), the algorithm repeats the following process for n overall iterations. It fixes the constraint function, and initializes and learns a policy using a CRL algorithm (lines 5-6). Then it uses the previously learned policies to adjust the constraint (line 7). The algorithm stops when either the number of iterations are exhausted or the last policy is close enough to the expert policy (line 8). Finally, it outputs the learned constraint function and the most recent policy.

In Algorithm 2, we assume that the constraint is given, and then perform constrained policy optimization which finds an optimal policy given the constraint function. The algorithm iterates between the feasibility projection step (line 2) and minimizing just the PPO loss (line 3). The feasibility projection step ensures that the constraint is satisfied before the

policy is updated. Minimizing the PPO loss updates the policy parameters so that it becomes more optimal.

Finally, in Algorithm 3, we adjust the constraint function assuming that the set of optimal policies found by Algorithm 2 are given. Lines 1-3 follow the process of finding the dissimilarity of a trajectory (for more description, see Section 4.1.3). Then, the algorithm constructs individual policy datasets (line 4) and a combined mixture policy dataset (line 6) by considering the policies in proportion to their weights. This is followed by repeatedly minimizing the soft loss (lines 9-10), which consists of two terms. The expectation w.r.t. the mixture policy is computed by reweighing the trajectories according to their dissimilarity (line 8), while the expectation w.r.t. the expert policy is computed by considering the cumulative costs for the trajectories in the expert dataset (line 9). At the end, the learned constraint function is output.

Algorithm 1 INVERSE-CONSTRAINT-LEARNING (ICL)

hyper-parameters: number of ICL iterations n, tolerance ϵ
input: expert dataset $\mathcal{D} = \{\tau\}_{\tau \in \mathcal{D}} := \{\{(s_t, a_t)\}_{1 \leq t \leq |\tau|}\}_{\tau \in \mathcal{D}}$
1: **initialize** normalizing flow f
2: **optimize** likelihood of f on expert state action data: $\max_f \textsc{Sum}_{(s,a) \in \tau, \tau \in \mathcal{D}}(\log p_f(s, a))$
3: **initialize** constraint function c (parameterized by $\boldsymbol{\phi}$)
4: **for** $1 \leq i \leq n$ **do**
5: **initialize** policy π_i
6: **perform** $\pi_i := \textsc{Constrained-RL}(\pi_i, c)$
7: **perform** $c := \textsc{Constraint-Adjustment}(\pi_{1:i}, c, \mathcal{D}, f)$
8: **break** if $\textsc{Normalized-Accrual-Dissimilarity}(\mathcal{D}, \mathcal{D}_{\pi_i}) \leq \epsilon$
▷ *See Section 4.3.4* for normalized accrual dissimilarity metric
9: **end for**
output: learned constraint function c (neural network with sigmoid output),
learned most recent policy π_i

4.3 Experiments

4.3.1 Experiment design

We wish to design experiments that answer the following questions:

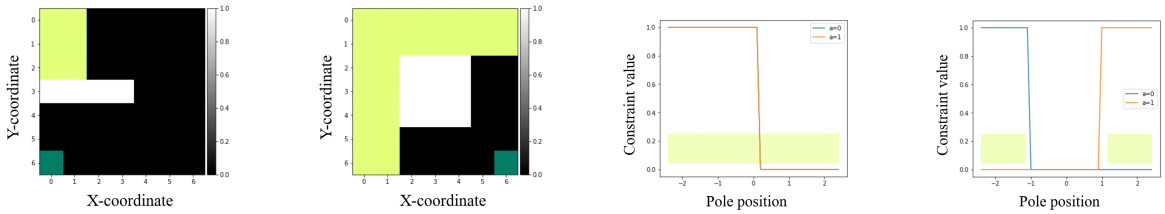
- Are the learned constraints similar to the true constraints?

Algorithm 2 PPO-Penalty

hyper-parameters: learning rates η_1, η_2, constrained RL epochs m
input: policy π_i parameterized by $\boldsymbol{\theta_i}$, constraint function c, constraint threshold β
for $1 \leq j \leq m$ **do**
  correct π_i to be feasible: (iterate) $\boldsymbol{\theta_i} \leftarrow \boldsymbol{\theta_i} - \eta_1 \nabla_{\boldsymbol{\theta_i}} \text{ReLU}(\mathbb{E}_{\tau \sim \pi_i}[G_c(\tau)] - \beta)$
  optimize expected discounted reward: $\boldsymbol{\theta_i} \leftarrow \boldsymbol{\theta_i} - \eta_2 \nabla_{\boldsymbol{\theta_i}} \text{PPO-Loss}(\pi_i)$
    ▷ *Proximal Policy Optimization* [119]
end for
output: learned policy π_i

Algorithm 3 Constraint-Adjustment

hyper-parameters: learning rate η_3, penalty wt. λ, constraint adjustment epochs e
input: policies $\pi_{1:i}$, constraint function c, trained normalizing flow f,
  expert dataset $\mathcal{D}_E = \{\tau\}_{\tau \in \mathcal{D}_E} := \{\{(s_t, a_t)\}_{1 \leq t \leq |\tau|}\}_{\tau \in \mathcal{D}_E}$,
  constraint threshold β
given: $c_\gamma(\tau) := \text{Sum}_{1 \leq t \leq |\tau|}(\gamma^{t-1} c(s_t, a_t))$,
  $\text{Sample}_\tau(\Pi, p)$ which generates $|\mathcal{D}|$ trajectories $\tau = \{(s_t, a_t)\}_{1 \leq t \leq |\tau|}$, where for each τ, we choose $\pi \in \Pi$ with prob. $p(\pi)$, then, $\tau \sim \pi$
$\mu_E := \text{Mean}_{(s,a) \in \tau, \tau \in \mathcal{D}}(-\log p_f(s, a))$
$\sigma_E := \text{Std-Dev}_{(s,a) \in \tau, \tau \in \mathcal{D}}(-\log p_f(s, a))$
$w(\tau) := \text{Mean}_{(s,a) \in \tau}(\mathbf{1}(-\log p_f(s, a) > \mu_E + \sigma_E))$ ▷ *trajectory dissimilarity w.r.t. expert*
construct policy dataset $\mathcal{D}_{\pi_i} = \text{Sample}_\tau(\Pi = \{\pi_i\}, p = \{1\})$
$\tilde{w}_i := \text{Mean}_{\tau \in \mathcal{D}_{\pi_i}} w(\tau)$ ▷ *unnormalized policy weights*
construct agent dataset $\mathcal{D}_A = \text{Sample}_\tau(\Pi = \pi_{1:i}, p(\pi_i) \propto \tilde{w}_i)$ ▷ *policy reweighting*
for $1 \leq j \leq e$ **do** ▷ *constraint function adjustment*
  compute $\mathbb{E}_{\tau \sim \pi_{\text{mix}}}[G_c(\tau)] := \text{SUM}_{\tau \in \mathcal{D}_A} \frac{w(\tau) c_\gamma(\tau)}{\text{SUM}_{\tau \in \mathcal{D}_A} w(\tau)}$ ▷ *trajectory reweighting*
  compute $\mathbb{E}_{\tau \sim \pi^E}[G_c(\tau)] := \text{Mean}_{\tau \in \mathcal{D}_E}(c_\gamma(\tau))$
  compute soft loss $L_{\text{soft}}(c) := -\mathbb{E}_{\tau \sim \pi_{\text{mix}}}[G_c(\tau)] + \lambda \text{ReLU}(\mathbb{E}_{\tau \sim \pi^E}[G_c(\tau)] - \beta)$
  optimize constraint function c: $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \eta_3 \nabla_{\boldsymbol{\phi}} L_{\text{soft}}(c)$
    ▷ c is parameterized by $\boldsymbol{\phi}$
end for
output: constraint function c



Gridworld (A) Gridworld (B) CartPole (MR) CartPole (Mid)

Figure 4.2: Gridworld and CartPole environment setup. For the Gridworld environments, white regions indicate constraint value of 1, green/yellow regions correspond to start states, and dark green region is the goal state. For the Cartpole environments, start region is in yellow, true constraint value is plotted on Y-axis and X-coordinate of the pole is plotted on the X-axis.

- Does the learned policy accrual match the expert policy accrual?
- Can the method learn complex real world constraints, for example, from autonomous driving?
- Can the method handle stochastic environment dynamics?
- Do we really need to use the normalizing flow in computing trajectory and policy dissimilarities?

We answer these questions through experiments with the following environments.

4.3.2 Environments

We now provide a description of the environments used in our experiments.

Gridworld environments

The layout of the Gridworld environments are provided in Figure 4.2. In these 7x7 gridworld environments, the agent starts in the start region and must navigate to the goal. However, without the constraint, the agent can directly go downward or diagonally towards the goal, which leads to a different policy than if the constraint is specified. When the constraint is specified, the agent must avoid the high constraint regions and then go to the

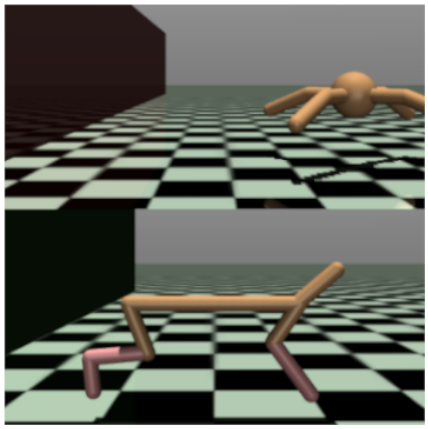


Figure 4.3: Mujoco environments, Ant-Constrained (above) and HalfCheetah-Constrained (below). Source: [88].

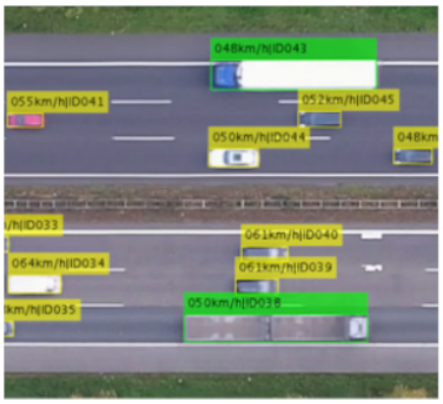


Figure 4.4: HighD environment, based on a real-world highway driving dataset [63].

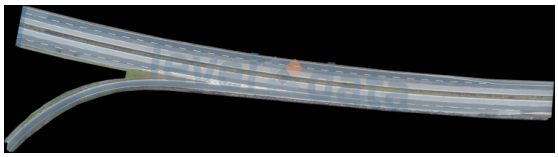
goal region. The state space consists of two features, i.e. x and y positions, overall assuming 7x7=49 values. The action space consists of 8 discrete actions including 4 nominal directions and 4 diagonal directions.

CartPole environments

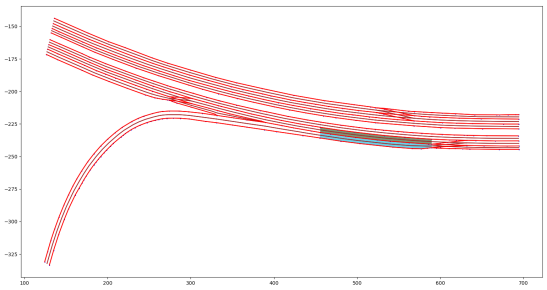
The setup of the CartPole environments are provided in Figure 4.2. More specifically, in this environment, the agent (balancing the pole in the standard CartPole problem) starts in the yellow region and must navigate to keep the constraint satisfied, depending on the constraint specification (expected or probabilistic). All variants of the environment may start in a region of high constraint value (yellow region can correspond to high cost or low cost), and the objective is to move to a region of low constraint value and balance the pole there, while the constraint function is being learned. *Note* that there are two actions in this environment, i.e. $a = 0$ (left) and $a = 1$ (right). The state and action spaces are the same as the standard CartPole environment [18].

Mujoco environments

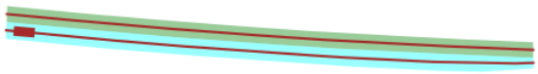
Ant-Constrained and HalfCheetah-Constrained environments (Figure 4.3) are Mujoco robotics environments used in Malik et al. [88]. In our experiments, we use the $z \geq -1$ constraint



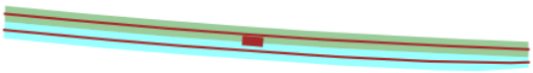
Real world map for the shown scenario.



Start (cyan) and target (green) lanes for the lane change in this scenario.



Simplified environment (agent starts from leftmost point)



Simplified environment (agent moves according to chosen action)

Figure 4.5: ExiD dataset lane change environment (one of multiple scenarios). The state space contains the signed distance to the target lane, and the action space contains the lateral velocity. There are no other vehicles, only the ego car. The objective is to find a constraint on the lateral velocity (action) that an agent must maintain for any certain signed distance from the target lane.

(i.e. the agent should stay in region $z \geq -1$) rather than $z \geq -3$ constraint used in [88] to make the setting challenging, similar to [42]. The state and action spaces are the same as the standard Ant and HalfCheetah Mujoco environments [130].

HighD environment (car following)

For this environment (Figure 4.4), we use the real world highway dataset simulator (HighD) from Gaurav et al. [42], which in turn, is based on dataset and environments by Krajewski et al. [63] and Lee et al. [65]. More specifically, the agent starts on a straight road on the left side of the highway, and the goal is to reach the right side of the highway without any longitudinal collisions. The state space is comprised of 7 ego features and 7 other features (predicates, propositions and processed features). The action space consists of a two continuous actions, i.e., acceleration and rate of change of steering angle. Also, note that in this environment, the true constraint function is not known. Instead, we want to

learn a constraint function that explains the relation between the ego velocity and the gap to the car in the front.

ExiD environment (lane change)

Similar to the HighD environment, we construct an environment (Figure 4.5) using $\approx$ 1000 trajectories of varying lengths ($\leq$ 1000) from 5 different scenarios of ExiD highway driving dataset [95]. To do so, we select trajectories where ego performs a lane change such that there are no other vehicles around the ego. On every reset, the environment initializes to the starting location of one of the $\approx$ 1000 trajectories. The goal is to perform a lane change to reach the target lane, which can be either to the left or right. The state consists of the signed distance (m) to the target lane, the distance to the lead vehicle (m), the distance to the rear vehicle (m) and whether or not there is an overlap in the lateral dimension (boolean). The action is a continuous lateral velocity (ms^{-1}) that can be directly controlled by the agent. The last 3 state features are fixed to predefined values during constraint visualization. For this environment, we wish to learn a constraint that shows the relationship between the signed distance to the target lane and the lateral velocity action.

4.3.3 Methods and baselines

We conduct experiments to compare the performances of three algorithms:

1. **ICL** with threshold β
2. **GAIL-Constraint**: Generative adversarial imitation learning [53] is an imitation learning method that can be used to learn a policy that mimics the expert policy. The discriminator can be considered as a local reward function that incentivizes the agent to mimic the expert. We assume that the agent is maximizing the reward $r(s,a) := r_0(s,a) + \log(1 - c(s,a))$ where r_0 is the given true reward, and then the log term corresponds to the GAIL's discriminator. Note that when $c(s,a) = 0$, the discriminator reward is 0, and when $c(s,a) = 1$, the discriminator reward tends to $-\infty$.
3. **Inverse constrained reinforcement learning i.e. ICRL** [88], which is another method that can learn arbitrary neural network constraints[1]. The algorithm simula-

[1]different from ICRL defined in Section 2.5.2

taneously performs CRL and constraint adjustment. It formulates the problem from a maximum entropy perspective, however this can only handle hard constraints.

4.3.4 Metrics

We define and report two metrics for our ICL experiments:

1. **Constraint Mean Squared Error i.e. CMSE**, which is the mean squared error between the true constraint and the recovered constraint on a uniformly discretized state-action space for the respective environment. The details of this discretization are provided in Appendix B.3. If the discrete set of state-action pairs is $\mathcal{D}$, then

$$\text{CMSE}(c, c_{true}) = \frac{1}{|\mathcal{D}|} \sum_{(s,a)\in\mathcal{D}} (c(s,a) - c_{true}(s,a))^2 \tag{4.23}$$

2. **Normalized Accrual Dissimilarity i.e. NAD**, which is a dissimilarity measure computed between the expert and agent accruals. This is the practical implementation of distance function described in Section 2.1.3. This metric is computed as follows. Given a policy learned by the method, we compute an agent dataset of trajectories. Then, the accrual (state-action visitation frequency) is computed for both the agent dataset and the expert dataset over a uniformly discretized state-action space, which is the same as the one used for CMSE. Finally, the accruals are normalized, and the Wasserstein distance is computed between the accruals. The pseudocode is provided in Algorithm 4.

Here, using only the CMSE metric is problematic since two equivalent constraints can have arbitrarily large CMSE[2]. Thus, we also consider the NAD metric.

4.3.5 Training setup

We perform 5 training runs for all the methods. For constraint learning, we use a constraint neural network with a sigmoid output[3], so that we can also interpret the outputs as safeness

[2]Suppose we consider constraints c and αc which are equivalent (as shown in Section 2.9.1). Then, by setting α to a relatively large positive value, we can have a large CMSE even though the constraints are equivalent.

[3]The complete architecture includes an input layer, 2 hidden layers of 64 nodes each, and an output layer with a sigmoid output. The intermediate activations are ReLU.

Algorithm 4 NORMALIZED-ACCRUAL-DISSIMILARITY (NAD)

hyper-parameters: discrete state-action bins $B = \{(s_0^t \leq s \leq s_1^t, a_0^t \leq a \leq a_1^t)\}_t$
input: datasets $\mathcal{D}_1 \sim \pi_1, \mathcal{D}_2 \sim \pi_2$, $|\mathcal{D}_1| = |\mathcal{D}_2| = n$
initialize bin counters B_1, B_2
for $1 \leq i \leq n$ **do**
    $(s, a) = i$-th state-action pair in $\mathcal{D}_1$
    $(s', a') = i$-th state-action pair in $\mathcal{D}_2$
    $B_1[t]$ += 1 if (s, a) lies in t-th bin of B_1
    $B_2[t]$ += 1 if (s', a') lies in t-th bin of B_2
end for
normalize B_1 and B_2
compute $d :=$ WASSERSTEIN-DISTANCE(B_1, B_2)
output: d

values[4]. Such a network is able to capture the true constraints arbitrarily closely.

Table 4.1: Choice of β

Environment	β
Gridworld (A, B)	0.99
CartPole (MR, Mid)	20
Ant-Constrained	15
HalfCheetah-Constrained	15
HighD	0.1
ExiD	5

To reduce training time, we use ICL with (highly optimized) PPO Lagrange [113] instead of ICL with PPO Penalty for complex environments (Mujoco environments and ExiD environment). In principle, any procedure that can efficiently optimize Equation (2.13) can be used for the policy optimization part of ICL (Equation (4.1)).

[4] A constraint value of 0 indicates complete feasibility or safe state, while a constraint value of 1 indicates infeasibility or unsafe state.

Table 4.2: Constraint Mean Squared Error for ICL synthetic experiments (lower is better)

Environment	GAIL-Constraint	ICRL	ICL
Gridworld (A)	0.31 ± 0.01	0.11 ± 0.02	**0.08 ± 0.01**
Gridworld (B)	0.25 ± 0.01	0.21 ± 0.04	**0.04 ± 0.01**
Cartpole (MR)	0.12 ± 0.03	0.21 ± 0.16	**0.02 ± 0.00**
Cartpole (Mid)	0.25 ± 0.02	0.27 ± 0.03	**0.08 ± 0.05**

4.3.6 Results

We compare the performance of our proposed method (ICL) w.r.t. baselines, i.e. GAIL-Constraint [53] and ICRL [88].

Constraint error

We provide the CMSE results for the synthetic environments in Table 4.2 (lower values are better). We also provide the constraint function plots for Gridworld (A) (Figure 4.6), Gridworld (B) (Figure 4.7), CartPole (MR) (Figure 4.8) and CartPole (Mid) (Figure 4.9) environments. Note that in the constraint function plots, when the constraint value is 1, it can be interpreted as an infeasible state or state-action pair (i.e. a safeness violation) that makes the constraint satisfaction less likely.

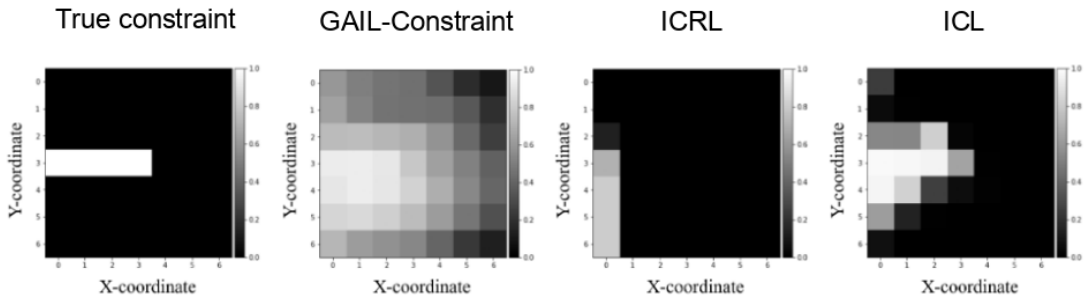


Figure 4.6: Average constraint function value (averaged across 5 seeds) for Gridworld (A) environment.

While our method is not guaranteed to produce the true constraint function (due to constraint unidentifiability, Section 2.9.1), empirically, we find that our method is still able to learn constraint functions that strongly resemble the true constraint function, as can be

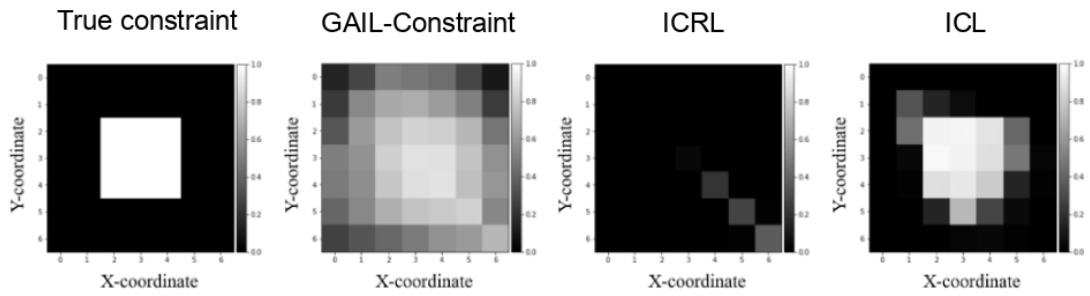


Figure 4.7: Average constraint function value (averaged across 5 seeds) for Gridworld (B) environment.

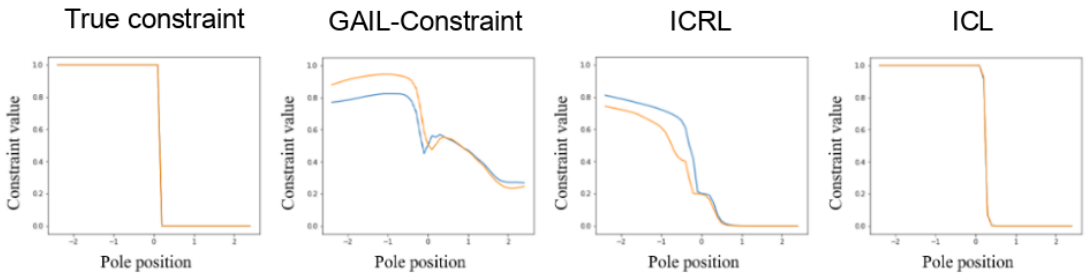


Figure 4.8: Average constraint function value (averaged across 5 seeds) for CartPole (MR) environment.

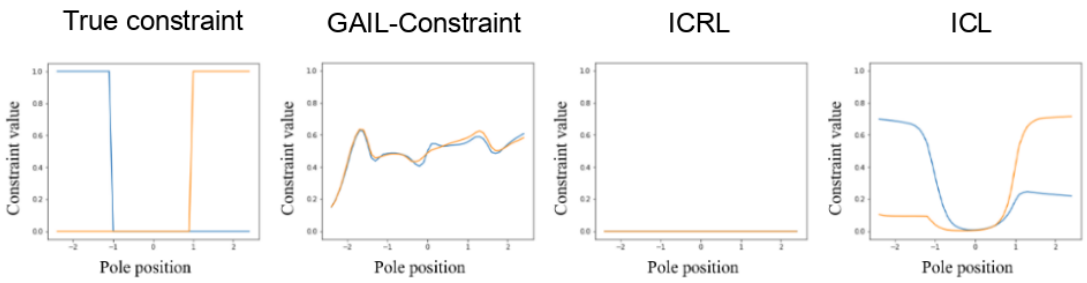


Figure 4.9: Average constraint function value (averaged across 5 seeds) for CartPole (Mid) environment.

Table 4.3: Normalized Accrual Dissimilarity for ICL synthetic experiments (lower is better)

Environment	GAIL-Constraint	ICRL	ICL
Gridworld (A)	1.76 ± 0.25	1.73 ± 0.47	**0.36 ± 0.10**
Gridworld (B)	1.29 ± 0.07	2.15 ± 0.92	**1.26 ± 0.62**
Cartpole (MR)	1.80 ± 0.24	12.32 ± 0.48	**1.63 ± 0.89**
Cartpole (Mid)	7.23 ± 3.88	13.21 ± 1.81	**3.04 ± 1.93**

seen by our low CMSE scores (Table 4.2). Other methods, e.g., GAIL-Constraint can find the correct constraint function for all environments except CartPole (Mid) (Figure 4.9), however, the recovered constraint is more diffused throughout the state action space (see Figure 4.6 and Figure 4.7). In contrast, our recovered constraint is pretty sharp, even without a regularizer.

From the training plots, we also find that the ICRL method is able to find the correct constraint function only for CartPole (MR) (Figure 4.8) and to a less acceptable degree for Gridworld (A) (Figure 4.6). This is surprising as ICRL should be able to theoretically learn any arbitrary constraint function (note that we used the settings in [88] except for common hyperparameters), and expect it to perform better than GAIL-Constraint. Our justification for this is two-fold. One, the authors of ICRL have only demonstrated their approach with simple single-proposition constraints, and for more complex settings, ICRL may not be able to perform as well. Second, ICRL may require more careful hyperparameter tuning for each constraint function setting, even with the same environment, depending on the constraint. On the other hand, our method is able to learn these relatively complex constraints satisfactorily with relatively fewer hyperparameters.

Accrual dissimilarity

We provide the NAD results for the synthetic environments in Table 4.2 (lower values are better). We also provide the normalized accrual plots for Gridworld (A) (Figure 4.10), Gridworld (B) (Figure 4.11), CartPole (MR) (Figure 4.12) and CartPole (Mid) (Figure 4.13) environments.

We find a strong resemblance between the accruals recovered by our method and the expert, as can be seen by our low NAD scores (Table 4.3). In these scores, there is some variability (e.g., in Cartpole (MR) and Gridworld (B) environments, plots provided in Figure 4.12 and Figure 4.11) which is expected, since the policies are stochastic. For baselines, GAIL-Constraint accruals are similar to the expert accruals except for CartPole (Mid) en-

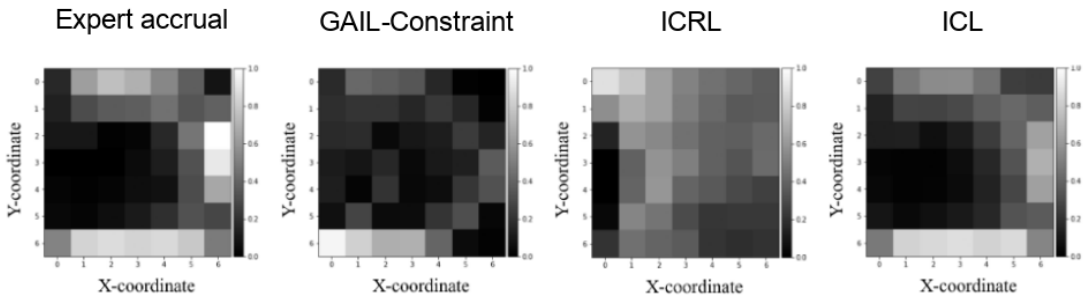


Figure 4.10: Average normalized accruals (averaged across 5 seeds) for Gridworld (A) environment.

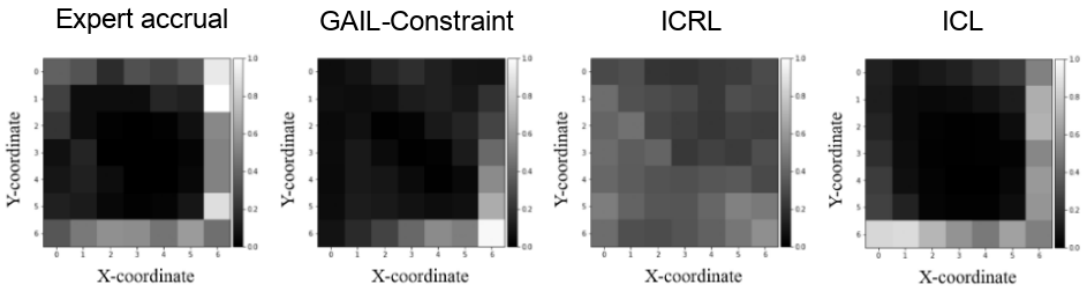


Figure 4.11: Average normalized accruals (averaged across 5 seeds) for Gridworld (B) environment.

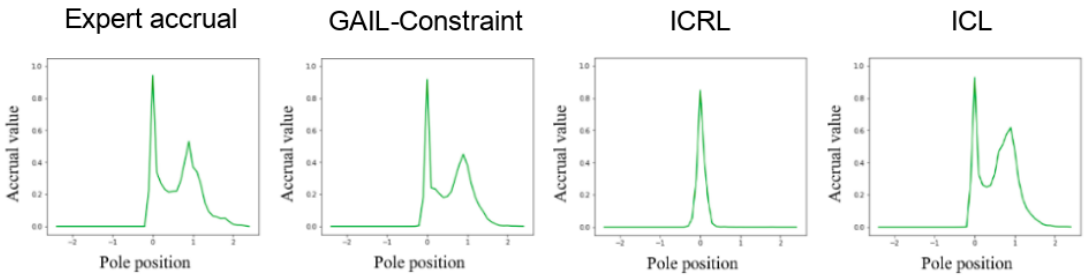


Figure 4.12: Average normalized accruals (averaged across 5 seeds) for CartPole (MR) environment.

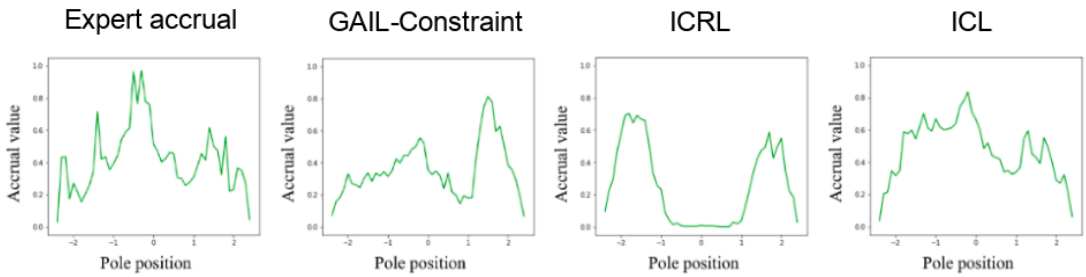


Figure 4.13: Average normalized accruals (averaged across 5 seeds) for CartPole (Mid) environment.

Table 4.4: Constraint Mean Squared Error for ICL Mujoco experiments (lower is better)

Environment	GAIL-Constraint	ICRL	ICL
Ant-Constrained	0.17 ± 0.04	0.41 ± 0.00	**0.07 ± 0.00**
HalfCheetah-Constrained	0.20 ± 0.03	0.35 ± 0.17	**0.05 ± 0.00**

vironment (Figure 4.13), where it is also unable to learn the correct constraint function. Overall, this indicates that GAIL is able to correctly imitate the constrained expert across most environments, as one would expect. On the other hand, ICRL accruals are even worse than GAIL, indicating that it is unable to satisfactorily imitate the constrained expert, even on the environments for which it is able to generate a somewhat satisfactory constraint function.

Mujoco results

We provide the CMSE results for the Mujoco experiments in Table 4.4 (lower values are better). We also provide the constraint function plots for Ant-Constrained (Figure 4.14) and HalfCheetah-Constrained (Figure 4.16) environments. Similarly, we provide the NAD results for the Mujoco experiments in Table 4.2 (lower values are better). Then, we provide the normalized accrual plots for Ant-Constrained (Figure 4.15) and HalfCheetah-Constrained (Figure 4.17) environments.

For the Mujoco environments, we observe that ICL is able to find the constraint more accurately compared to GAIL-Constraint and ICRL, as can be seen from the low CMSE scores in Table 4.4. With respect to the accruals, our method achieves the lowest NAD score for Ant-Constrained environment (Table 4.5). However, the NAD score achieved by ICL

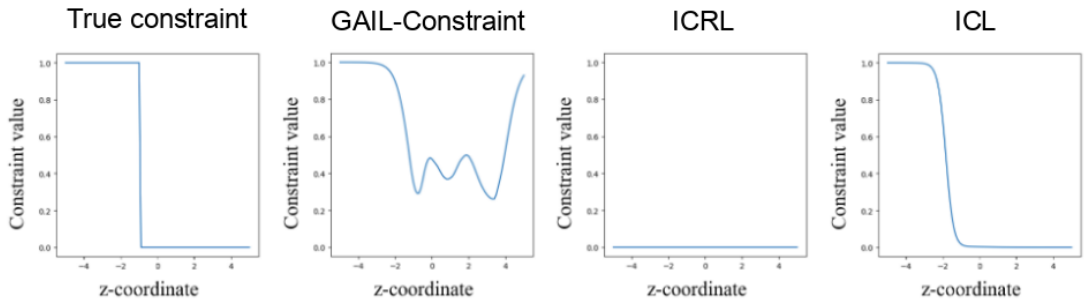


Figure 4.14: Average constraint function value (averaged across 5 seeds) for Ant-Constrained environment.

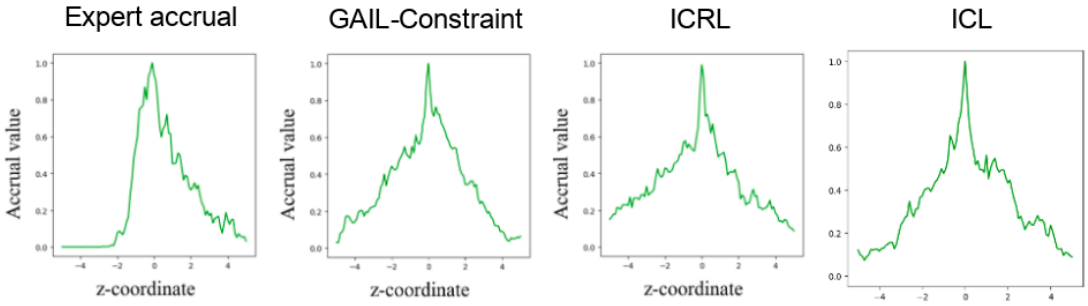


Figure 4.15: Average normalized accruals (averaged across 5 seeds) for Ant-Constrained environment.

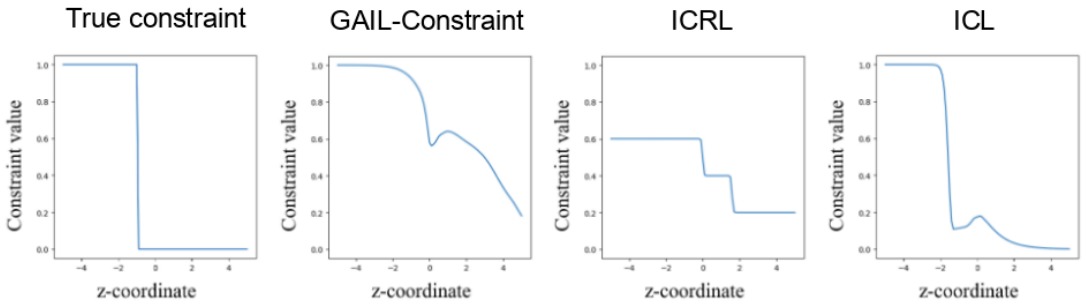


Figure 4.16: Average constraint function value (averaged across 5 seeds) for HalfCheetah-Constrained environment.

Table 4.5: Normalized Accrual Dissimilarity for ICL Mujoco experiments (lower is better)

Environment	GAIL-Constraint	ICRL	ICL
Ant-Constrained	8.02 ± 2.84	9.50 ± 2.84	**6.84 ± 1.29**
HalfCheetah-Constrained	14.38 ± 2.36	**7.50 ± 4.97**	10.16 ± 7.49

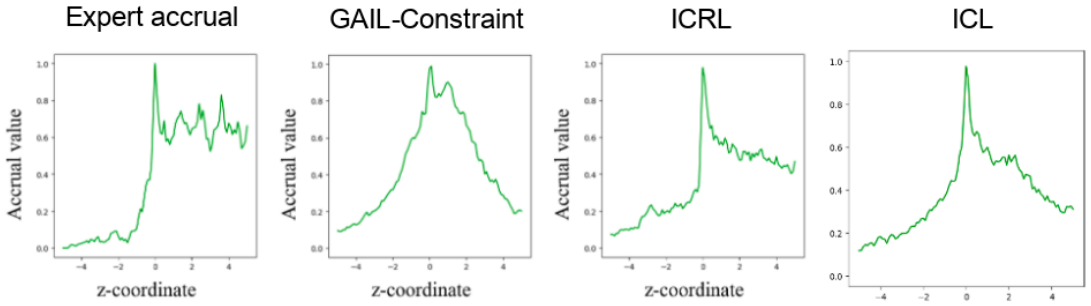


Figure 4.17: Average normalized accruals (averaged across 5 seeds) for HalfCheetah-Constrained environment.

is not the lowest for HalfCheetah-Constrained environment, and instead, ICRL achieves a lower score. This can be explained by visually inspecting the accruals for ICL and ICRL for the HalfCheetah-Constrained environment (Figure 4.17). More precisely, ICRL is able to find a better constrained optimal policy for this specific environment, compared to ICL. However, we do have a better constraint function (lower CMSE), and thus a constrained RL procedure that could find the optimal constrained policy in this specific setting could alleviate this issue.

We also note that the accruals don't seem to go much further to the right. Running the CRL procedure for a longer period of time will produce a better expert or agent policy that goes further right.

HighD results

The recovered constraint for the HighD environment is provided in Figure 4.18. The accruals are provided in Figure 4.19.

Overall, all the methods are able to find a constraint function that shows that as the agent's velocity increases, the center-to-center distance that must be maintained between the cars also must increase. The time gap that must be maintained can be calculated by dividing

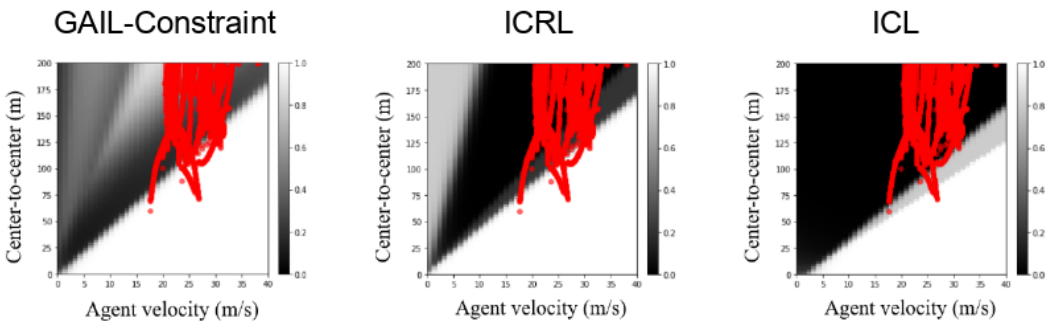


Figure 4.18: Recovered expected constraint function for the HighD environment (averaged over 5 seeds)

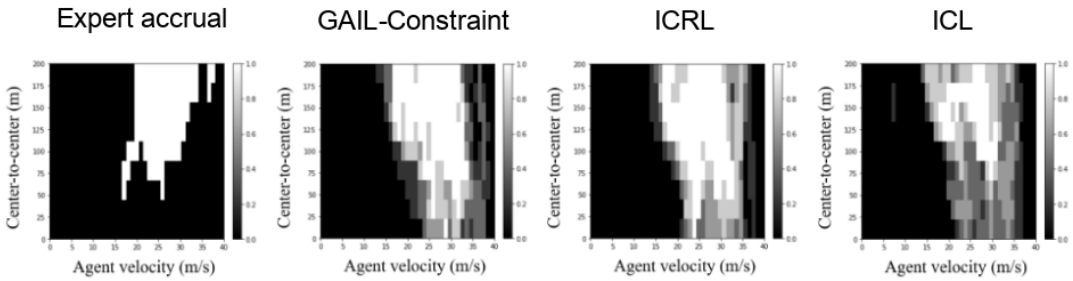


Figure 4.19: Accruals for the HighD environment ($\beta = 0.1$)

the requisite center-to-center distance by the agent's velocity. For all the methods, this time gap is approximately between 3-4.5 seconds.

More precisely, at $v = 20\text{ms}^{-1} = 72\text{kmh}^{-1}$, the gap is roughly between 60-80 m (equivalently, 3-4 seconds) for all other methods, while close to $v = 40\text{ms}^{-1} = 144\text{kmh}^{-1}$, the gap is between 125-150 m (equivalently ≈3-4 seconds) for our method, and 160-180 m (equivalently 4-4.5 seconds) for GAIL-Constraint and ICRL. Note that our method is the only one which doesn't assign high constraint value to large center-to-center gaps (no white areas in the black regions). The possible justification for this is that the other methods are unable to explicitly ensure that expert trajectories are assigned low constraint value, while our method is able to do so through the constraint adjustment step. Moreover, our method has fewer accrual points (red) in the white (high constraint value) region compared to the baselines.

ExiD results

The recovered constraint for the ExiD environment is provided in Figure 4.20. Similarly, the normalized accrual plot for the ExiD environment (Figure 4.21) is also provided.

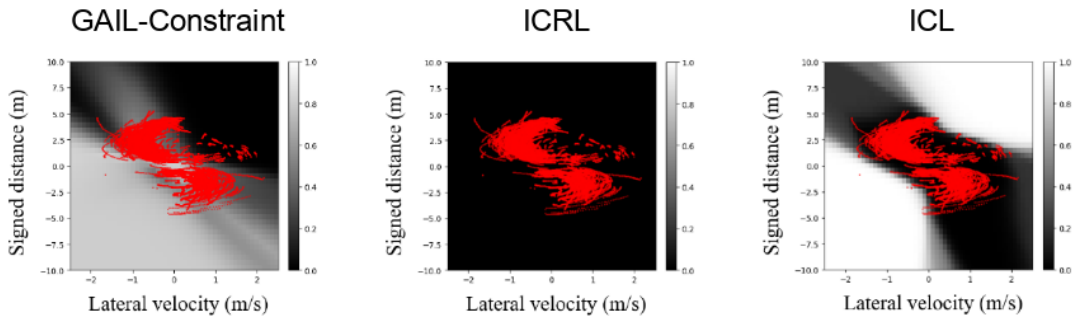


Figure 4.20: Recovered expected constraint function for the ExiD environment (averaged over 5 seeds)

For the ExiD environment, we find that only ICL is able to find a satisfactory constraint function for this setting (Figure 4.20). GAIL-Constraint finds a diffused constraint that overlaps the expert state-action pairs (and hence disallows them), while ICRL is unable to find a reasonable constraint. We can interpret the constraint plot of ICL as follows. Since the top-right and bottom-left quadrants ($v \geq 0, d \geq 0$ and $v \leq 0, d \leq 0$) are high in constraint value (white), they are relatively unsafe/prohibited. This is in line with expectation, since depending on the sign of the distance to the target lane, movement in

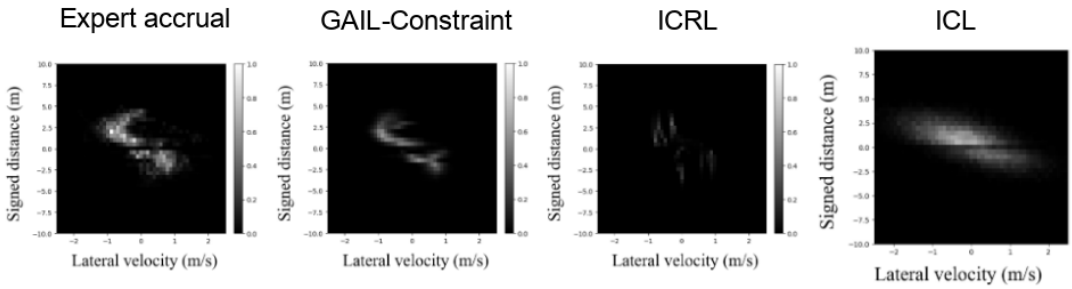


Figure 4.21: Accruals for the ExiD environment ($\beta = 5$)

one direction should be allowed and other should be disallowed (some risky behaviour is permitted since this is a soft constraint). More specifically, to reduce the magnitude of d, we must apply a control input v of opposite sign. Further, the shape of the constraint is also not exactly square/rectangular. That is, it constrains movement to be in one direction when far away (e.g., for $d = 10$, $v \leq -1$), and allows movement in both directions as we get closer to the target lane (e.g., for $d = 2.5$, any v is allowed). Note that the learned constraint is actually 5-dimensional where 3 features capture the traffic density information, 1 feature captures the agent vehicle's information and 1 feature reflects the action that is taken. For the purposes of visualization, we fix the traffic density and show the corresponding 2-dimensional plot.

This experiment highlights the ability of the proposed method in learning a time varying constraint. For this environment, the traffic density changes at each time step. The proposed method is able to learn a 5-dimensional constraint that when given the processed features of the traffic, produces information about the feasible actions that can be taken at that time step.

4.4 Other questions

4.4.1 Use of penalty method vs. Lagrangian

As mentioned in Section 2.8, Lagrangian formulations are quite popular in the literature, however, they are inherently min-max problems that admit gradient ascent-descent solutions which are prone to oscillatory behavior, as we observe in Appendix A.3. Thus, we prefer the simpler framework of penalty method optimization which works well empirically.

4.4.2 Stochastic dynamics

Our algorithm can also be applied to stochastic environments, which we demonstrate in Appendix A.2 with slippery Gridworld environments. Stochasticity would lead to noisier demonstrations, and we find that a little noise in the demonstrations helps in the quality of the recovered constraint, but too much noise can hamper the constraint learning.

4.4.3 Quality of learned policy w.r.t. imitation learning

We reiterate that our objective is to learn constraints, while the goal of imitation learning techniques (e.g., behavior cloning) is to directly learn a policy that mimics the expert without inferring any reward or constraint function. Even then, once the constraints have been learned, they can be used with the given reward to learn a constrained policy. Depending on factors like quality of expert data etc., ICL can learn a better or worse policy compared to an imitation learning method. We do not have definitive evidence to conclude that either is true.

4.4.4 Entropy regularized approaches and overfitting

Maximum entropy techniques (e.g., [120, 88]) have been previously proposed in the literature. Entropy maximization can indeed help in reducing overfitting, but they require a completely different mathematical setup. Our method doesn't belong to this family of approaches, but we hope to incorporate these strategies in the future to reap the mentioned benefits. We still note that empirically, our approach doesn't seem to suffer from any overfitting issues.

4.4.5 Use of normalizing flow and reweighting

We justify our choice of using a normalizing flow to reweigh policies and trajectories (Algorithm 3) in Appendix A.1. We find that ICL with reweighting performs better than ICL without reweighting, although the improvement is minor.

4.5 Summary

In this chapter, we provided a method to learn an expected constraint from expert demonstrations. To do so, we proposed alternating between two procedures, a constrained policy optimization procedure and a constraint adjustment procedure, until convergence. Later, we conducted experiments to validate the proposed technique (ICL) w.r.t. two baselines, GAIL-Constraint [53] and ICRL [88]. Our results show that compared to the baselines, ICL shows superior performance, and can recover a sharp and accurate constraint that looks like the true constraint. Moreover, it learns a policy similar to the expert's policy, in most cases. We also demonstrate the applicability of ICL to complex real world autonomous driving scenarios, and interpret the recovered constraint functions. Finally, we showcase the efficacy of our approach w.r.t. stochastic environments and provide rationale for using a normalizing flow to measure dissimilarity (see Appendix A.2 and Appendix A.1).

Chapter 5

Inverse probabilistic constraint learning

In this chapter, we will propose methods to learn a probabilistic constraint from expert demonstrations. We will first explain the difference between an expected constraint learned in the previous chapter and a probabilistic constraint that we will learn with the methods proposed in this chapter.

We can define a probabilistic constraint as follows,

$$\mathbb{P}_{\tau\sim\pi}(G_c(\tau) \leq \beta) \geq \delta \tag{5.1}$$

That is, given a set of trajectories, the constraint is satisfied for at least $\delta\%$ of the trajectories, for $0 \leq \delta \leq 1$ (G is defined in Equation (2.4)). Compared to an expected constraint which allows for some violations, such a probabilistic constraint allows violations at most $100(1-\delta)\%$ of the time. This can be seen by rewriting the probabilistic constraint as,

$$\mathbb{P}_{\tau\sim\pi}(G_c(\tau) > \beta) \leq 1 - \delta = 100(1-\delta)\% \tag{5.2}$$

Note that this means that if $\delta = 1$, then no violations are allowed, and in such cases, the probabilistic constraint becomes a hard constraint.

We now describe the approaches that can learn a probabilistic constraint from constrained expert demonstrations.

5.1 Framework

In order to learn a probabilistic constraint, our objective is to minimize the dissimilarity between the expert dataset and the agent dataset, where the expert dataset is generated by the expert policy and the agent dataset is generated by the policy learned through an algorithm for the IPCL setting (Equation (2.27)):

$$\arg\min_c d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{PCRL}(r, \{c\}; \Gamma_{\beta,\delta}))$$

Again, similar to ICL, we do this minimization empirically by alternating between policy optimization and constraint adjustment until this distance is below a threshold (described later in Algorithm 5).

5.1.1 Probabilistic constraint implied by an expected constraint

Since expected constraints have already been extensively studied [4] (and there exist methods which can learn such constraints from expert data), we first ask whether it is possible to derive a probabilistic constraint (Equation (2.15)) that is implied by an expected constraint (Equation (2.13)), which would enable the reuse of inverse constraint learning techniques designed for expected constraints, presented in the previous section.

The answer is *yes*. Leveraging Chebychev's bound, the following theorem shows that an expected constraint with threshold $\beta(1-\delta)$ implies a probabilistic constraint with thresholds β, δ.

Theorem 2. *Given any non-negative function $f : \Gamma \to [0, \infty)$ (where Γ is the space of trajectories), satisfying an expected constraint with threshold $\beta(1-\delta)$ is a sufficient condition for satisfying the probabilistic version of this constraint with thresholds β, δ.*

$$\mathbb{E}_{\tau\sim\pi}[f(\tau)] \leq \beta(1-\delta) \implies \mathbb{P}_{\tau\sim\pi}(f(\tau) \leq \beta) \geq \delta$$

Proof. The probabilistic constraint that must be satisfied is:

$$\begin{aligned} \mathbb{P}_{\tau\sim\pi}(f(\tau) \leq \beta) &\geq \delta \\ 1 - \mathbb{P}_{\tau\sim\pi}(f(\tau) > \beta) &\geq \delta \\ (1-\delta) - \mathbb{P}_{\tau\sim\pi}(f(\tau) > \beta) &\geq 0 \end{aligned} \tag{5.3}$$

Using Chebychev's inequality,

$$\mathbb{P}_{\tau\sim\pi}(f(\tau) \geq \beta) \leq \frac{\mathbb{E}_{\tau\sim\pi}[f(\tau)]}{\beta} \tag{5.4}$$

This means that (ignoring the equality on the left side without loss of generality),

$$(1-\delta) - \mathbb{P}_{\tau\sim\pi}(f(\tau) > \beta) \geq (1-\delta) - \frac{1}{\beta}\mathbb{E}_{\tau\sim\pi}[f(\tau)]$$

Since the right side is a lower bound of the left side, in order to satisfy (5.3), it is sufficient to satisfy right side ≥ 0, which gives us:

$$(1-\delta) - \frac{1}{\beta}\mathbb{E}_{\tau\sim\pi}[f(\tau)] \geq 0 \tag{5.5}$$

This is the same as the expected constraint with threshold $\beta(1-\delta)$, which completes the proof.

□

When choosing $f(\tau) := G_{c_\phi}(\tau)$ to correspond to the cumulative constraint function then $f(\tau) \geq 0$ as long as $c_\phi(s,a) \geq 0\ \forall s,a$.[1] Hence, we can use the ICL algorithm (Section 4.1.1) to learn an expected constraint $\mathbb{E}_{\tau\sim\pi}[f(\tau)] \leq \beta(1-\delta)$ that indirectly ensures satisfaction of the probabilistic constraint $\mathbb{P}_{\tau\sim\pi}(f(\tau) \leq \beta) \geq \delta$.

This gets us to an algorithm by which we can utilize any ICL algorithm to learn a probabilistic constraint, by altering β as specified in Theorem 2. However, as our empirical results indicate, it might not be always possible to learn a suitable probabilistic constraint this way. This can happen because reducing the β value can make learning difficult as $\beta \to 0$.

To show this, we use the setup of a Gridworld-O[2] environment, described in Figure 5.1. Using this setup, we conduct experiments (5 training runs each) with IPCL[3] with $\beta =$

[1] Since we can always rescale c_ϕ and β to be positive by adding a suitably large positive constant that does not change the nature of the constraint, there is no loss of generality in assuming that $c_\phi(s,a) \geq 0\ \forall s,a$.

[2] O for overlap

[3] IPCL (*Inverse Probabilistic Constraint Learning*) is the method proposed in Section 5.1.2 to learn a probabilistic constraint from expert demonstrations. Note that both the approach and the setting are called IPCL for simplicity. Since IPCL doesn't need to deal with a low β, it can successfully learn a probabilistic constraint compared to ICL with $\beta := \beta(1-\delta)$. Here, we run ICL with a constraint threshold value β which is defined to be $\beta(1-\delta)$ as per Theorem 2.

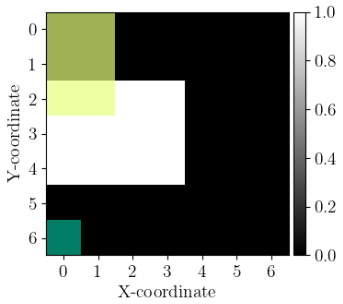


Figure 5.1: Gridworld-O (O for overlap) environment setup, where the true constraint (white refers to a value of 1, black refers to a value of 0) overlaps with the start region (green/yellow). Dark green region is the goal state. This overlap ensures that the constraint is violated in at least some of the trajectories, when the agent starts in a bad state. Due to this, it is more difficult to satisfy the probabilistic threshold.

$0.99, \delta = 0.6$ and ICL with $\beta = 0.99(1 - 0.6) = 0.396$. The recovered constraints and normalized accruals for these experiments are provided in Figure 5.2. As can be seen from Figure 5.2, IPCL can successfully be applied to settings where finding an expected constraint is difficult, or when the expert constraint is probabilistic. Note that this can be extended to arbitrarily complex constraint settings since the constraint function is represented by a neural network.

5.1.2 Alternating procedures and convergence

We now describe an algorithm called IPCL (i.e. *Inverse Probabilistic Constraint Learning*) to directly learn a probabilistic constraint. It follows the same high level procedure that alternates between constrained policy optimization and constraint adjustment as the ICL algorithm (Section 4.1.1), but important algorithmic changes are needed to work with probabilistic constraints instead of expected constraints.

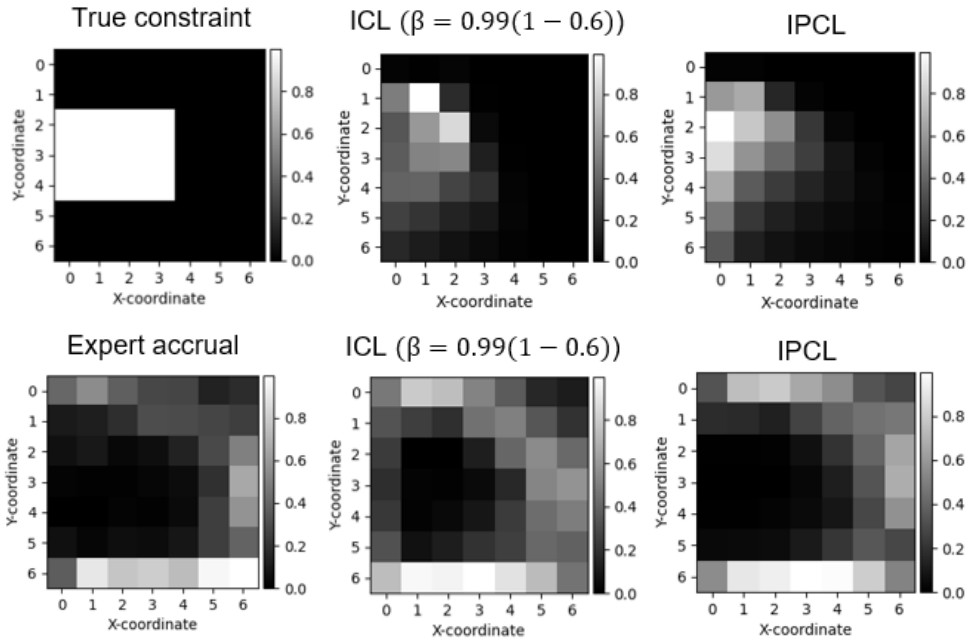


Figure 5.2: Average constraint function value and normalized accruals (averaged across 5 seeds) for the Gridworld-O environment (setup in Figure 5.1). Due to the design of the environment (overlap between true constraint and start states), an expected constraint on the trajectory constraint return is always violated. So, it is difficult for ICL to recover the constraint. A reduced β, i.e. $\beta := \beta(1-\delta)$ makes the problem harder as the overall β decreases. However, IPCL ($\delta = 0.6$) can find a probabilistic constraint that ensures that the per-trajectory constraint holds with confidence at least δ.

IPCL-Policy-Optimization:

$$\theta^* := \arg\max_{\theta} \mathbb{E}_{\tau\sim\pi_\theta}[G_r(\tau)] \text{ s.t. } \mathbb{P}_{\tau\sim\pi_\theta}(G_{c_\phi}(\tau) \leq \beta) \geq \delta \tag{5.6}$$

$$\Pi \leftarrow \Pi \cup \{\pi_{\theta^*}\}$$

IPCL-Constraint-Adjustment:

$$\phi^* := \arg\max_{\phi} \min_{\pi_\theta\in\Pi} \mathbb{P}_{\tau\sim\pi_\theta}(G_{c_\phi}(\tau) > \beta) \text{ s.t. } \mathbb{P}_{\tau\sim\pi^E}(G_{c_\phi}(\tau) \leq \beta) \geq \delta \tag{5.7}$$

In particular, the policy optimization procedure must now be performed with a probabilistic constraint. Additionally, the constraint adjustment procedure in this formulation now uses

the probability of infeasibility $\mathbb{P}_{\tau\sim\pi_\theta}(G_{c_\phi}(\tau) > \beta)$ instead of the expected cost used in the ICL formulation. The effect is analogous: this procedure still selects the most feasible policy and maximizes the probability of infeasibility of this policy, thereby making it more infeasible.

Assuming that the above optimization problems can be solved exactly, we prove that IPCL terminates and finds a policy that is equivalent to the expert policy i.e., the distribution over trajectories induced by both policies are the same.

Theorem 3. *If the space of policies is finite, then the alternation of optimization procedures in Equation (5.6) and Equation (5.7) converges to a set of policies* Π *such that the last policy* π^* *added to* Π *is equivalent to the expert policy* π^E *in the sense that* π^* *and* π^E *generate the same trajectories.*

Proof. We give a proof by contradiction, similar to the convergence proof for Theorem 1. We argue that since there are finitely many policies, and we add a new policy to Π at every iteration, in the worst case, the algorithm will terminate once all policies have been added to Π. Note that a finite space of policies is a reasonable assumption, since in practice we typically only consider policies with a specific parameterization, where the parameters have finite precision.

Suppose that θ^* and ϕ^* are the parameters of the policy π_{θ^*} and constraint c_{ϕ^*} that the algorithm converges to. According to Equation (5.6), the probability that constraint c_{ϕ^*} is violated by policy π_{θ^*} is at most $1-\delta$:

$$\mathbb{P}_{\tau\sim\pi_{\theta^*}}(G_{c_{\phi^*}}(\tau) > \beta) \leq 1-\delta$$

Suppose that $\pi_{\theta^*} \neq \pi^E$, then we can show a contradiction in the sense that the probability that π^* violates c_{ϕ^*} will be greater than $1-\delta$.

$$\begin{aligned}
&\max_{\phi} \min_{\pi_\theta\in\Pi} \mathbb{P}_{\tau\sim\pi_\theta}(G_{c_\phi}(\tau) > \beta) \text{ s.t. } \mathbb{P}_{\tau\sim\pi^E}(G_{c_\phi}(\tau) \leq \beta) \geq \delta \\
&= \max_{\phi} \min_{\pi_\theta\in\Pi} \mathbb{P}_{\tau\sim\pi_\theta}(G_{c_\phi}(\tau) > \beta) \text{ s.t. } \mathbb{P}_{\tau\sim\pi^E}(G_{c_\phi}(\tau) > \beta) \leq 1-\delta \\
&= \min_{\pi_\theta\in\Pi} P_{\tau\sim\pi_\theta}(G_{c_{\phi^*}}(\tau) > \beta) \text{ s.t. } P_{\tau\sim\pi^E}(G_{c_{\phi^*}}(\tau) > \beta) \leq 1-\delta \\
&\geq \min_{\pi_\theta\in\Pi} P_{\tau\sim\pi_\theta}(G_{c_{\phi'}}(\tau) > \beta) \text{ where } \phi^* \neq \phi' \\
&> 1-\delta
\end{aligned}$$

For the last two steps, we can choose $c_{\phi'}(S,A) := (\beta+\epsilon)(1-\gamma), \forall S, A$ for some $\epsilon > 0$. With this $c_{\phi'}$, we have

$$G_{c_{\phi'}}(\tau) = \sum_t \gamma^t c_{\phi'}(S_t, A_t) = (\beta+\epsilon)(1-\gamma)\sum_t \gamma^t$$

Then in the infinite horizon setting,

$$P_{\tau\sim\pi_\theta}\left(G_{c_{\phi'}}(\tau) > \beta\right) = P_{\tau\sim\pi_\theta}\left(\frac{(\beta+\epsilon)(1-\gamma)}{1-\gamma} > \beta\right) = 1 > 1-\delta$$

As a result, the minimum probability of violation among the policies in Π is greater than $1-\delta$.

□

5.1.3 Practical algorithm

In practice, we have the same problem as before, i.e. the max-min optimization in the constraint adjustment step (Equation (5.7)) is difficult to solve, hence we approximate it in the same way as the original ICL algorithm (Section 4.1.2) by replacing the minimization over Π by a single mixture policy π_{mix} that consists of a weighted combination of the policies in Π. The weight of each policy depends on the policy's dissimilarity to the expert. More specifically, the mixture policy reweighs optimal policies found in the policy optimization step (as well as trajectories sampled from these policies), so that behaviour dissimilar to the expert has a higher weight in the mixture policy (Section 4.1.3). This gives us the following simplified procedure.

$$\textsc{Simplified-Constraint-Adjustment:}$$
$$\phi^* := \arg\max_\phi \mathbb{P}_{\tau\sim\pi_{\text{mix}}}(G_{c_\phi}(\tau)) > \beta) \text{ s.t. } \mathbb{P}_{\tau\sim\pi^E}(G_{c_\phi}(\tau) \leq \beta) \geq \delta \tag{5.8}$$

Overall, finding a probabilistic constraint requires alternating between procedures Equation (5.6) and Equation (5.8). Directly solving the above optimization problems by gradient descent based optimization is still challenging since this requires differentiating through cumulative density functions that are not known in practice.

5.1.4 Differentiable approximation of cumulative density function

In this section, we provide an approximation of the cumulative density function that enables gradient-based optimization for the problems in Equation (5.6) and Equation (5.8).

First, we note the following regarding cumulative density functions.

$$\mathbb{P}_{\tau\sim\pi}(G_c(\tau) \leq \beta) \quad = \int_{\tau} \mathbb{P}_{\tau\sim\pi}(\tau)\ \mathbf{I}(G_c(\tau) \leq \beta) d\tau \quad = \mathbb{E}_{\tau\sim\pi}[\mathbf{I}(G_c(\tau) \leq \beta)] \tag{5.9}$$

Overall, this states that a cumulative density function value can be rewritten as an expectation of indicator functions. The indicator function, in turn, can also be approximated by a sigmoid-like function such as the logistic function, tanh function, etc. In this work, we approximate the indicator function as follows[4],

$$\begin{gathered} \mathbf{I}(a \leq b) \approx h(a, b) \\ h(a, b) := \frac{1}{2} - \frac{1}{2}\left(\frac{(a-b)}{\sqrt{(a-b)^2 + \epsilon}}\right), \text{ where } \epsilon \approx 0 \end{gathered} \tag{5.10}$$

Together, Equations (5.9) and (5.10) allow us to approximate the cumulative density function such that optimizations represented by Equations (5.6) and (5.8) become tractable.

5.2 Implementation

We now provide algorithms to solve Equation (5.6) and Equation (5.8) using the penalty method. We do not use the Lagrangian method to solve Equation (5.8) since it suffers from the same oscillatory behaviour that we encountered in the implementation of ICL constraint function adjustment (Section 4.2). Similarly, for constrained policy optimization, we do not use PPO Lagrangian (provided in OpenAI's implementation [113]) since it cannot handle probabilistic constraints and significant changes were required to make it work with probabilistic constraints. Instead, we modify PPO Penalty (Section 4.2.1) to work with probabilistic constraints, and optimize it to work with complex environments as well.

5.2.1 Probabilistic constrained reinforcement learning (PCRL)

Using the penalty method for Equation (5.6), we get the following update rules.

[4]The primary reason for choosing this algebraic approximation over a sigmoid-like approximation is that away from the region close to $a = b$, the derivative of sigmoid-like functions is very close to 0, while for this algebraic approximation, it is small but not as close to 0. This makes gradient-based learning more feasible.

$$\textsc{Feasibility Projection}: \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_1 \nabla_{\boldsymbol{\theta}} \textsc{ReLU}(\delta - \mathbb{P}_{\tau \sim \pi_{\boldsymbol{\theta}}}(G_{c_{\boldsymbol{\phi}}}(\tau) \leq \beta)) \tag{5.11}$$

$$\textsc{PPO Update}: \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_2 \nabla_{\boldsymbol{\theta}} \textsc{PPO-Loss}(\pi_{\boldsymbol{\theta}}) \tag{5.12}$$

5.2.2 Constraint adjustment

Using the penalty method for probabilistic constraint adjustment (Equation (5.8)), we arrive at the following update rule.

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \eta_3 \nabla_{\boldsymbol{\phi}} \left\{ 1 - \mathbb{P}_{\tau \sim \pi_{\text{mix}}}(G_{c_{\boldsymbol{\phi}}}(\tau) \leq \beta) + \lambda \textsc{ReLU}(\delta - \mathbb{P}_{\tau \sim \pi_{\boldsymbol{\theta}}}(G_{c_{\boldsymbol{\phi}}}(\tau) \leq \beta) \right\} \tag{5.13}$$

5.2.3 Differentiating the cumulative density function

The gradient of both these objectives require us to then compute the gradient of the ReLU term, which can be written in terms of the derivative of the cumulative density function as follows.

$$\nabla_{\theta} \textsc{ReLU}\left(\delta - \mathbb{P}_{\tau \sim \pi_{\boldsymbol{\theta}}}\left(G_{c_{\boldsymbol{\phi}}}(\tau) \leq \beta\right)\right) =$$
$$-\mathbf{I}\left(\mathbb{P}_{\tau \sim \pi_{\boldsymbol{\theta}}}\left(G_{c_{\boldsymbol{\phi}}}(\tau) \leq \beta\right) < \delta\right) \nabla_{\theta}\left(\mathbb{P}_{\tau \sim \pi_{\boldsymbol{\theta}}}\left(G_{c_{\boldsymbol{\phi}}}(\tau) \leq \beta\right)\right) \tag{5.14}$$

The analytical form of the derivative of the cumulative density function is described below, using the approximation in Equation (5.10).

Lemma 4. *The derivative of* $\mathbb{P}_{\tau \sim \pi_{\boldsymbol{\theta}}}(f(\tau) \leq \beta)$ *(where* $f(\tau) := G_{c_{\boldsymbol{\phi}}}(\tau)$*) with respect to policy parameters* $\boldsymbol{\theta}$ *is:*

$$\nabla_{\boldsymbol{\theta}} \mathbb{P}_{\tau \sim \pi_{\boldsymbol{\theta}}}(f(\tau) \leq \beta) =$$
$$\mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}\left[h(f(\tau), \beta) \sum_t \nabla_{\theta} \log \pi_{\boldsymbol{\theta}}(a_t | s_t)\right] - \frac{\epsilon}{2} \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}\left[\frac{\nabla_{\theta} f(\tau)}{(\sqrt{(f(\tau) - \beta)^2 + \epsilon})^3}\right] \tag{5.15}$$

Here, $h(a, b)$ *is defined as in Equation (5.10),*

$$\mathbf{I}(a \leq b) \approx h(a, b) := \frac{1}{2} - \frac{1}{2}\left(\frac{(a - b)}{\sqrt{(a - b)^2 + \epsilon}}\right), \ \epsilon \approx 0$$

Proof. We can prove this as follows,

$$\begin{aligned}
\nabla_\theta \mathbb{P}_{\tau\sim\pi_\theta}(f(\tau) \leq \beta) &= \nabla_\theta \int \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ \mathbf{I}(f(\tau) \leq \beta) d\tau \\
&\approx \nabla_\theta \int \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ h(f(\tau), \beta) d\tau \\
&= \int \Big[(\nabla_\theta \mathbb{P}_{\tau\sim\pi_\theta}(\tau))\ h(f(\tau), \beta) + \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ \nabla_\theta (h(f(\tau), \beta)) \Big] d\tau \quad (5.16) \\
&\quad \text{using derivative chain rule}
\end{aligned}$$

We can compute these two terms separately. The first term is,

$$\begin{aligned}
\int (\nabla_\theta \mathbb{P}_{\tau\sim\pi_\theta}(\tau))\ h(f(\tau), \beta) d\tau &= \int (\mathbb{P}_{\tau\sim\pi_\theta}(\tau) \nabla_\theta \log \mathbb{P}_{\tau\sim\pi_\theta}(\tau))\ h(f(\tau), \beta) d\tau, \text{ using log trick} \\
&= \mathbb{E}_{\tau\sim\pi_\theta} \Big[\nabla_\theta \log \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ h(f(\tau), \beta) \Big] \\
&= \mathbb{E}_{\tau\sim\pi_\theta} \Big[\nabla_\theta \Big\{ \log p(S_0) \quad (5.17) \\
&\qquad + \sum_t \Big(\log \pi_\theta(A_t|S_t) + \log p(S_{t+1}|S_t, A_t) \Big) \Big\} h(f(\tau), \beta) \Big] \\
&\quad \text{using the definition of probability of a trajectory, see [66]} \\
&= \mathbb{E}_{\tau\sim\pi_\theta} \Big[\Big(\sum_t \nabla_\theta \log \pi_\theta(A_t|S_t) \Big)\ h(f(\tau), \beta) \Big] \quad (5.18)
\end{aligned}$$

The second term can be obtained as follows,

$$\begin{aligned}
\int \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ \nabla_\theta (h(f(\tau), \beta)) d\tau &= \int \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ \nabla_\theta \Big\{ \frac{1}{2} - \frac{1}{2} \Big(\frac{(f(\tau) - \beta)}{\sqrt{(f(\tau) - \beta)^2 + \epsilon}} \Big) \Big\} d\tau \\
&= -\frac{1}{2} \int \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ \nabla_\theta \Big\{ \frac{(f(\tau) - \beta)}{\sqrt{(f(\tau) - \beta)^2 + \epsilon}} \Big\} d\tau \\
&= -\frac{1}{2} \int \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ \Big\{ \frac{\epsilon\ \nabla_\theta f(\tau)}{(\sqrt{(f(\tau) - \beta)^2 + \epsilon})^3} \Big\} d\tau \quad (5.19) \\
&\quad \text{using derivative quotient rule} \\
&= -\frac{\epsilon}{2} \mathbb{E}_{\tau\sim\pi_\theta} \Big[\frac{\nabla_\theta f(\tau)}{(\sqrt{(f(\tau) - \beta)^2 + \epsilon})^3} \Big] \quad (5.20)
\end{aligned}$$

Using Equations (5.18) and (5.20) in Equation (5.16), we get the required result. □

The first term of the expression can be computed for both discrete or continuous action-space policies using automatic differentiation. The second term has $\nabla_\theta f(\tau)$, which is not directly computable since for a stochastic policy, the actions are sampled from $\pi_{\boldsymbol{\theta}}$, and so the discounted sum of constraint values isn't directly differentiable in terms of θ. *The sampling can however, be rewritten using the reparameterization trick.* For continuous action-space policies, if we use Gaussian policies, this is similar to the reparameterization trick used in variational auto-encoders [61]. If we use a discrete action-space policy, we need to reparameterize using the Gumbel-softmax trick [57] (see Section 2.9.3). After the reparameterization, the quantity becomes differentiable.

Lemma 5. *The derivative of* $\mathbb{P}_{\tau\sim\pi_{\boldsymbol{\theta}}}(f(\tau) \leq \beta)$ *(where* $f(\tau) := G_{c_{\boldsymbol{\phi}}}(\tau)$*) with respect to constraint function parameters* ϕ *is given as,*

$$\nabla_\phi \mathbb{P}_{\tau\sim\pi_{\boldsymbol{\theta}}}(f(\tau) \leq \beta) = -\frac{\epsilon}{2}\mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}\left[\frac{\nabla_\phi f(\tau)}{(\sqrt{(f(\tau)-\beta)^2+\epsilon})^3}\right] \tag{5.21}$$

Proof. We can see that,

$$\begin{aligned}
\nabla_\phi \mathbb{P}_{\tau\sim\pi_\theta}(f(\tau) \leq \beta) &= \nabla_\phi \int \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ \mathbf{I}(f(\tau) \leq \beta) d\tau \\
&\approx \nabla_\phi \int \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ h(f(\tau), \beta) d\tau \\
&= \int \mathbb{P}_{\tau\sim\pi_\theta}(\tau)\ \nabla_\phi (h(f(\tau), \beta))\, d\tau, \text{ since } \mathbb{P}_{\tau\sim\pi_\theta}(\tau) \text{ doesn't depend on } \phi
\end{aligned}$$

This quantity is very similar to the second term of Equation (5.16) (see proof of Lemma 4). Using the quotient rule in a similar way (Equation (5.20)), we can arrive at the required result. □

5.2.4 Overall algorithm

The overall algorithm for IPCL is provided in Algorithm 5. We now describe this algorithm in further detail.

First, the algorithm trains a normalizing flow on the expert dataset (lines 1-2). Similar to the ICL algorithm (Algorithm 1), it alternates between constrained policy optimization (lines 5-10) and constraint function adjustment (lines 11-15) until the iterations are exhausted or until the agent's policy becomes similar to the expert policy (line 16). For constrained policy optimization, the algorithm learns an optimal policy given a probabilistic constraint, by following the method described in Section 5.2.1. For constraint function adjustment, the algorithm learns a constraint function given optimal policies from previous iterations, as per the description in Section 5.2.2. The mixture policy is computed as per the process described in ICL mixture policy computation, in Section 4.1.3. In the mixture policy, the policies dissimilar to the expert policy are weighed more. Finally, the algorithm outputs the most recent learned policy and the learned constraint function.

Algorithm 5 INVERSE-PROBABILISTIC-CONSTRAINT-LEARNING (IPCL)

hyper-parameters: iterations n, penalty weight λ, learning rates η_1, η_2, η_3
define: $L^{\beta}_{\pi} := \mathbb{P}_{\tau\sim\pi}(G_{c_\phi}(\tau) \leq \beta)$ ▷ See Equation (5.10)
input: expert dataset $\mathcal{D}_E = \{\tau\}_{\tau\in\mathcal{D}}$, thresholds β, δ
1: **initialize** normalizing flow f
2: **optimize** likelihood of f on expert state action data: $\max_f \sum_{(s,a)\in\tau,\tau\in\mathcal{D}}(\log p_f(s,a))$
3: **initialize** constraint function $c_{\boldsymbol{\phi}}$
4: **for** $1 \leq i \leq n$ **do**
5: **initialize** policy $\pi_{\boldsymbol{\theta}_i}$
6: **perform** policy optimization
7: *alternate until convergence, between*
8: $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \eta_1 \nabla_{\boldsymbol{\theta}_i} \text{ReLU}(\delta - L^{\beta}_{\pi_{\boldsymbol{\theta}_i}})$ ▷ Equation (5.14), Lemma 4 (*constraint feasibility*)
9: *and*
10: $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \eta_2 \nabla_{\boldsymbol{\theta}_i} \text{PPO-Loss}(\pi_{\boldsymbol{\theta}_i})$ ▷ PPO [119]
11: **perform** constraint adjustment
12: $L_{\text{soft}}(c_{\boldsymbol{\phi}}) := -(1 - L^{\beta}_{\pi_{\text{mix}}}) + \lambda \text{ReLU}(\delta - L^{\beta}_{\pi_E})$
13:
14: $\phi \leftarrow \phi - \eta_3 \nabla_{\phi} L_{\text{soft}}(c_\phi)$ ▷ compute gradient using Lemma 5
15: ▷ (for π_{mix} weight computation, see Algorithm 3)
16: **break** if NORMALIZED-ACCRUAL-DISSIMILARITY$(\mathcal{D}, \mathcal{D}_{\pi_{\boldsymbol{\theta}_i}}) \leq \epsilon$
▷ *See Section 5.3.4* for normalized accrual dissimilarity metric
17: **end for**
18: **output:** $c_{\boldsymbol{\phi}}$, most recent policy π_i

5.3 Experiments

5.3.1 Experiment design

We wish to design experiments that answer the following questions:

- Does directly learning the probabilistic constraint (IPCL) perform better than indirectly learning the constraint (ICL with $\beta := \beta(1-\delta)$), in terms of constraint recovery and accrual similarity between the agent's policy and expert policy?
- Does ICL with $\beta := \beta(1-\delta)$ learn an expected constraint that empirically satisfies the probabilistic constraint, as postulated by Theorem 2?
- What is the effect of δ in terms of learning a constraint? Does a high δ correspond to a hard constraint, and if so, what are the differences w.r.t. constraint learned with a lower δ?

In order to answer these questions, we conduct experiments with the following environments.

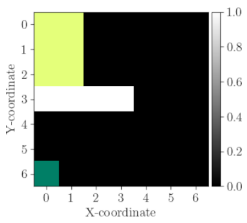
5.3.2 Environments

Gridworld Environment

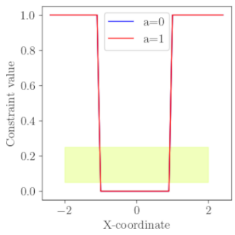
The setup of the Gridworld environment is provided in Figure 5.3. This 7x7 gridworld environment is the same as Gridworld (A) environment in Chapter 4. The agent starts in the start region and must navigate to the goal. Without the constraint, the agent can directly go downward towards the goal, which leads to a different policy than if the constraint is present. The state space consists of two features, i.e. x and y positions, overall assuming 7x7=49 values. The action space consists of 8 discrete actions including 4 nominal directions and 4 diagonal directions.

CartPole environment

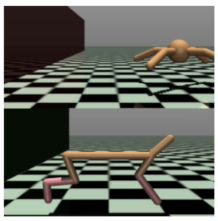
The setup of the CartPole environment is provided in Figure 5.3. This environment is different from CartPole (MR) and CartPole (Mid) environments in Chapter 4. More specifically, in this environment, the agent (balancing the pole in the standard CartPole problem) starts



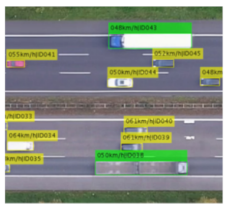
Gridworld



CartPole



Mujoco environments, source: [88].



HighD environment based on [63].

Figure 5.3: Environment setup for various environments. For the Gridworld environment, white regions indicate constraint value of 1, green/yellow regions correspond to start states, and dark green region is the goal state. For the CartPole environment, start region is shown in yellow, the true constraint value is shown on Y-axis while the X-coordinate of the pole is shown on the X-axis. The Mujoco and HighD environments follow the same setup as ICL experiments in Chapter 4.

in the yellow region and must navigate to keep the constraint satisfied, depending on the constraint specification. The agent may start in a region of high constraint value (yellow region can correspond to high cost or low cost), and the objective is to move to a region of low constraint value and balance the pole there, while the constraint function is being learned. There are two actions in this environment, i.e. $a = 0$ (left) and $a = 1$ (right). The state and action spaces are the same as the standard CartPole environment [18].

Mujoco environments

These comprise the Ant-Constrained and HalfCheetah-Constrained environments (Figure 5.3), which is the same as the Mujoco robotics environments used in ICL experiments. The constraint used is $z \geq -1$, i.e. the agent must stay in region $z \geq -1$. The state and action spaces are the same as the standard Ant and HalfCheetah Mujoco environments [130].

HighD environment

This environment is the same as the HighD environment used in ICL experiments. For this environment (Figure 5.3), we use the real world highway dataset simulator (HighD) from Gaurav et al. [42], which in turn, is based on dataset and environments by Krajewski

et al. [63] and Lee et al. [65]. We describe this environment again for clarity. The agent starts on a straight road on the left side of the highway, and the goal is to reach the right side of the highway without any longitudinal collisions. The state space is comprised of 7 ego features and 7 other features (predicates, propositions and processed features). The action space consists of two continuous actions, i.e., acceleration and rate of change of steering angle. Here, the true constraint function is not known. Instead, we want to learn a constraint function that explains the relation between the ego velocity and the gap to the car in the front.

5.3.3 Methods and baselines

We conduct experiments with three algorithms in order to answer the posed questions:

1. **ICL** with a modified β, which recovers a probabilistic constraint, i.e. with $\beta := \beta(1-\delta)$ as described in Theorem 2[5]
2. **IPCL** described in Algorithm 5
3. **UAICRL i.e. Uncertainty Aware ICRL** [134] which is a recent method that supports learning a constraint similar to the probabilistic constraint defined in this work. More specifically, the UAICRL process alternates between risk sensitive CRL and constraint function adjustment. The risk sensitive CRL procedure updates the policy using techniques from distributional RL, capturing aleatoric uncertainty and inherent stochasticity in the environment and dynamics. The constraint function adjustment reduces epistemic uncertainty (uncertainty due to lack of knowledge on out of distribution data) and augments the agent's dataset through flow based trajectory generation.

5.3.4 Metrics

We define and report three metrics for our IPCL experiments:

1. **Constraint Mean Squared Error i.e. CMSE**, which is the mean squared error between the true constraint and the recovered constraint, computed on a uniformly discretized state-action space for the respective environment. This is the same metric

[5] ICL with β is not directly comparable with algorithms that learn a probabilistic constraint

which is described in Section 4.3.4. We provide the details of the discretization in Appendix B.3. If the discrete set of state-action pairs is $\mathcal{D}$, then

$$\text{CMSE}(c, c_{true}) = \frac{1}{|\mathcal{D}|} \sum_{(s,a)\in\mathcal{D}} (c(s,a) - c_{true}(s,a))^2 \tag{5.22}$$

2. **Normalized Accrual Dissimilarity i.e. NAD**, which is a dissimilarity measure computed between the expert and agent accruals (state-action visitations of policy/dataset). This is the same metric which is described in Section 4.3.4. Given a policy learned by a method like ICL or IPCL, we compute an agent dataset of trajectories. Then, the accrual (state-action visitation frequency) is computed for both the agent dataset and the expert dataset over a uniformly discretized state-action space, which is the same as the one used for CMSE. Finally, the accruals are normalized, and the Wasserstein distance is computed between the accruals. Again, we provide the details of the discretization in Appendix B.3. The pseudocode for this computation is provided in Algorithm 4.

3. **Probability of constraint satisfaction i.e. cumulative density function (CDF) values** corresponding to $\mathbb{P}_{\tau\sim\pi}(G_c(\tau) \leq \beta)$, which should be $\geq \delta$ depending on the environment. This is equivalent to the expectation of indicator variables (Equation (5.9)):
$$\mathbb{P}_{\tau\sim\pi}(G_c(\tau) \leq \beta) = \mathbb{E}_{\tau\sim\pi}[\mathbf{I}(G_c(\tau) \leq \beta)]$$

5.3.5 Training setup

We perform 5 training runs for all the methods. We also use a constraint neural network with a sigmoid output[6], so that we can also interpret the outputs as safeness values[7], similar to the training setup in Chapter 4.

5.3.6 Results

Our results compare the performances of three algorithms: ICL with $\beta := \beta(1-\delta)$, IPCL and UAICRL. As shown in the previous section, there are trivial cases where ICL with

[6]The complete architecture includes an input layer, 2 hidden layers of 64 nodes each, and an output layer with a sigmoid output. The intermediate activations are ReLU.

[7]A constraint value of 0 indicates complete feasibility or safe state, while a constraint value of 1 indicates infeasibility or unsafe state.

Table 5.1: Choice of β, δ

Environment	β	δ
Gridworld	0.99	0.6
CartPole	20	0.7
Ant-Constrained	15	0.7
HalfCheetah-Constrained	15	0.5
HighD	0.1	$0.5, 0.9$

Table 5.2: Constraint mean squared error for IPCL experiments (lower is better)

Environment	**ICL** w/ $\beta(1-\delta)$	**IPCL** w/ β, δ	**UAICRL**
Gridworld	0.08 ± 0.01	$\mathbf{0.07 \pm 0.00}$	0.37 ± 0.00
CartPole	0.33 ± 0.02	$\mathbf{0.17 \pm 0.04}$	0.59 ± 0.00
Ant-Constrained	$\mathbf{0.02 \pm 0.00}$	0.13 ± 0.10	0.22 ± 0.00
HalfCheetah-Constrained	0.16 ± 0.15	$\mathbf{0.08 \pm 0.01}$	0.31 ± 0.00

$\beta := \beta(1-\delta)$ can fail to recover a suitable probabilistic constraint, and in such cases, IPCL (Algorithm 5) can be used. The choices of β, δ are provided in Table 5.1. Overall, δ should be a high value but not too high since then it becomes infeasible in stochastic environments. Similarly, a low δ is quite easy to satisfy and not really useful in practice.

Constraint error

The CMSE results for IPCL experiments are provided in Table 5.2 (lower is better). Further, the constraint function plots are provided for Gridworld (Figure 5.4), CartPole (Figure 5.5), Ant-Constrained (Figure 5.6), and HalfCheetah-Constrained (Figure 5.7) environments. Note that in the constraint function plots, when the constraint value is 1, it can be interpreted as an infeasible state or state-action pair (i.e. a safeness violation) that makes the constraint satisfaction less likely.

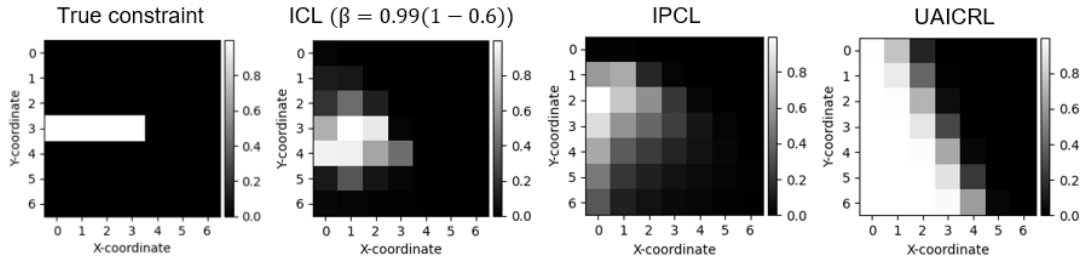


Figure 5.4: Average constraint function value (averaged across 5 seeds) for Gridworld environment.

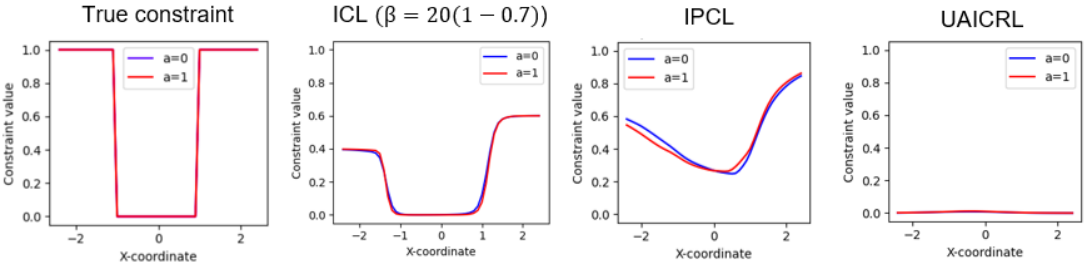


Figure 5.5: Average constraint function value (averaged across 5 seeds) for CartPole environment.

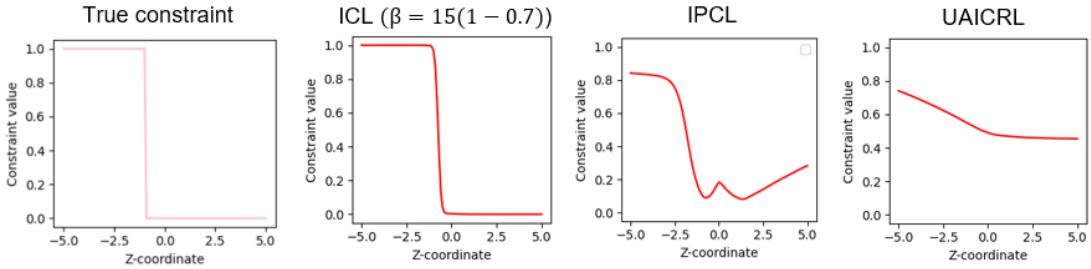


Figure 5.6: Average constraint function value (averaged across 5 seeds) for Ant-Constrained environment.

Table 5.3: Normalized accrual dissimilarity for IPCL experiments (lower is better)

Environment	ICL w/ $\beta(1-\delta)$	IPCL w/ β, δ	UAICRL
Gridworld	1.24 ± 0.88	$\mathbf{0.57 \pm 0.13}$	4.07 ± 0.17
CartPole	4.86 ± 2.73	$\mathbf{3.10 \pm 0.99}$	3.04 ± 1.37
Ant-Constrained	3.67 ± 0.91	$\mathbf{2.75 \pm 0.51}$	3.20 ± 1.02
HalfCheetah-Constrained	7.33 ± 5.46	$\mathbf{6.62 \pm 1.85}$	13.08 ± 1.66

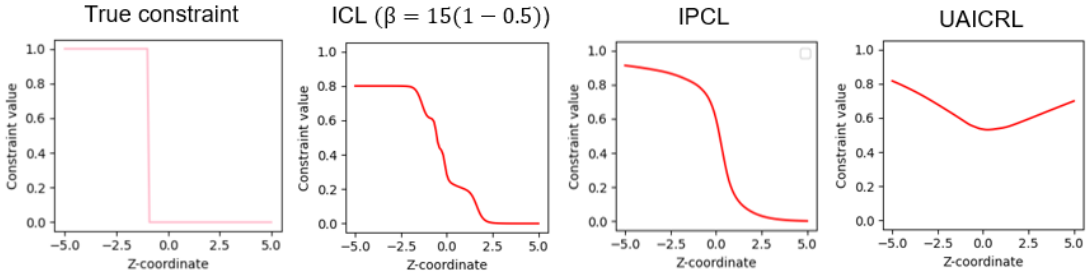


Figure 5.7: Average constraint function value (averaged across 5 seeds) for HalfCheetah-Constrained environment.

We find that in terms of accuracy, IPCL recovers more accurate constraints (lower CMSE in Table 5.2) compared to ICL with $\beta := \beta(1-\delta)$ and UAICRL. That is, in 3 out of 4 environments (i.e. except Ant environment), IPCL's accuracy is better than the accuracy of ICL with $\beta := \beta(1-\delta)$. UAICRL can recover a constraint in all but one environment (CartPole environment), but the recovered constraint is either less sharp (Mujoco environments) or overfit to the expert data (Gridworld environment).

Accrual dissimilarity

The NAD results for IPCL experiments are provided in Table 5.3 (lower is better). Further, the normalized accrual plots are provided below for Gridworld (Figure 5.8), CartPole (Figure 5.9), Ant-Constrained (Figure 5.10) and HalfCheetah-Constrained (Figure 5.11) environments.

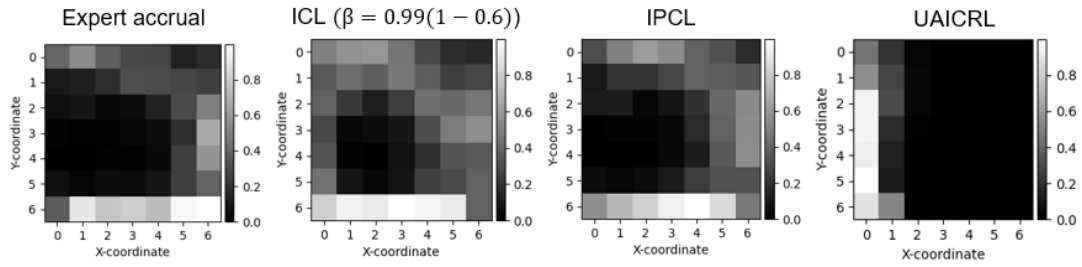


Figure 5.8: Average normalized accruals (averaged across 5 seeds) for Gridworld environment.

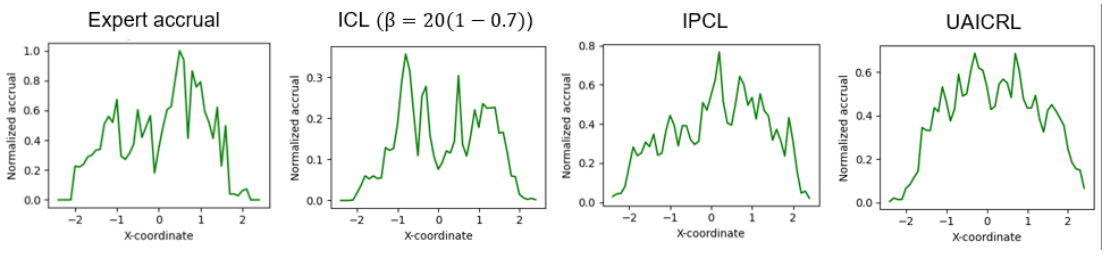


Figure 5.9: Average normalized accruals (averaged across 5 seeds) for CartPole environment.

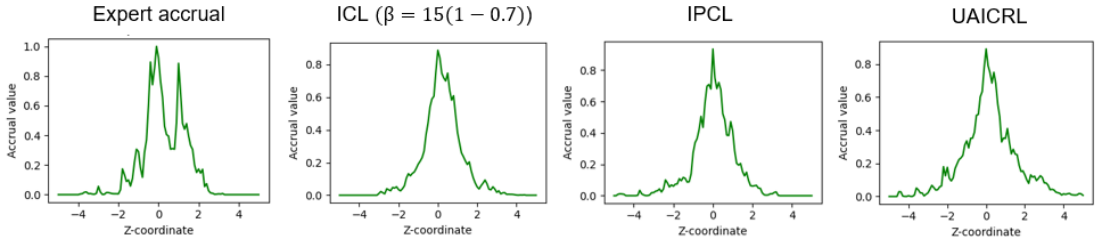


Figure 5.10: Average normalized accruals (averaged across 5 seeds) for Ant-Constrained environment.

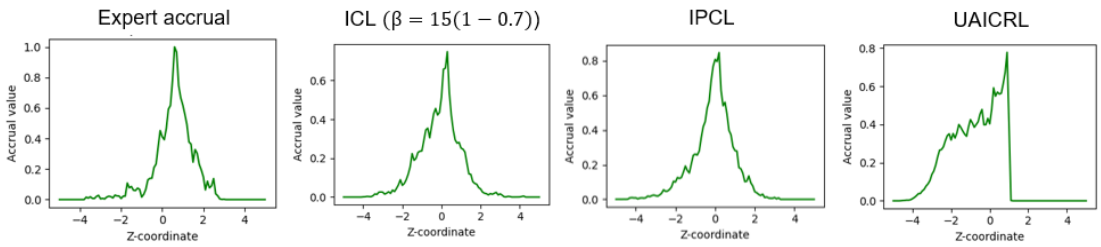


Figure 5.11: Average normalized accruals (averaged across 5 seeds) for HalfCheetah-Constrained environment.

In terms of the learned policy, IPCL is better than ICL with $\beta := \beta(1-\delta)$ in all the 4 environments (see Table 5.3). In the case of Ant environment, even though ICL with $\beta := \beta(1-\delta)$ has a lower CMSE (Table 5.2), IPCL has a better NAD. A higher CMSE could mean that an alternative constraint is being learned, for which the NAD is lower as it still yields the expert policy. Additionally, since the final accrual is generated by running the forward CRL procedure with the learned constraint, the NAD metric also depends on the forward CRL procedure. This, in turn, means that the accrual quality depends on the CRL procedure and the specific environment. This procedure is different in ICL with $\beta := \beta(1-\delta)$ and IPCL. In ICL, we use PPO-Lagrange provided by OpenAI [113], whereas in IPCL we use PPO-Penalty proposed in [42] and modified it to learn a probabilistic constraint directly.

We also note that the accruals don't seem to go much further to the right. Running the CRL procedure for a longer period of time will produce a better expert or agent policy that goes further right.

Constraint satisfaction

We provide the probability of constraint satisfaction for IPCL experiments in Table 5.4. If these probabilities are at least δ, then the probabilistic constraint is satisfied. Further, we also provide the histograms corresponding to 50 trajectory samples from the learned policy for Gridworld (Figure 5.12), CartPole (Figure 5.13), Ant-Constrained (Figure 5.14) and HalfCheetah-Constrained (Figure 5.15) environments.

Table 5.4: Probability of constraint satisfaction for IPCL experiments (bold if $\geq \delta$)

Environment (δ)	**ICL** w/ $\beta(1-\delta)$	**IPCL** w/ β, δ	**UAICRL**
Gridworld (0.6)	$\mathbf{0.86 \pm 0.28}$	$\mathbf{0.72 \pm 0.06}$	0.00 ± 0.00
CartPole (0.7)	$\mathbf{0.95 \pm 0.02}$	$\mathbf{0.71 \pm 0.11}$	$\mathbf{1.00 \pm 0.00}$
Ant-Constrained (0.7)	$\mathbf{0.85 \pm 0.03}$	$\mathbf{0.72 \pm 0.32}$	0.00 ± 0.00
HalfCheetah-Constrained (0.5)	$\mathbf{0.72 \pm 0.37}$	$\mathbf{0.61 \pm 0.17}$	0.00 ± 0.00

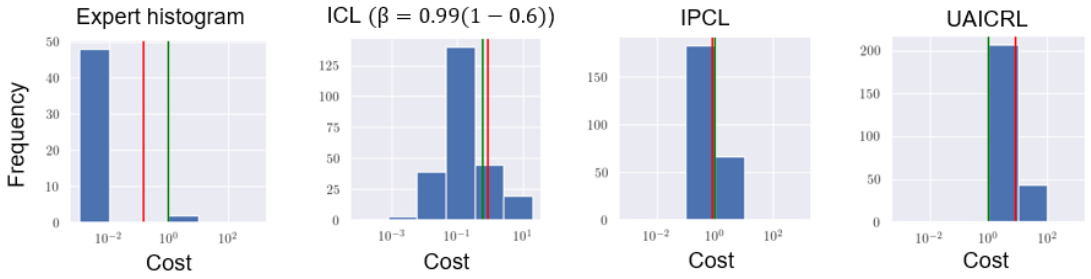


Figure 5.12: Histograms for the cumulative density function of the constraint values based on samples (Gridworld environment). Red line denotes the average value of the samples and green line denotes $\beta = 0.99$. Roughly, at least $\delta * 100\% = 60\%$ of the trajectories should be before the green line.

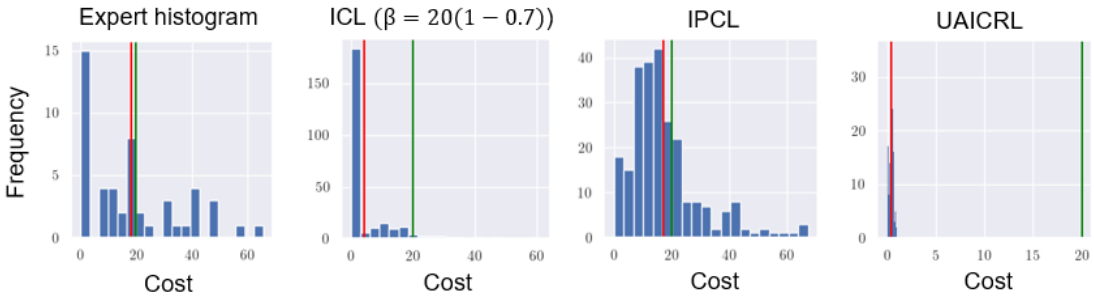


Figure 5.13: Histograms for the cumulative density function of the constraint values based on samples (CartPole environment). Red line denotes the average value of the samples and green line denotes $\beta = 20$. Roughly, at least $\delta * 100\% = 70\%$ of the trajectories should be before the green line.

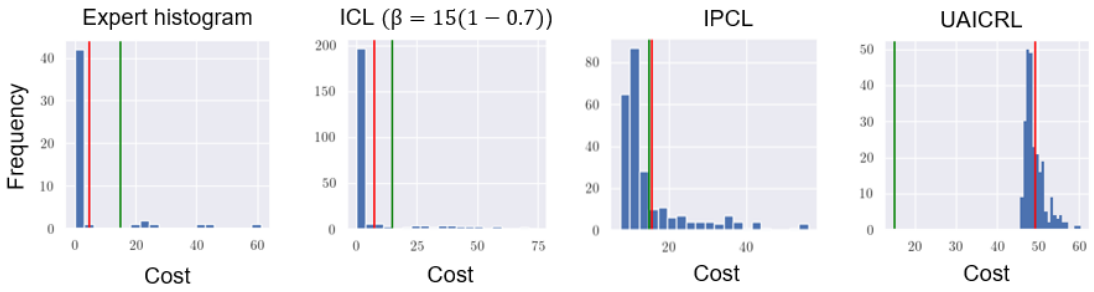


Figure 5.14: Histograms for the cumulative density function of the constraint values based on samples (Ant-Constrained environment). Red line denotes the average value of the samples and green line denotes $\beta = 15$. Roughly, at least $\delta * 100\% = 70\%$ of the trajectories should be before the green line.

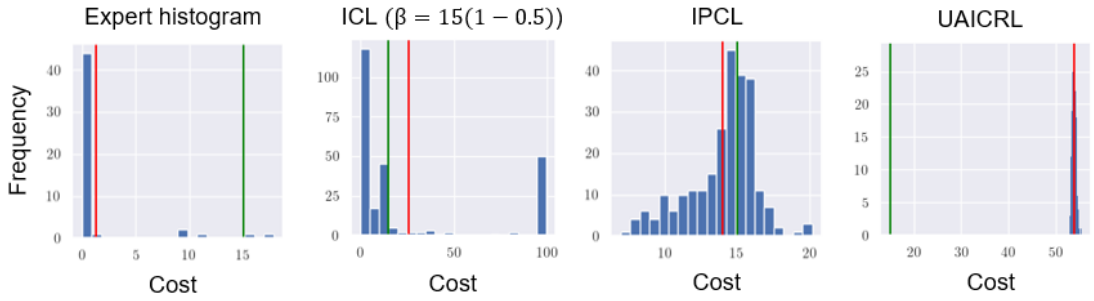


Figure 5.15: Histograms for the cumulative density function of the constraint values based on samples (HalfCheetah-Constrained environment). Red line denotes the average value of the samples and green line denotes $\beta = 15$. Roughly, at least $\delta * 100\% = 50\%$ of the trajectories should be before the green line.

Looking at probability of satisfaction, ICL with $\beta := \beta(1 - \delta)$ learns a more conservative policy on average (for e.g. in Cartpole, $\delta = 0.7$, but the learned policy satisfies $\delta = 0.9$). Another observation is that both IPCL and ICL final policies always satisfy the probabilistic constraint (for the learned constraint and specific δ-threshold), as expected.

Performance of UAICRL

UAICRL [134] is only able to satisfy the probabilistic constraint in 1 out of 4 environments. Additionally, the learned constraints are not sharp compared to the proposed methods,

although the accrual dissimilarity is low for 3 out of 4 environments. This could be because we run UAICRL to learn a VaR constraint (which is exactly the kind of constraint we are learning in this chapter), however, UAICRL was originally intended to learn a CVaR constraint.

Experiments with real world dataset environment

We provide the recovered constraints for the HighD environment in Figure 5.16, the normalized accruals in Figure 5.17), and the histograms corresponding to 50 trajectory samples from the learned policy in Figure 5.18.

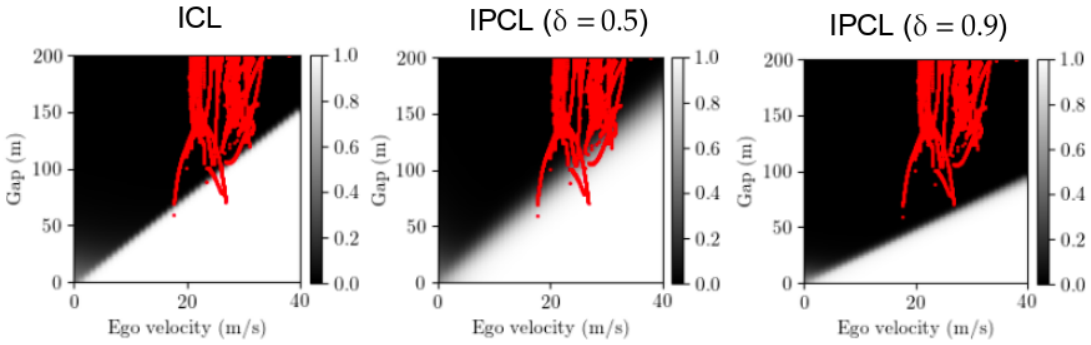


Figure 5.16: Recovered constraint function for the HighD environment (averaged over 5 seeds). The red scatter points correspond to the expert accruals, provided here for clarity. In the learned constraint, the expert trajectories should lie mostly in the feasible or black region.

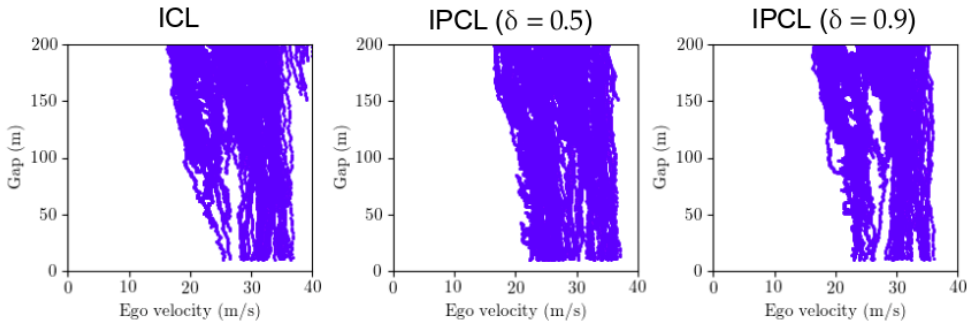


Figure 5.17: Accruals for the HighD environment ($\beta = 0.1$)

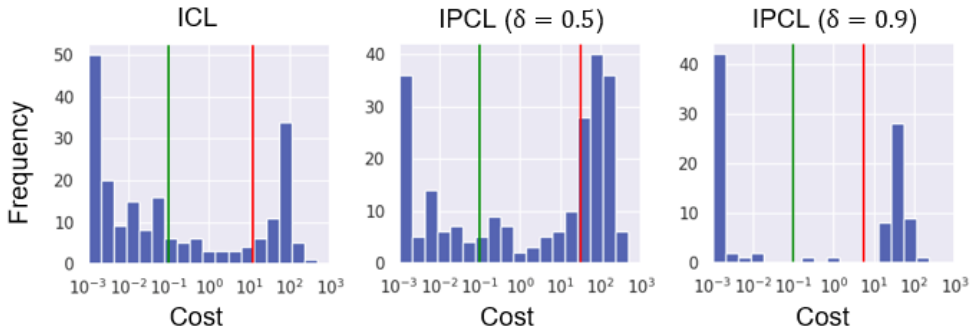


Figure 5.18: Histograms for the HighD environment (logarithmic x-scale, $\beta = 0.1$). Red line denotes the sample average and green line denotes $\beta = 0.1$. Roughly, at least $\delta * 100\%$ of the trajectories should be before the green line.

As can be seen in the recovered constraint plots for the HighD environment[8] (Figure 5.16), a high δ learns a constraint that completely avoids any overlap between the learned constraint and the expert behaviour. This is closer to a hard constraint. Similarly, a lower δ learns a constraint with an overlap that can be defined using δ. In this way, a probabilistic constraint is *more flexible* than a traditional hard constraint, since it can represent hardness depending on the choice of δ.

The accruals are mostly similar to each other, but not too close to the expert. This is also seen in ICL experiments.

5.4 Summary

In this chapter, we proposed methods to learn a probabilistic constraint from expert demonstrations. Specifically, we provide two methods that learn such a constraint, one which reuses existing methods that learn an expected constraint, and another which directly learns a probabilistic constraint. We conduct experiments to compare the performances of the proposed techniques and a baseline, i.e. UAICRL [134]. Our results indicate that in terms of finding an accurate and sharp constraint, the proposed method IPCL which directly learns a probabilistic constraint works better than the method that reuses techniques to learn an expected constraint (i.e. ICL with $\beta := \beta(1-\delta)$). In terms of constraint satisfaction, we find that ICL's policy is more conservative than needed (i.e. it satisfies

[8] In these experiments, the true constraint is not known

the constraint too strongly in all cases). We also find that probabilistic constraints are more flexible than traditional hard constraints and can be used to specify the amount of violations allowed in a scenario. However, this flexibility comes at the cost of manual specification, since now we have an additional hyperparameter that must be specified in advance by the designer.

Chapter 6

Feature selection

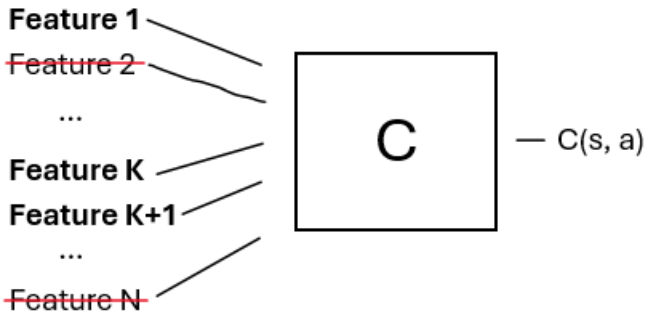


Figure 6.1: Underlying constraint function may just depend on a subset of input features.

In this chapter, we further discuss the challenge of high dimensional input spaces for ICL algorithms. In the ICL algorithms discussed so far, the input space of the constraint function is predefined according to domain knowledge. However, in many cases, it may be more desirable to determine procedurally which input features are necessary to predict the constraint function output.

6.1 Background: reduced input spaces

We will need to refer to reduced state spaces and reduced action spaces. Suppose the dimensionality of the state space is $m := dim(\mathcal{S})$ and the dimensionality of the action

space is $n := dim(\mathcal{A})$. Then we can consider the state-action space for an MDP $\mathcal{M}$ to be:

$$\mathcal{F}(\mathcal{M}) := \mathcal{S} \times \mathcal{A} \tag{6.1}$$

This state-action space is composed of individual dimensions:

$$\mathcal{S} \times \mathcal{A} := (\times_{i=1}^{m} \mathcal{S}^i) \times (\times_{j=1}^{n} \mathcal{A}^j) \tag{6.2}$$

Here $\mathcal{S}^i$ and $\mathcal{A}^j$ are the reduced state space and reduced action space along the i-th and j-th dimension of the complete state space and complete action space respectively. For a complete input space $\mathcal{X}$, the reduced input space along the k-th dimension $\mathcal{X}^k$ consists only of the values that the k-th dimension or feature can assume. This set may be discrete or continuous.

Further, assume that the set $[q]$ is defined as:

$$[q] := \{1, 2, 3, \cdots, q\} \tag{6.3}$$

Then, we define a reduced input space for some $Q \subseteq [dim(\mathcal{S}) + dim(\mathcal{A})]$ as:

$$\mathcal{F}(\mathcal{M})^Q = \times_{i \in Q} \mathcal{F}(\mathcal{M})^i \tag{6.4}$$

This is akin to selecting the features in $\mathcal{F}(\mathcal{M})$ that belong to the indices in Q.

Further, we also define a reduced constraint function c^Q, where $Q \subseteq [dim(\mathcal{S}) + dim(\mathcal{A})]$ (see Section 6.1) to be a constraint function with a reduced input space, i.e.

$$c^Q : \mathcal{F}(\mathcal{M})^Q \to \mathbb{R}^+ \tag{6.5}$$

Similarly, we can also define an ICL procedure that can learn a reduced constraint function $c^Q; Q \subseteq [dim(\mathcal{S}) + dim(\mathcal{A})]$ (see Equation (6.5)):

$$\text{ICL}_Q(r, \mathcal{D}; \mathcal{D} \sim \pi^*) := c^Q \implies \text{CRL}(r, \{c^Q\}; \Gamma_\beta) = \pi^* \tag{6.6}$$

This is equivalent to minimizing the dissimilarity between the expert dataset $\mathcal{D}_E$ and the agent's dataset $\mathcal{D}$, where the agent uses a reduced constraint function:

$$\arg\min_{c^Q} d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{CRL}_Q(r, \{c^Q\}; \Gamma_\beta)) \tag{6.7}$$

6.2 Learning a constraint with all input features

Before considering a constraint function which takes as input a reduced feature space, it might be useful to understand the constraint learned when the complete feature space is considered. Consider the example of a Mujoco environment like Ant or HalfCheetah where the true underlying constraint is to stay in the region $z \geq -1$. Here, the true constraint depends only on the z-coordinate.

We can learn a constraint with the complete input space using the expert data $\mathcal{D}_E$, i.e.

$$\tilde{c} = \text{ICL}(r, \mathcal{D}_E; \mathcal{D}_E \sim \pi^E) \tag{6.8}$$

This procedure uses the complete input space, thus it may or may not produce a constraint that looks like the true underlying constraint. Now, we can learn a policy:

$$\pi^* = \text{CRL}(r, \tilde{c}) \tag{6.9}$$

We can then generate a nominal dataset $\mathcal{D} \sim \pi^*$. After discretizing the z-coordinate space, we can plot the recovered constraint function, averaged across various points (or state-action pairs) lying in each discrete interval. This way, we can visualize the recovered constraint function along the z-coordinate dimension. These plots can be found in Figure 6.2 and Figure 6.3.

As can be seen from these figures, the learned constraint does not resemble a step function at $z = -1$. So, ICL with all input features learns a different constraint than expected. In order to test if this constraint is an alternative constraint (Section 2.9.1), we report the NAD values (explained in Section 4.3.4) for two policies, one learned by using all the input features, and another learned by using the reduced input space with just one dimension (z-coordinate). We report these values in Table 6.1. We find that the NAD values for the reduced input space are lower than the NAD values for the complete input space, which means we learn a better policy with a reduced input space.

Table 6.1: NAD values with and without all input features (std. error; lower is better)

Environment	NAD	NAD (all features)
HalfCheetah-Constrained	3.43 ± 0.70	5.20 ± 1.70
Ant-Constrained	2.49 ± 0.33	2.95 ± 0.79

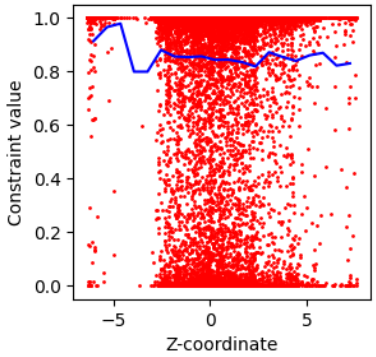


Figure 6.2: Learned constraint for Ant-Constrained environment (scatter points correspond to nominal data, blue line corresponds to the average constraint value)

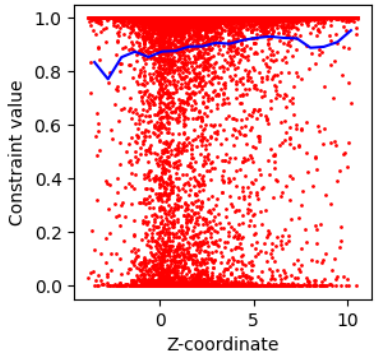


Figure 6.3: Learned constraint for HalfCheetah-Constrained environment (scatter points correspond to nominal data, blue line corresponds to the average constraint value)

We find that these recovered constraints do not resemble the true underlying function in a meaningful way. Additionally, the policy learned using the entire input space is worse than the policy learned using the reduced input space (i.e. just the z-coordinate). This rules out the case that ICL is finding an alternative constraint (Section 2.9.1), since any alternative constraint would still have a policy close to the expert.

The main reason behind the mismatch between recovered and true constraints is overfitting. As the number of input dimensions increase, it it less and less likely that we have sufficient data per dimension, due to which the learned constraint function is less likely to be the true constraint or an alternative constraint. In principle, we should be able to find either the true constraint or an alternative constraint with infinitely many demonstrations, but in practice we only have finitely many demonstrations.

This experiment motivates the need for feature selection, i.e. a way to reduce the input space size for ICL algorithms, since running ICL with the right reduced input space is better than running it will all features.

6.3 Using a neural network to construct features

It is natural to use a neural network to automatically reduce the input space into embeddings [61, 51, 52]. We do not use such a strategy because:

- the learned embeddings may not be necessarily interpretable
- a model learned with these embeddings as input features doesn't generalize to unseen data [13, 14, 15]

6.4 Feature selection objective

ICL algorithms learn a constraint function from expert data. In the context of ICL, the objective of feature selection is to determine the input features necessary to predict the learned constraint function value, given access to an expert dataset ($\mathcal{D}_E$) and a nominal dataset ($\mathcal{D}_N$). The nominal dataset contains non-expert like behaviour and it may be generated in different ways, depending on the algorithm used for feature selection. We describe this later.

6.4.1 Difference from feature selection in classification

The primary difference between feature selection in the context of ICL and in the context of classification is that for classification, the label variable Y is given and well defined for every example input X. For ICL, the label variable corresponds to the learned constraint function value (for the reduced feature space) $c^*(X)$, and since this needs to be learned with ICL, it is not known in advance.

6.4.2 Dataset generation

Since the learned constraint function c^* is not known in advance, we need to use a different strategy to come up with the label variables needed for feature selection. For this, we can argue that the learned constraint function distinguishes between expert and nominal data, i.e. it allows expert behaviour (label of 0) and disallows nominal behaviour (label of 1). In reality, this label is more of a continuous value in $[0, 1]$ where 0 corresponds to expert behaviour and 1 corresponds to completely non-expert like behaviour. We'll use a

simplifying assumption that the label is either 0 or 1 depending on whether the behaviour belongs to the expert dataset or the nominal dataset.

For an expected constraint formulation, since we have two batches of trajectories (or two datasets), we'll label the entire expert dataset as 0 and the entire nominal dataset as 1. This is too few examples (2 examples to be specific) for feature selection. The example set would be:

$$\{(\mathcal{D}_E, 0), (\mathcal{D}_N, 1)\} \tag{6.10}$$

For a hard constraint formulation, we can label each trajectory in the expert dataset as 0 and each trajectory in the nominal dataset as 1. This is still not too many examples ($2K$ examples to be specific, where $K = |D_E| = |D_N|$), and additionally, since each trajectory consists of a variable number of state-action pairs, each example will have a variable sized input space. In this case, the example set would be:

$$\{(\tau_E, 0); \forall \tau_E \in \mathcal{D}_E\} \cup \{(\tau_N, 1); \forall \tau_N \in \mathcal{D}_N\} \tag{6.11}$$

Therefore, we need to consider an instantaneous constraint formulation, where we consider that each state-action pair in the expert dataset is allowed and each state-action pair in the nominal dataset is disallowed. This is the *other assumption* that we need to make in order to generate a sufficient number of examples for feature selection ($2K \cdot \mathcal{O}(T)$ examples to be specific, where T is the horizon for each trajectory). Moreover, in this formulation, the input space size is fixed to be the size of the combined state-action space. For this case, the example set would be as follows:

$$\{([S_t^i A_t^i], 0); \forall (S_t^i, A_t^i) \in \tau_i, \forall \tau_i \in \mathcal{D}_E\} \cup \{([S_t^j A_t^j], 1); \forall (S_t^j, A_t^j) \in \tau_j, \forall \tau_j \in \mathcal{D}_N\} \tag{6.12}$$

6.5 Baselines: mutual information based methods

Using the approach described in the previous section and under the two stated assumptions, we can construct a dataset of examples with corresponding labels. Then, we can use any of the known techniques for feature selection in classification to do feature selection in the context of ICL. We use filter based methods for this task since they are pretty fast to apply. In practice however, these methods fail for the primary set of environments (i.e. Gridworld, CartPole and HighD-S, see Section 6.8.5). Thus, we use these methods as baselines.

We discuss these baseline techniques as follows.

6.5.1 Conditional infomax feature extraction (CIFE)

Conditional infomax feature extraction (CIFE) [71] sequentially extracts features by optimizing the following objective:

$$X^{t^*} = \arg\max_t \left\{ I(X^t; Y) - \sum_{u=1}^{t-1} R_c(X^u; X^t) \right\} \tag{6.13}$$

Here, maximizing the first term chooses a feature that is helpful in predicting the label Y, whereas minimizing the second term reduces the class relevant redundancy R_c i.e.

$$R_c(X^1; X^2) := I(X^1; X^2) - I(X^1; X^2|Y) \tag{6.14}$$

6.5.2 Double input symmetrical relevance (DISR)

Double input symmetrical relevance (DISR) [93] uses the notion of symmetrical relevance which is equivalent to normalized mutual information i.e.

$$\mathrm{SR}(X; Y) = \frac{I(X; Y)}{H(X; Y)} \tag{6.15}$$

Then, the objective is to maximize the overall joint symmetrical relevance of selected and unselected features, w.r.t. the label Y. More specifically, if the set of selected features is X^S and the set of unselected features is X^{-S}, then the objective is:

$$X^{t^*} = \arg\max_{X^t \in X^{-S}} \left\{ \sum_{X^u \in X^S} \mathrm{SR}(X^{t,u}; Y) \right\} \tag{6.16}$$

6.5.3 Conditional mutual information maximization (CMIM)

Conditional mutual information maximization (CMIM) [39] maximizes the minimum mutual information of the next feature w.r.t label Y conditional on the previously selected features:

$$X^{t^*} = \arg \max_{X^t \in X^{-S}} \min_{X^u \in X^S} I(X^t; Y|X^u) \tag{6.17}$$

6.5.4 Maximum relevance minimum redundancy (MRMR)

Max relevance min redundancy (MRMR) [108] is similar to CIFE except that the redundancy is computed as an average of individual mutual information terms, where the mutual information is calculated between the variable to be selected and a variable that has been previously selected:

$$X^{t^*} = \arg\max_{X^t \in X^{-S}} \left\{ I(X^t; Y) - \frac{1}{|X^S|} \sum_{X^u \in X^S} I(X^t; X^u) \right\} \tag{6.18}$$

6.6 Our approach

6.6.1 Motivation

Since traditional feature learning algorithms do not work well in the context of feature selection for ICL, we propose an alternative approach to feature selection, inspired by the functionality of ICL algorithms.

We note that ICL algorithms learn a constraint from demonstrations, i.e.

$$\tilde{c} = \text{ICL}(r, \mathcal{D}_E; \mathcal{D}_E \sim \pi^E) \tag{6.19}$$

The goodness of the constraint is judged by how similar the accrual of the agent is to the accrual of the expert, i.e. if we generate the agent dataset

$$D_N \sim \pi^* = \text{CRL}(r, \tilde{c}) \tag{6.20}$$

Then, if the following value is small enough (d_{dis} is a notion of distance between dataset accruals or visitations, Section 2.1.3), then the learned constraint is either the true constraint or an alternative constraint (Section 2.9.1):

$$d_{dis}(D_E, D_N) \approx 0 \tag{6.21}$$

We can further assume the accrual dissimilarity $d_{dis}(D_E, D_N)$ to be correlated with individual dissimilarities $d_{dis}(D_E^i, D_N^i), \forall i$, where D_E^i and D_N^i are the marginal datasets along

feature i. In fact, it is reasonable to assume that a larger individual dissimilarity correlates more with the overall accrual dissimilarity. Thus, the magnitude of the individual dissimilarity is correlated with the quality of the learned constraint. This means that the degree of feature mismatch is correlated with the learned constraint function value. More specifically, when the marginal dissimilarity is low, the marginal distributions don't change much, however, the constraint function values change (expert vs non-expert data). Thus in this setting, the specific feature under consideration has low effect on the constraint function value.

This mismatch can be measured using a simple statistical test, described next.

6.6.2 Kolmogorov-Smirnov two sample test

We borrow a simple non parametric statistical test from the hypothesis testing literature to ascertain the mismatch between the marginal distributions of the individual features.

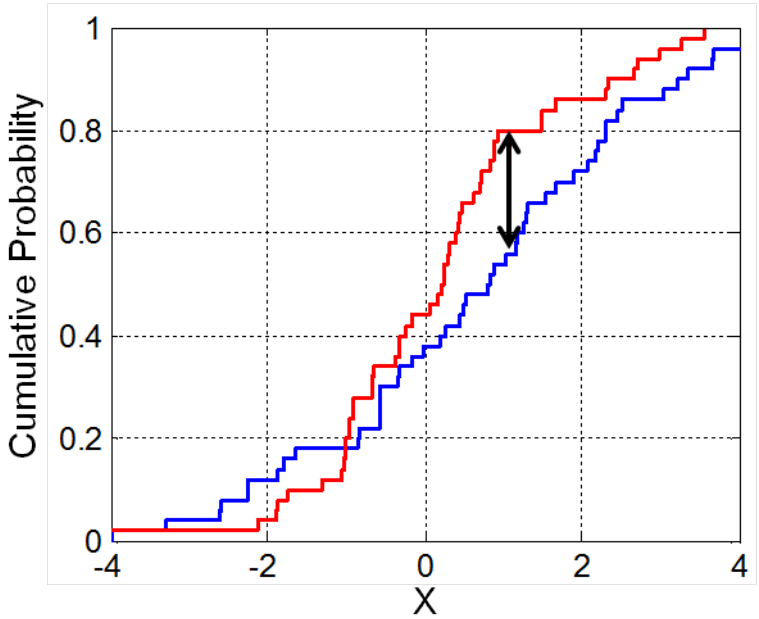


Figure 6.4: Computation of KS2ST statistic.

Given samples from two distributions (n from the first distribution and m from the second distribution), the KS2ST constructs empirical CDFs F_1^n, F_2^m for both the distributions

which are then compared. The KS statistic is then defined as the maximum difference between these empirical CDFs.

$$D_{n,m} = \max_x |F_1^n(x) - F_2^m(x)| \tag{6.22}$$

The *null hypothesis* is that both the samples belong to the same distribution. It is rejected if the statistic is fairly high, i.e.

$$D_{n,m} > \sqrt{-\ln\left(\frac{\alpha}{2}\right) \cdot \frac{1}{2}} \cdot \sqrt{\frac{n+m}{n \cdot m}} \tag{6.23}$$

For our experiments, we use $\alpha = 0.01$, which is common in the hypothesis testing literature.

6.7 Algorithms

6.7.1 Without ICL

Given the expert dataset D_E and a nominal dataset D_N (D_N is generated with the reward, but without any constraint; labels are generated as per the technique explained in Section 6.4.2), we can:

- rank features using a feature selection algorithm (in Section 6.5)
- rank features using our method (in Section 6.6)

The algorithm stops when the stopping criterion is met, i.e. when the p-value of the selected feature drops below 0.01 (this is standard in hypothesis testing literature).

6.7.2 With ICL

The algorithm with ICL is provided in Algorithm 7. This algorithm is inspired by the feature selection objective (Equation (6.7)):

$$\arg\min_{Q \subseteq [dim(\mathcal{S})+dim(\mathcal{A})]} d_{dis}(\mathcal{D}, \mathcal{D}_E; \mathcal{D} \sim \text{CRL}(r, \{\text{ICL}_Q(r, \mathcal{D}_E)\}; \Gamma_\beta))$$

Algorithm 6 Feature selection without ICL

procedure FEATURE-SELECT-NO-ICL(expert dataset D_E, reward r, entire feature set $= \mathbb{X}$)
    X_S = features selected so far $= \varnothing$
    $D_N = \text{CRL}(r, \{\})$
    Create dataset D by labeling $(S, A) \in D_E, (S, A) \in D_N$ as per Section 6.4.2
    for iteration $= 1$ to $\mathbb{X}$.size **do**
        Select next feature $X^{i^*} \in \mathbb{X}$ using a baseline or our method (arguments X_S, D)
        $X_S = X_S \cup \{X^{i^*}\}$
        Stop if stopping criterion is met
    end for
end procedure

The idea is to greedily and repeatedly select a feature to add to the list of selected features. A nominal dataset is computed each time using the constraint recovered from ICL (reduced input space) which is used to select the next feature. This nominal dataset would have low mismatch for the feature selected previously, since as we add features, the learned constraint and policy should become better and better. The algorithm stops when the stopping criterion is met, i.e. when the p-value of the selected feature becomes statistically insignificant, i.e. it goes above 0.01. For selecting the first feature, this is equivalent to the "without ICL" algorithm (Section 6.7.1).

6.7.3 Mitigating information overlap

It is likely that several input features have information overlap, i.e. they have redundant information. In such a case, the algorithm with ICL prioritizes the selection of features with less information overlap, allowing the informative subset of features to remain small. Thus the input feature with repeated information is selected much later.

We conduct an experiment to ascertain the effect of having overlapping features in the input space. Specifically, we construct a variant of the CartPole environment with 6 features (5 state features + 1 action feature), where the extra state feature corresponds to a bogus feature that assumes the same value as x feature, and is therefore perfectly correlated with x. In this case, KS2ST without ICL should select both x and x (bogus) one after the another, while the ICL variant (Section 6.7.2) should select x (bogus) feature much later, since once we have selected x, any overlapping feature will be ignored. This is indeed what we observe in our experiment (Table 6.2). Specifically, we find that without ICL,

Algorithm 7 Feature selection with ICL

procedure FEATURE-SELECT(expert dataset D_E, reward r, entire feature set = $\mathbb{X}$)
  X_S = features selected so far = $\varnothing$
  for iteration = 1 to $\mathbb{X}$.size **do**
    if X_S.size = 0 **then**
      $D_N = \text{CRL}(r, \{\})$
    else
      $c^* = \text{ICL}_S(r, D_E)$
      $D_N = \text{CRL}(r, \{c^*\}; \Gamma_\beta)$
    end if
    Create dataset D by labeling $(S, A) \in D_E, (S, A) \in D_N$ as per Section 6.4.2
    Select next feature $X^{i^*} \in \mathbb{X}$ using a baseline or our method (arguments X_S, D)
    $X_S = X_S \cup \{X^{i^*}\}$
    Stop if stopping criterion is met
  end for
end procedure

the median position of selection of x is #1, and x (bogus) is #2, whereas with ICL, the median position of selection of x is #1 and x (bogus) is #4.

Table 6.2: KS2ST for overlapping features

Seed	Without ICL	With ICL
1	x: #2, x (bogus): #3	x: #4, x (bogus): #5
2	x: #1, x (bogus): #2	x: #1, x (bogus): #2
3	x: #1, x (bogus): #2	x: #1, x (bogus): #5
4	x: #1, x (bogus): #2	x: #1, x (bogus): #4
5	x: #1, x (bogus): #2	x: #1, x (bogus): #3

6.8 Experiments

6.8.1 Experiment design

We wish to answer the following questions in our experiments:

- How does our method perform w.r.t. feature selection methods from classification?

- Can our method perform well for a complex real world environment, for example, a scenario in autonomous driving?
- How does our method perform in large state-action spaces (e.g. in Mujoco environments)?
- Can our method select the right subset of features?

To answer these questions, we perform experiments with the following environments.

6.8.2 Environments

We divide the environments for our experiments into two sets. For the primary set of environments, we conduct experiments to compare the performance of our methods against the baselines, i.e. feature selection methods from classification. As we will see, even simple environments are sufficient to prove the superiority of our method. For the secondary set of environments, we compare the without and with ICL based algorithms (Algorithm 6 and Algorithm 7) and see their performances in large state action spaces.

Primary environments:

1. **Gridworld** environment: This environment is the same as the one used in IPCL experiments. The state space consists of two features, x and y position of the agent. There are 8 possible discrete actions corresponding to the nominal and diagonal directions.
2. **CartPole** environment: This environment is also the same as the one used in IPCL experiments. The state space and action spaces are the same as the original CartPole environment provided by OpenAI [19].
3. **A simplified HighD** environment (HighD-S) [63]: This is the HighD car following environment described in ICL and IPCL experiments, but with a reduced state action space. The state consists of only 4 features, i.e. gap to the car in the front, speed, and x/y positions of the car and the action consists only of one continuous feature i.e. acceleration of the car.

Secondary environments:

1. **HighD** environment [63]: This is the HighD environment described in ICL and IPCL experiments, but with the complete state action space. The state consists of 14 continuous as well as boolean features (7 car features and 7 predicates) and the action consists of two continuous features, i.e. acceleration of the car and rate of change of steering angle of the car.

2. **Mujoco** environments (Ant-Constrained, HalfCheetah-Constrained): These are the Mujoco environments described in ICL and IPCL experiments. The Ant environment has 121 combined state action features and the HalfCheetah environment has 24 combined state action features.

Since these environments have been previously described in ICL and IPCL experiments, we omit further description here, and refer the reader to Section 5.3.2 for a more detailed description of these environments.

6.8.3 Training setup

We run our experiments with a single seed for all environments, since running with multiple seeds is computationally expensive for the with ICL (Algorithm 7) variant.

We run the Mujoco experiments with 5 seeds to report the mean and median position when z-coordinate is selected. We terminate these experiments when the z-coordinate is selected, otherwise the algorithm will take too much time (HalfCheetah-Constrained has 24 input features and Ant-Constrained has 121 input features).

6.8.4 Ground truth

We provide the ground truth for the feature selection experiments in Table 6.3. For the feature ranking experiments, the features listed in Table 6.3 can come in any order, followed by the rest of the features.

Table 6.3: Feature selection ground truth

Environment	Correct feature selection subset
Gridworld	y, x
CartPole	x
HighD-S	acc., gap, speed
Mujoco Ant	z-coordinate
Mujoco HalfCheetah	z-coordinate
HighD	gap, gap-parsed, speed, a_t, a_{t+1}

6.8.5 Feature ranking for primary environments

The results for feature ranking for the primary environments are provided in Tables 6.4, 6.5 and 6.6. We can see from these experiments that while the baselines don't work well with the HighD-S environment, our method works well with all the environments.

Table 6.4: Gridworld feature ranking results. Blue indicates a relevant feature, red indicates an irrelevant feature that appears before a relevant feature, and black indicates an irrelevant feature that appears after the relevant features. Red features correspond to mistakes.

Method	Order of feature selection
CIFE	y, x, action
DISR	y, x, action
CMIM	y, x, action
MRMR	y, **action**, x
KS2ST (without ICL)	y, x, action
KS2ST (with ICL)	y, x, action

6.8.6 HighD environment

The results for the HighD environment (with and without ICL) are provided in Tables 6.7 and 6.8. While the method without ICL works well, the method with ICL makes a few mistakes, but discovers all the 5 important features in the first 7 selected features.

6.8.7 Mujoco environments

We further apply the KS2ST strategy to two Mujoco environments, Ant and HalfCheetah. For the ICL variant (Section 6.7.2), we only run the algorithm until we find the

Table 6.5: CartPole feature ranking results. Blue indicates a relevant feature, red indicates an irrelevant feature that appears before a relevant feature, and black indicates an irrelevant feature that appears after the relevant features. Red features correspond to mistakes.

Method	Order of feature selection
CIFE	x, θ, action, dx/dt, dθ/dt
DISR	x, action, dθ/dt, dx/dt, θ
CMIM	x, θ, dθ/dt, dx/dt, action
MRMR	x, θ, action, dx/dt, dθ/dt
KS2ST (without ICL)	x,θ, dx/dt, dθ/dt, action
KS2ST (with ICL)	x, θ, dθ/dt, dx/dt, action

Table 6.6: HighD-S feature ranking results. Blue indicates a relevant feature, red indicates an irrelevant feature that appears before a relevant feature, and black indicates an irrelevant feature that appears after the relevant features. Red features correspond to mistakes.

Method	Order of feature selection
CIFE	**y**, gap, acc., speed, x
DISR	**y**, acc., speed, **x**, gap
CMIM	acc., **y**, gap, speed, x
MRMR	acc., speed, **x**, gap, y
KS2ST (without ICL)	speed, acc., gap, y, x
KS2ST (with ICL)	speed, acc., gap, x, y

Table 6.7: KS2ST HighD results (without ICL)

Order of feature selection	Selected feature
1	gap
2	speed
3	a_{t+1}
4	a_t
5	gap-parsed
6	y pos
7	collision
...	...
16	$\dot{\psi}_{t+1}$

Table 6.8: KS2ST HighD results (with ICL)

Order of feature selection	Selected feature
1	gap
2	a_{t+1}
3	**y pos**
4	speed
5	**x pos**
6	gap-parsed
7	a_t
...	...
16	$\dot{\psi}_{t+1}$

z-coordinate. In both these environments, the expert data is generated by an underlying constraint function that depends only on the z-coordinate (1st feature, starting from index 1). Thus, in the order of feature selection, this feature must be selected first. The Ant environment has 121 features (113 state features + 8 action features), whereas HalfCheetah has 24 features (18 state features + 6 action features). Here, we report the position of z-coordinate in the order of feature selection, since enumerating the entire list takes up a considerable amount of space. Our results are available in Tables 6.9 and 6.10.

We empirically observe that with ICL, the mean position of the z-coordinate in the order of feature selection is usually lower than without ICL, which affirms the improvement offered by the with ICL algorithm variant. Moreover, in HalfCheetah, the median position is #1 across 5 seeds which means that the most important feature is chosen first. However, in Ant, the median position is #6, which means that the most important feature is not selected first. This could be because of the high dimensionality of the input space (121 features).

6.8.8 Feature selection experiments

We also choose a stopping criterion for our feature selection experiments. Specifically, we stop the feature selection (with ICL) algorithm when we find the mismatch is insignificant i.e. p-value is more than 0.01. This value is standard in hypothesis testing literature. We report our results in Table 6.11. Overall, except the HighD-S environment, our method is able to reduce the input space considerably and makes few mistakes. This is acceptable since the primary goal is to reduce the number of features without missing any relevant

Table 6.9: KS2ST Mujoco Ant results

Algorithm	Statistic	Position of z-coordinate (positions for 5 seeds)
Without ICL	Median	#6 (#3, #5, #6, #15, #21)
	Mean	#10 (#3, #5, #6, #15, #21)
With ICL	Median	#6 (#5, #5, #6, #7, #7)
	Mean	#6 (#5, #5, #6, #7, #7)

Table 6.10: KS2ST Mujoco HalfCheetah results

Algorithm	Statistic	Position of z-coordinate (positions for 5 seeds)
Without ICL	Median	#1 (#1, #1, #1, #3, #11)
	Mean	#3.4 (#1, #1, #1, #3, #11)
With ICL	Median	#1 (#1, #1, #1, #2, #7)
	Mean	#2.4 (#1, #1, #1, #2, #7)

feature.

Here, for the HalfCheetah and Ant environments, FN refers to the N-th feature.

Table 6.11: Feature selection results

Environment	Selected vs total features	List of selected features
Gridworld	**2** / 3	y, x
CartPole	**2** / 5	x, **dθ/dt**
HighD-S	**5** / 5	speed, acc, gap, **x pos**, **y pos**
HighD	**7** / 16	gap, a_{t+1}, **y pos**, speed, **x pos**, gap-parsed, a_t
HalfCheetah	**6** / 24	F0, **F6, F2, F1, F5, F20**
Ant	**4** / 121	**F2, F12, F3**, F0

6.9 Summary

In this chapter, we discussed the problem of feature selection in the context of inverse constraint learning. This problem is different and more challenging than the feature selection problem in classification because the constraint function value is not known and has to be simultaneously learned in parallel with feature selection. We provided a method for dataset construction in such a case. Later, we used a simple test from hypothesis testing literature and devised a method to select features using the dataset previously constructed, motivated by the functionality of ICL algorithms. We then provide algorithms that use

this method to select features. Finally, we conducted experiments to validate the proposed method. Our experiments indicate that feature selection techniques for classification fail with simple environments in the context of ICL. Our method can select the right features with simple as well as complex environments with few mistakes, thereby being able to reduce huge state-action spaces into much smaller state-action spaces, amenable to constraint learning.

Chapter 7

Summary

In this work,

1. We propose new techniques to learn expected and probabilistic constraints from demonstrations. Specifically, we propose an approach that can recover expected cumulative constraints common in CMDPs. We also discuss the problem of finding a probabilistic or chance constraint, which may be desired in many circumstances where we wish to have a guarantee over the probability distribution of constraint returns. Our methods provide a principled way to learn such a probabilistic constraint.

2. Our constraint learning experiments show that ICL is able to learn a sharp and accurate expected constraint, and that the learned policy is close to the expert policy. For learning a probabilistic constraint, we propose two methods. The method which reuses existing ICL techniques performs worse than IPCL in terms of learning an accurate constraint. Further, it also learns a more conservative constraint than needed. On the other hand, IPCL learns a constraint that "just" satisfies the probabilistic threshold by a smaller margin.

3. Further, we define the problem of feature selection for ICL algorithms, i.e. identifying the features relevant for predicting the constraint function output, to be used as an apriori specification for running ICL. In order to do this feature selection, we discuss various mutual information based techniques in the literature, as well as propose using the KS2ST from hypothesis testing literature.

4. Our experiments for feature selection show that while mutual information based techniques are unable to perform well with simple environments, the KS2ST based

approach works well for simple as well as complex environments. We also discuss and demonstrate an ICL based variant of the feature selection algorithm that is able to select features by mitigating information overlap.

7.1 Limitations

Our methods have several limitations:

1. The proposed techniques assume that the reward function is known and can only learn a single constraint function. The challenge with simultaneously learning the reward function as well as multiple constraints is that there can be many equivalent configurations of reward and constraint functions that can generate the same trajectories (i.e. *unidentifiability*, see Section 2.9.1).

2. It is possible that expert demonstrations are sub-optimal. In that case, we hypothesize that our method would learn other constraints such that the provided demonstrations become optimal for these constraints.

3. Our approach requires several outer iterations of forward CRL and constraint adjustment, and as a result, it should in principle require more training time than the existing baselines. Similarly, our feature learning method (with ICL) requires several iterations of ICL, and is therefore, even more computationally expensive.

4. For learning a chance constraint, while our method allows learning a constraint with a probabilistic guarantee, it also adds a new hyperparameter δ, which may vary between environments and needs to be set in advance by the designer.

5. For feature selection, it is difficult to assess the performance of our method when the ground truth feature subset is not known, especially when the input space is huge, which is true for several real world settings.

7.2 Future work

There are several avenues for further research:

1. For learning rewards and/or multiple constraints, we need a way to specify preference for specific combinations of reward and constraint functions. Understanding and quantifying the unidentifiability in the constraint learning case is also an important alternative direction of future work.

2. We also hope to extend our work to handle demonstrations from multiple experts and incorporate entropy regularization into our framework for its benefits.

3. We hope to improve our feature learning algorithm to handle large input spaces more accurately (e.g. Ant-Constrained and HalfCheetah-Constrained).

7.3 Conclusions

In this work, we explored methods to learn constraints from demonstrations. Unlike prior work which focuses on learning hard constraints [25, 120, 105, 90, 88] or task specific constraints [5, 109, 72, 73], we propose methods to learn an expected constraint and a probabilistic constraint from expert demonstrations. We also provide a method to reduce large input spaces to make constraint learning more amenable.

The main advantage of using expected or probabilistic constraints can be explained as follows. While hard constraints do not allow any violations, expected constraints allow for some violations and probabilistic constraints allow the degree of acceptable violations to be set by the practitioner. This in turn leads to constraints that function well in the real world, since such constraints better account for sensor noise and measurement error in the real world. There also exist numerous applications that would benefit from learning an expected constraint or a probabilistic constraint. These applications span across inventory planning [24], robotics [70, 117, 128, 2, 49, 29, 86, 106], autonomous driving [28, 29, 87, 54, 100, 107], video compression [89], power systems [45], war assessment [58] and more.

Newer work in constraint learning focuses on learning constraints under uncertainty [75, 134], or learning both reward and constraints together [81, 82, 83]. There still remain open problems of unidentifiability, dynamic constraint learning, generalizability and extension to multi-agent and partially-observable scenarios. These are exciting topics to explore in future work.

References

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning, page 1, 2004.

[2] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In International conference on machine learning, pages 22–31. PMLR, 2017.

[3] A. Agarwal, N. Jiang, S. M. Kakade, and W. Sun. Reinforcement learning: Theory and algorithms. 2021. URL https://rltheorybook. github. io, 2022.

[4] E. Altman. Constrained Markov decision processes: stochastic modeling. Routledge, 1999.

[5] L. Armesto, J. Bosga, V. Ivan, and S. Vijayakumar. Efficient learning of constraints and generic null space policies. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 1520–1526. IEEE, 2017.

[6] S. Arora and P. Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. Artificial Intelligence, 297:103500, 2021.

[7] M. Baert, P. Mazzaglia, S. Leroux, and P. Simoens. Maximum causal entropy inverse constrained reinforcement learning. arXiv preprint arXiv:2305.02857, 2023.

[8] J. Bagnell, J. Chestnutt, D. Bradley, and N. Ratliff. Boosting structured prediction for imitation learning. Advances in Neural Information Processing Systems, 19, 2006.

[9] D. Baimukashev, G. Alcan, and V. Kyrki. Automated feature selection for inverse reinforcement learning. arXiv preprint arXiv:2403.15079, 2024.

[10] D. P. Bertsekas. Constrained optimization and Lagrange multiplier methods. Academic press, 2014.

[11] S. Bhatnagar and K. Lakshmanan. An online actor–critic algorithm with function approximation for constrained markov decision processes. Journal of Optimization Theory and Applications, 153:688–708, 2012.

[12] L. Blackmore, H. Li, and B. Williams. A probabilistic approach to optimal robust path planning with obstacles. In 2006 American Control Conference, pages 7–pp. IEEE, 2006.

[13] A. Bobu, A. Bajcsy, J. F. Fisac, S. Deglurkar, and A. D. Dragan. Quantifying hypothesis space misspecification in learning from human–robot demonstrations and physical corrections. IEEE Transactions on Robotics, 36(3):835–854, 2020.

[14] A. Bobu, M. Wiggert, C. Tomlin, and A. D. Dragan. Feature expansive reward learning: Rethinking human input. In Proceedings of the 2021 ACM/IEEE international conference on human-robot interaction, pages 216–224, 2021.

[15] A. Bobu, M. Wiggert, C. Tomlin, and A. D. Dragan. Inducing structure in reward learning by learning features. The International Journal of Robotics Research, 41(5):497–518, 2022.

[16] S. Bohez, A. Abdolmaleki, M. Neunert, J. Buchli, N. Heess, and R. Hadsell. Value constrained model-free continuous control. arXiv preprint arXiv:1902.04623, 2019.

[17] V. S. Borkar. An actor-critic algorithm for constrained markov decision processes. Systems & control letters, 54(3):207–213, 2005.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[20] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján. Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. The journal of machine learning research, 13(1):27–66, 2012.

[21] S. Calinon and A. 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, 2008.

[22] M. Castillo-Lopez, P. Ludivig, S. A. Sajadi-Alamdari, J. L. Sanchez-Lopez, M. A. Olivares-Mendez, and H. Voos. A real-time approach for chance-constrained motion planning with dynamic obstacles. IEEE Robotics and Automation Letters, 5(2):3620–3625, 2020.

[23] G. Chandrashekar and F. Sahin. A survey on feature selection methods. Computers & electrical engineering, 40(1):16–28, 2014.

[24] Y. Chen, J. Dong, and Z. Wang. A primal-dual approach to constrained markov decision processes. arXiv preprint arXiv:2101.10895, 2021.

[25] G. Chou, D. Berenson, and N. Ozay. Learning constraints from demonstrations. In International Workshop on the Algorithmic Foundations of Robotics, pages 228–245. Springer, 2018.

[26] G. Chou, D. Berenson, and N. Ozay. Uncertainty-aware constraint learning for adaptive safe motion planning from demonstrations. In Conference on Robot Learning, pages 1612–1639. PMLR, 2021.

[27] G. Chou, N. Ozay, and D. Berenson. Learning parametric constraints in high dimensions from demonstrations. In Conference on Robot Learning, pages 1211–1230. PMLR, 2020.

[28] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. Advances in neural information processing systems, 31, 2018.

[29] Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh. Lyapunov-based safe policy optimization for continuous control. arXiv preprint arXiv:1901.10031, 2019.

[30] P. De Haan, D. Jayaraman, and S. Levine. Causal confusion in imitation learning. Advances in neural information processing systems, 32, 2019.

[31] F. Ding and Y. Xue. X-men: Guaranteed xor-maximum entropy constrained inverse reinforcement learning, 2022.

[32] Y. Ding, L. Zhang, C. Huang, and R. Ge. Two-stage travel itinerary recommendation optimization model considering stochastic traffic time. Expert Systems with Applications, 237:121536, 2024.

[33] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. arXiv preprint arXiv:1605.08803, 2016.

[34] P. L. Donti, D. Rolnick, and J. Z. Kolter. Dc3: A learning method for optimization with hard constraints. In International Conference on Learning Representations, 2020.

[35] J. R. Doppa, J. Yu, P. Tadepalli, and L. Getoor. Learning algorithms for link prediction based on chance constraints. In Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part I 21, pages 344–360. Springer, 2010.

[36] W. Fan, K. Liu, H. Liu, P. Wang, Y. Ge, and Y. Fu. Autofs: Automated feature selection via diversity-aware interactive reinforcement learning. In 2020 IEEE International Conference on Data Mining (ICDM), pages 1008–1013. IEEE, 2020.

[37] J. Fischer, C. Eyberg, M. Werling, and M. Lauer. Sampling-based inverse reinforcement learning algorithms with safety constraints. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 791–798, 2021.

[38] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer. Pot: Python optimal transport. Journal of Machine Learning Research, 22(78):1–8, 2021.

[39] F. Fleuret. Fast binary feature selection with conditional mutual information. Journal of Machine learning research, 5(Nov):1531–1555, 2004.

[40] G. Forman. An extensive empirical study of feature selection metrics for text classification. Journal of machine learning research, 3(Mar):1289–1305, 2003.

[41] S. Gao, G. Ver Steeg, and A. Galstyan. Variational information maximization for feature selection. Advances in neural information processing systems, 29, 2016.

[42] A. Gaurav, K. Rezaee, G. Liu, and P. Poupart. Learning soft constraints from constrained expert demonstrations. In The Eleventh International Conference on Learning Representations, 2023.

[43] P. Geibel and F. Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. Journal of Artificial Intelligence Research, 24:81–108, 2005.

[44] A. Geletu, M. Klöppel, H. Zhang, and P. Li. Advances and applications of chance-constrained approaches to systems optimisation under uncertainty. International Journal of Systems Science, 44(7):1209–1232, 2013.

[45] X. Geng and L. Xie. Data-driven decision making in power systems with probabilistic guarantees: Theory and applications of chance-constrained optimization. Annual reviews in control, 47:341–363, 2019.

[46] A. Glazier, A. Loreggia, N. Mattei, T. Rahgooy, F. Rossi, and K. B. Venable. Making human-like trade-offs in constrained environments by learning from demonstrations. arXiv preprint arXiv:2109.11018, 2021.

[47] J. Grosso, C. Ocampo-Martínez, V. Puig, and B. Joseph. Chance-constrained model predictive control for drinking water networks. Journal of process control, 24(5):504–516, 2014.

[48] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, and A. Knoll. A review of safe reinforcement learning: Methods, theory and applications. arXiv preprint arXiv:2205.10330, 2022.

[49] M. Han, Y. Tian, L. Zhang, J. Wang, and W. Pan. Reinforcement learning control of constrained dynamic systems with uniformly ultimate boundedness stability guarantee. Automatica, 129:109689, 2021.

[50] L. Haug, S. Tschiatschek, and A. Singla. Teaching inverse reinforcement learners via features and demonstrations. Advances in Neural Information Processing Systems, 31, 2018.

[51] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In International conference on learning representations, 2017.

[52] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. science, 313(5786):504–507, 2006.

[53] J. Ho and S. Ermon. Generative adversarial imitation learning. Advances in neural information processing systems, 29, 2016.

[54] X. Huang, A. Jasour, M. Deyo, A. Hofmann, and B. C. Williams. Hybrid risk-aware conditional planning with applications in autonomous vehicles. In 2018 IEEE Conference on Decision and Control (CDC), pages 3608–3614. IEEE, 2018.

[55] S. Huh and I. Yang. Safe reinforcement learning for probabilistic reachability and safety specifications: A lyapunov-based approach. arXiv preprint arXiv:2002.10126, 2020.

[56] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei. Reward learning from human preferences and demonstrations in atari. Advances in neural information processing systems, 31, 2018.

[57] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1611.01144, 2016.

[58] Z. Jia, E. Ben-Michael, and K. Imai. Bayesian safe policy learning with chance constrained optimization: Application to military security assessment during the vietnam war. arXiv preprint arXiv:2307.08840, 2023.

[59] G. Kalweit, M. Huegle, M. Werling, and J. Boedecker. Deep inverse q-learning with constraints. Advances in Neural Information Processing Systems, 33, 2020.

[60] Y. Kim, W. N. Street, and F. Menczer. Feature selection in unsupervised learning via evolutionary search. In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 365–369, 2000.

[61] D. P. Kingma and M. Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.

[62] W. B. Knox, S. Hatgis-Kessell, S. Booth, S. Niekum, P. Stone, and A. Allievi. Models of human preference for learning reward functions. arXiv preprint arXiv:2206.02231, 2022.

[63] R. Krajewski, J. Bock, L. Kloeker, and L. 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.

[64] M. Kuderer, S. Gulati, and W. Burgard. Learning driving styles for autonomous vehicles from demonstration. In 2015 IEEE international conference on robotics and automation (ICRA), pages 2641–2646. IEEE, 2015.

[65] J. Lee, A. Balakrishnan, A. Gaurav, K. Czarnecki, and S. Sedwards. W ise m ove: A framework to investigate safe deep reinforcement learning for autonomous driving. In International Conference on Quantitative Evaluation of Systems, pages 350–354. Springer, 2019.

[66] S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. arXiv preprint arXiv:1805.00909, 2018.

[67] S. Levine, Z. Popovic, and V. Koltun. Feature construction for inverse reinforcement learning. Advances in neural information processing systems, 23, 2010.

[68] C. Li and D. Berenson. Learning object orientation constraints and guiding constraints for narrow passages from one demonstration. In International symposium on experimental robotics, pages 197–210. Springer, 2016.

[69] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu. Feature selection: A data perspective. ACM computing surveys (CSUR), 50(6):1–45, 2017.

[70] Q. Liang, F. Que, and E. Modiano. Accelerated primal-dual policy optimization for safe reinforcement learning. arXiv preprint arXiv:1802.06480, 2018.

[71] D. Lin and X. Tang. Conditional infomax learning: An integrated framework for feature extraction and fusion. In Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9, pages 68–82. Springer, 2006.

[72] H.-C. Lin, M. Howard, and S. Vijayakumar. Learning null space projections. In 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 2613–2619. IEEE, 2015.

[73] H.-C. Lin, P. Ray, and M. Howard. Learning task constraints in operational space formulation. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 309–315. IEEE, 2017.

[74] D. Lindner, X. Chen, S. Tschiatschek, K. Hofmann, and A. Krause. Learning safety constraints from demonstrations with unknown rewards. In International Conference on Artificial Intelligence and Statistics, pages 2386–2394. PMLR, 2024.

[75] G. Liu, Y. Luo, A. Gaurav, K. Rezaee, and P. Poupart. Benchmarking constraint inference in inverse reinforcement learning. In International Conference on Learning Representations (ICLR), 2023.

[76] G. Liu, S. Xu, S. Liu, A. Gaurav, S. G. Subramanian, and P. Poupart. A comprehensive survey on inverse constrained reinforcement learning: Definitions, progress and challenges. arXiv preprint arXiv:2409.07569, 2024.

[77] K. Liu, Y. Fu, P. Wang, L. Wu, R. Bo, and X. Li. Automating feature subspace exploration via multi-agent reinforcement learning. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 207–215, 2019.

[78] K. Liu, Y. Fu, L. Wu, X. Li, C. Aggarwal, and H. Xiong. Automated feature selection: A reinforcement learning perspective. IEEE Transactions on Knowledge and Data Engineering, 35(3):2272–2284, 2021.

[79] K. Liu, D. Wang, W. Du, D. O. Wu, and Y. Fu. Interactive reinforced feature selection with traverse strategy. Knowledge and Information Systems, 65(5):1935–1962, 2023.

[80] K. Liu, P. Wang, D. Wang, W. Du, D. O. Wu, and Y. Fu. Efficient reinforced feature selection via early stopping traverse strategy. In 2021 IEEE International Conference on Data Mining (ICDM), pages 399–408. IEEE, 2021.

[81] S. Liu and M. Zhu. Distributed inverse constrained reinforcement learning for multi-agent systems. Advances in Neural Information Processing Systems, 35:33444–33456, 2022.

[82] S. Liu and M. Zhu. Meta inverse constrained reinforcement learning: Convergence guarantee and generalization analysis. In The Twelfth International Conference on Learning Representations, 2023.

[83] S. Liu and M. Zhu. Learning multi-agent behaviors from distributed and streaming demonstrations. Advances in Neural Information Processing Systems, 36, 2024.

[84] Y. Lu. Irl-imitation. https://github.com/yrlu/irl-imitation, 2019.

[85] B. Luders, M. Kothari, and J. How. Chance constrained rrt for probabilistic robustness to environmental uncertainty. In AIAA guidance, navigation, and control conference, page 8160, 2010.

[86] W. Luo, W. Sun, and A. Kapoor. Multi-robot collision avoidance under uncertainty with probabilistic safety barrier certificates. Advances in Neural Information Processing Systems, 33:372–383, 2020.

[87] Y. Lyu, W. Luo, and J. M. Dolan. Probabilistic safety-assured adaptive merging control for autonomous vehicles. In 2021 IEEE International conference on robotics and automation (ICRA), pages 10764–10770. IEEE, 2021.

[88] S. Malik, U. Anwar, A. Aghasi, and A. Ahmed. Inverse constrained reinforcement learning. In International Conference on Machine Learning, pages 7390–7399. PMLR, 2021.

[89] A. Mandhane, A. Zhernov, M. Rauh, C. Gu, M. Wang, F. Xue, W. Shang, D. Pang, R. Claus, C.-H. Chiang, et al. Muzero with self-competition for rate control in vp9 video compression. arXiv preprint arXiv:2202.06626, 2022.

[90] D. L. McPherson, K. C. Stocking, and S. S. Sastry. Maximum likelihood constraint inference from stochastic demonstrations. In 2021 IEEE Conference on Control Technology and Applications (CCTA), pages 1208–1213. IEEE, 2021.

[91] N. Mehr, R. Horowitz, and A. D. Dragan. Inferring and assisting with constraints in shared autonomy. In 2016 IEEE 55th Conference on Decision and Control (CDC), pages 6689–6696. IEEE, 2016.

[92] M. Menner, P. Worsnop, and M. N. Zeilinger. Constrained inverse optimal control with application to a human manipulation task. IEEE Transactions on Control Systems Technology, 29(2):826–834, 2021.

[93] P. E. Meyer, C. Schretter, and G. Bontempi. Information-theoretic feature selection in microarray data using variable complementarity. IEEE Journal of Selected Topics in Signal Processing, 2(3):261–274, 2008.

[94] J. Miller and J. P. How. Demand estimation and chance-constrained fleet management for ride hailing. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4481–4488. IEEE, 2017.

[95] T. Moers, L. Vater, R. Krajewski, J. Bock, A. Zlocki, and L. Eckstein. The exid dataset: A real-world trajectory dataset of highly interactive highway scenarios in germany. In 2022 IEEE Intelligent Vehicles Symposium (IV), pages 958–964, 2022.

[96] Narendra and Fukunaga. A branch and bound algorithm for feature subset selection. IEEE Transactions on computers, 100(9):917–922, 1977.

[97] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In Icml, volume 1, page 2, 2000.

[98] X. V. Nguyen, J. Chan, S. Romano, and J. Bailey. Effective global approaches for mutual information based feature selection. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 512–521, 2014.

[99] F. Oldewurtel, A. Parisio, C. N. Jones, M. Morari, D. Gyalistras, M. Gwerder, V. Stauch, B. Lehmann, and K. Wirth. Energy efficient building climate control using stochastic model predictive control and weather predictions. In Proceedings of the 2010 American control conference, pages 5100–5105. IEEE, 2010.

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

[101] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. Advances in neural information processing systems, 35:27730–27744, 2022.

[102] L. Pais, K. Umezawa, Y. Nakamura, and A. Billard. Learning robot skills through motion segmentation and constraints extraction. In HRI Workshop on Collaborative Manipulation, page 5. Citeseer, 2013.

[103] D. Papadimitriou, U. Anwar, and D. S. Brown. Bayesian inverse constrained reinforcement learning. In Workshop on Safe and Robust Control of Uncertain Systems (NeurIPS), 2021.

[104] D. Papadimitriou, U. Anwar, and D. S. Brown. Bayesian methods for constraint inference in reinforcement learning. Transactions on Machine Learning Research, 2022.

[105] D. Park, M. Noseworthy, R. Paul, S. Roy, and N. Roy. Inferring task goals and constraints using bayesian nonparametric inverse reinforcement learning. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, Proceedings of the Conference on Robot Learning, volume 100 of Proceedings of Machine Learning Research, pages 1005–1014. PMLR, 30 Oct–01 Nov 2020.

[106] B. Peng, J. Duan, J. Chen, S. E. Li, G. Xie, C. Zhang, Y. Guan, Y. Mu, and E. Sun. Model-based chance-constrained reinforcement learning via separated proportional-integral lagrangian. IEEE Transactions on Neural Networks and Learning Systems, 35(1):466–478, 2022.

[107] B. Peng, Y. Mu, Y. Guan, S. E. Li, Y. Yin, and J. Chen. Model-based actor-critic with chance constraint for stochastic system. In 2021 60th IEEE Conference on Decision and Control (CDC), pages 4694–4700. IEEE, 2021.

[108] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on pattern analysis and machine intelligence, 27(8):1226–1238, 2005.

[109] C. Pérez-D'Arpino and J. 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.

[110] T. Phan-Minh, F. Howington, T.-S. Chu, S. U. Lee, M. S. Tomov, N. Li, C. Dicle, S. Findler, F. Suarez-Ruiz, R. Beaudoin, et al. Driving in real life with inverse reinforcement learning. arXiv preprint arXiv:2206.03004, 2022.

[111] G. Qiao, G. Liu, P. Poupart, and Z. Xu. Multi-modal inverse constrained reinforcement learning from a mixture of demonstrations. Advances in Neural Information Processing Systems, 36, 2024.

[112] G. Quan, G. Liu, et al. Learning constraints from offline demonstrations via superior distribution correction estimation. In Forty-first International Conference on Machine Learning, 2024.

[113] A. Ray, J. Achiam, and D. Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning, 2019.

[114] C. A. Rothkopf and C. Dimitrakakis. Preference elicitation and inverse reinforcement learning. In Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part III 22, pages 34–48. Springer, 2011.

[115] S. Russell. Learning agents for uncertain environments. In Proceedings of the eleventh annual conference on Computational learning theory, pages 101–103, 1998.

[116] D. Sadigh, A. Dragan, S. Sastry, and S. Seshia. Active preference-based learning of reward functions. 2017.

[117] H. Satija, P. Amortila, and J. Pineau. Constrained markov decision processes via backward value functions. In International Conference on Machine Learning, pages 8502–8511. PMLR, 2020.

[118] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438, 2015.

[119] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[120] D. R. Scobee and S. S. Sastry. Maximum likelihood constraint inference for inverse reinforcement learning. In International Conference on Learning Representations, 2019.

[121] G. Shi, N. Karapetyan, A. B. Asghar, J.-P. Reddinger, J. Dotterweich, J. Humann, and P. Tokekar. Risk-aware uav-ugv rendezvous with chance-constrained markov decision process. In 2022 IEEE 61st Conference on Decision and Control (CDC), pages 180–187. IEEE, 2022.

[122] A. Shishkin, A. Bezzubtseva, A. Drutsa, I. Shishkov, E. Gladkikh, G. Gusev, and P. Serdyukov. Efficient high-order interaction-aware feature selection based on conditional mutual information. Advances in neural information processing systems, 29, 2016.

[123] A. Sjodin, D. Gayme, and U. Topcu. Risk-mitigated optimal power flow with high wind penetration. power, 14(8):15, 2012.

[124] J. M. V. Skalse, M. Farrugia-Roberts, S. Russell, A. Abate, and A. Gleave. Invariance in policy optimisation and partial identifiability in reward learning. In International Conference on Machine Learning, pages 32033–32058. PMLR, 2023.

[125] V. Sugumaran, V. Muralidharan, and K. Ramachandran. Feature selection using decision tree and classification through proximal support vector machine for fault diagnostics of roller bearing. Mechanical systems and signal processing, 21(2):930–942, 2007.

[126] T. Summers, J. Warrington, M. Morari, and J. Lygeros. Stochastic optimal power flow based on convex approximations of chance constraints. In 2014 Power Systems Computation Conference, pages 1–7. IEEE, 2014.

[127] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[128] C. Tessler, D. J. Mankowitz, and S. Mannor. Reward constrained policy optimization. arXiv preprint arXiv:1805.11074, 2018.

[129] R. Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society Series B: Statistical Methodology, 58(1):267–288, 1996.

[130] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ international conference on intelligent robots and systems, pages 5026–5033. IEEE, 2012.

[131] D. Vasquez, B. Okal, and K. O. Arras. Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison. In 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1341–1346. IEEE, 2014.

[132] M. Vidal-Naquet and S. Ullman. Object recognition with informative features and linear classification. In ICCV, volume 3, page 281, 2003.

[133] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17:261–272, 2020.

[134] S. Xu and G. Liu. Uncertainty-aware constraint inference in inverse constrained reinforcement learning. In The Twelfth International Conference on Learning Representations, 2023.

[135] S. Xu and G. Liu. Robust inverse constrained reinforcement learning under model misspecification. In Forty-first International Conference on Machine Learning, 2024.

[136] J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. IEEE Intelligent Systems and their Applications, 13(2):44–49, 1998.

[137] Y. Yang, J. O. Pedersen, et al. A comparative study on feature selection in text categorization. In Icml, volume 97, page 35. Citeseer, 1997.

[138] G. Ye and R. Alterovitz. Demonstration-Guided Motion Planning, pages 291–307. Springer International Publishing, Cham, 2017.

[139] M. Yo, M. Ono, B. C. Williams, and S. Adachi. Risk-limiting, market-based power dispatch and pricing. In 2013 European Control Conference (ECC), pages 3038–3045. IEEE, 2013.

[140] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In Proceedings of the 20th international conference on machine learning (ICML-03), pages 856–863, 2003.

[141] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. Maximum entropy inverse reinforcement learning. In Aaai, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[142] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. arXiv preprint arXiv:1909.08593, 2019.

APPENDICES

Appendix A

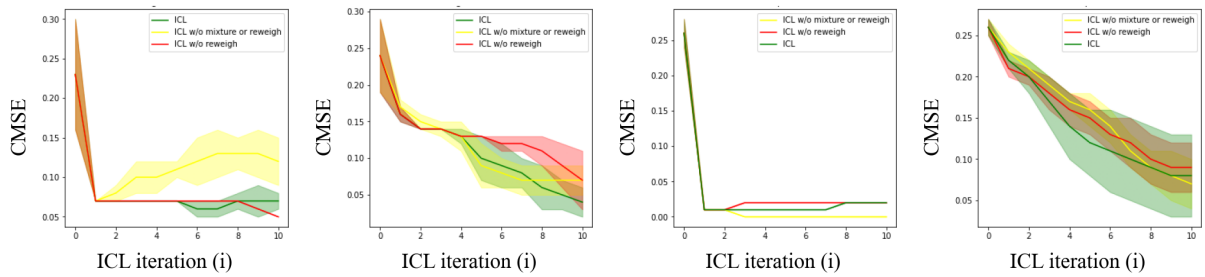
Additional experiments

A.1 Effect of policy mixture and reweighting

To assess the effect of having a policy mixture and reweighting, we conduct experiments with two more variants of ICL on all the synthetic environments:

- ICL wihout a mixture policy or reweighting (use the policy learned in constrained RL step instead of π_{mix} to do constraint adjustment step)
- ICL with a mixture policy but with no reweighting (use uniform weights)

Our results are reported in Figure A.1. We find that ICL without a mixture policy or reweighting is unable to perform as well as other methods with Gridworld (A) environment. Empirically, it is prone to divergence and cyclic behavior. This divergent behavior can be observed in the Gridworld (A) case. ICL with a mixture policy but with no reweighting is unable to converge as fast as ICL with mixture policy and reweighting in the Gridworld (B) and CartPole (Mid) environments. For the CartPole (MR) environment, all the methods converge fairly quickly and have almost the same performance. We also note that there is significant variance in the result for ICL on the CartPole (Mid) environment. Despite this, overall, ICL with a mixture policy and reweighting performs better than the other variants (or close to the best). Thus, we report this variant (with mixture policy and reweighting) in Table 4.2 and Table 4.3.



Gridworld (A) ($\beta = 0.99$) | Gridworld (B) ($\beta = 0.99$) | CartPole (MR) ($\beta = 50$) | CartPole (Mid) ($\beta = 30$)

Figure A.1: CMSE (Y-axis) vs ICL Iteration i (X-axis). In each iteration, we do one complete procedure of constrained RL and one complete procedure of constraint function adjustment.

A.2 Environment stochasticity

To assess the effect of environment stochasticity on the performance of our algorithm, we conduct experiments with a stochastic variant of Gridworld (A) environment. More specifically, we apply the ICL algorithm (Algorithm 1) to the Gridworld (A) environment, with varying values of the transition probability. A transition probability of 1 implies deterministic transition to the next state. A transition probability of $1 - p_{slip}$ implies a stochastic transition, such that the agent can enter the intended next state with this probability, and go to any other random direction with overall probability p_{slip}. Our results are reported in Tables A.1 and A.2, and in Figure A.2.

Empirically, we observe that the increase in environment stochasticity leads to a decrease in the performance of the constrained RL algorithm. Additionally, with a higher stochasticity, there is more noise in the demonstrations. When we increase the stochasticity slightly (going from a deterministic environment to $p_{slip} = 0.1$), we find that the recovered constraint function has lower error (see Figure A.2). This happens because with a little stochasticity, there is more exploration in terms of states visited, which leads to a more accurate constraint function since the algorithm tries to avoid more states. However, with more stochasticity ($p_{slip} = 0.5$), the demonstrations themselves become more noisy, and hence the algorithm recovers a worse constraint function than in the case with little stochasticity (see Table A.1). Finally, we also find that accruals of the learned policy also have more error w.r.t. provided demonstrations as the stochasticity increases (see Table A.2).

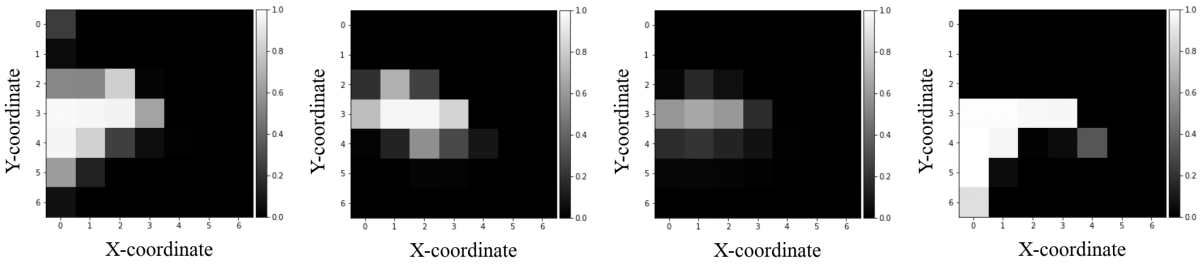
Table A.1: Constraint Mean Squared Error

Algorithm↓, Environment→	$p_{slip} = 0.1$	$p_{slip} = 0.3$	$p_{slip} = 0.5$
ICL	0.02 ± 0.00	0.03 ± 0.01	0.08 ± 0.00

Table A.2: Normalized Accrual Dissimilarity

Algorithm↓, Environment→	$p_{slip} = 0.1$	$p_{slip} = 0.3$	$p_{slip} = 0.5$
ICL	0.45 ± 0.10	1.13 ± 0.37	1.29 ± 0.20

Reported metrics (Mean ± Std. Deviation across 5 seeds) for the experiments on stochastic Gridworld (A) environments.



ICL ($p_{slip} = 0$) | ICL ($p_{slip} = 0.1$) | ICL ($p_{slip} = 0.3$) | ICL ($p_{slip} = 0.5$)

Figure A.2: Average constraint function value (averaged across 5 training seeds) for stochastic Gridworld (A) environment, demonstrating the effect of changing p_{slip}.

A.3 Comparison w.r.t a Lagrangian implementation

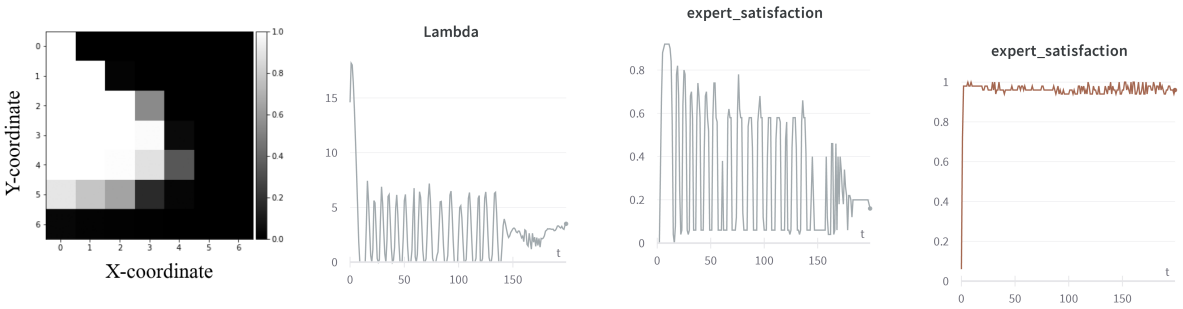
Equation (4.11) can also be represented using a min-max optimization:

$$\min_{\lambda \geq 0} \max_{c} J^{\pi_{mix}}(c) - \lambda(J^{\pi^E}(c) - \beta) \tag{A.1}$$

It is possible to perform this optimization instead of the penalty based strategy proposed in this work. In terms of implementation, this would amount to an alternating gradient ascent descent procedure, where we first ascent on c, followed by descent on λ, and we repeat these steps until convergence. However, as is the case with min-max optimizations, this is prone to training instability and oscillatory behavior. We demonstrate this with

the Gridworld (A) environment. In particular, we can see this oscillatory behavior in the plot for the Lagrange multiplier λ (Figure A.3, second subfigure). We also plot the expert satisfaction % for the Lagrangian and non Lagrangian implementation (ICL) (see third and fourth subfigure in Figure A.3). The expert satisfaction % is computed as the percentage of expert demonstrations satisfying the constraint using the current constraint function, during training. As the plots show, the Lagrangian implementation is prone to oscillatory behavior. Initially it does achieve a high satisfaction, but over time, the satisfaction degrades. This does not happen in the penalty based ICL implementation. This is our primary rationale for using a penalty based method for Equation (4.11).

In terms of the recovered constraint function, we find that the Lagrangian implementation is still able to achieve a noisy yet somewhat reasonable constraint function (see the first subfigure in Figure A.3). Therefore, the Lagrangian approach is indeed correct in principle and can be used to recover a constraint function. However, in terms of other training metrics, the Lagrangian approach may not perform well. Our proposed method is able to mitigate this.



Avg. constraint value (5 seeds) on Gridworld (A)

Lagrange multiplier λ for Lagrangian implementation

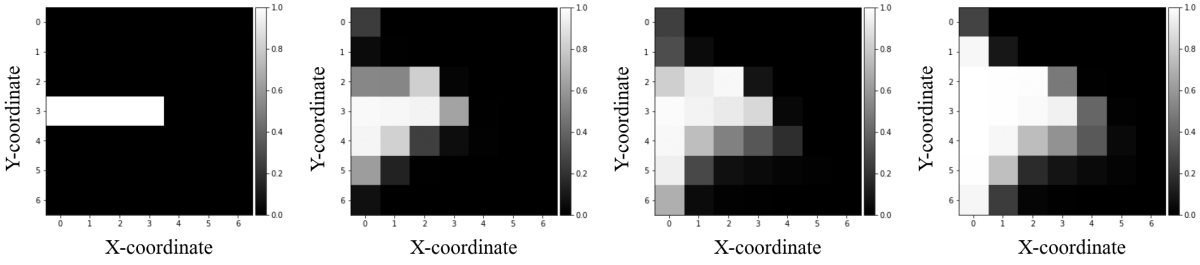
Expert satisfaction over time for Lagrangian implementation

Expert satisfaction over time for ICL

Figure A.3: The first figure shows the avg. constraint function (averaged across 5 seeds) for the Lagrangian implementation. The second figure shows the value of the Lagrange multiplier during training for 1 seed (X-axis denotes the adjustment iteration). The last two figures show the value of the expert satisfaction (%) for the Lagrangian implementation and the regular implementation (ICL).

A.4 Effect of β

We also assess the effect of β with the Gridworld (A, B) environments. We do not assess the effect of β on the CartPole (MR, Mid) environments since our claim can be easily verified by just observing the learned costs for the Gridworld (A, B) environments. Our claim is as follows. With a higher β, the learned constraint function should have a higher CMSE and more state-action pairs with higher constraint values. This is because since β is higher, the agent is allowed to visit more high constraint value state-action pairs, as the constraint threshold is more relaxed. To verify this claim, we run experiments on the Gridworld (A, B) environments. Specifically, in addition to $\beta = 0.99$ (which corresponds to the results in Tables 4.2 and 4.3), we also run experiments with $\beta = 2.99, 5.99$. Our results are reported in Figures A.4, A.5. The figures validate our claims.



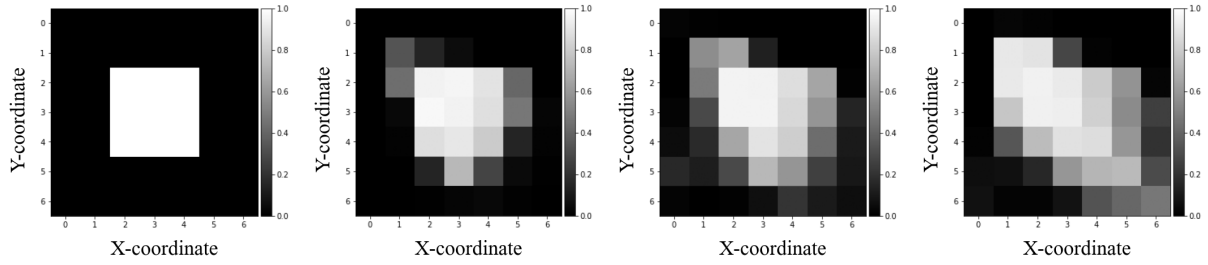
True constraint function | ICL ($\beta = 0.99$) | ICL ($\beta = 2.99$) | ICL ($\beta = 5.99$)

Figure A.4: Average constraint function value (averaged across 5 training seeds) for Gridworld (A) environment, demonstrating the effect of changing β.

A.5 Effect of δ

We now assess the effect of δ with the CartPole environment (used in the IPCL experiments). We generate the expert data to be used in these experiments with a new seed. The accrual for this expert can be found in Figure A.6. We then run IPCL with $\delta = 0.5, 0.7, 0.9$ and report the recovered costs in Figure A.7 and the normalized accruals in Figure A.8.

As δ increases, we expect that less and less violations are allowed. Specifically, we find that the IPCL algorithm sets several parts of the constraint function to a low value to reduce the cumulative costs across some trajectories. Since the accruals are still similar



True constraint function | ICL ($\beta = 0.99$) | ICL ($\beta = 2.99$) | ICL ($\beta = 5.99$)

Figure A.5: Average constraint function value (averaged across 5 training seeds) for Grid-world (B) environment, demonstrating the effect of changing β.

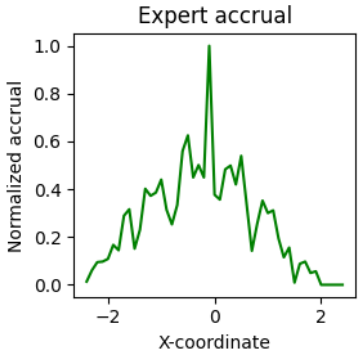


Figure A.6: Expert accrual for the CartPole environment. The experiment is to assess the effect of δ.

to the expert accruals, these recovered constraints are valid (and alternative constraints). Moreover, in the constraints, the left tail vanishes with increasing δ since the expert data is slightly left shifted.

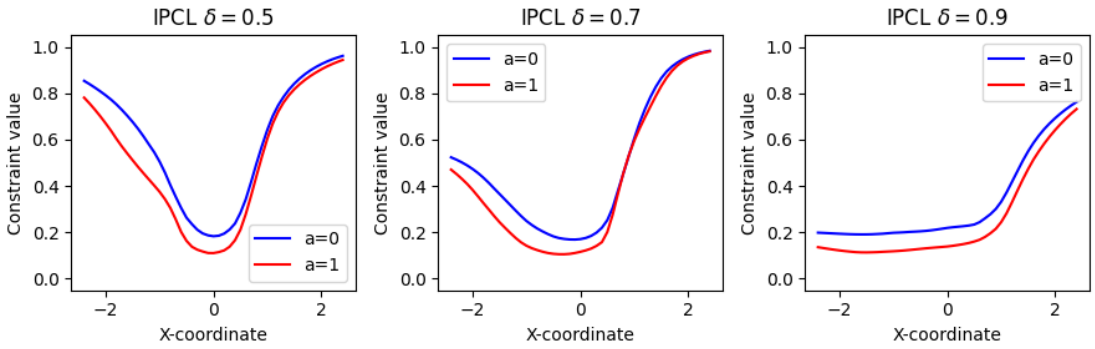


Figure A.7: Recovered costs for the CartPole environment for varying δ. The experiment is to assess the effect of δ.

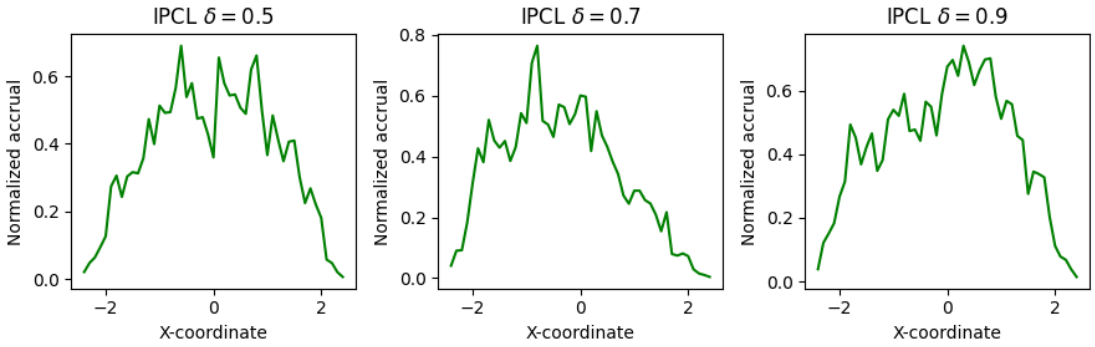


Figure A.8: Accrual for the CartPole environment for varying δ. The experiment is to assess the effect of δ.

Appendix B

Experiment details

B.1 Code

The code for this work can be found at github.com/ashishgaurav13/Thesis-code.

B.2 Constraint function inputs and horizons

The inputs to the constraint functions are as follows:

- **Gridworld (A, B), Gridworld, Gridworld-O**: x, y coordinates of the agent's position in the 7x7 grid
- **CartPole (MR, Mid), CartPole**: x position of the cart, and the discrete action (0/1 corresponding to left or right)
- **HighD**: velocity (ms^{-1}) of the ego, and the distance of the ego to the front vehicle in pixels (1 pixel $\approx 2.5\ m$)
- **Ant-Constrained, HalfCheetah-Constrained**: z-coordinate of the agent
- **ExiD**: signed distance (m) of the ego to the center line of the target lane, and the lateral velocity action (ms^{-1}) of the ego

The environment maximum horizons (episodes are terminated after these many steps) are as follows:

- **Gridworld (A, B), Gridworld, Gridworld-O**: 50 steps
- **CartPole (MR, Mid), CartPole**: 200 steps
- **HighD**: 1000 steps
- **Ant-Constrained**: 500 steps
- **HalfCheetah-Constrained**: 1000 steps for learning expected constraints, 500 steps for learning probabilistic constraints
- **ExiD**: 1000 steps

B.3 Metrics

We use three metrics for evaluate our results, described earlier in this work. The corresponding discretized state-action spaces are as follows:

- **Gridworld, Gridworld (A, B), Gridworld-O**: $\{(x, y) \mid x, y \in \{0, 1, \ldots, 6\}\}$
- **CartPole, Cartpole (MR, Mid)**: $\{(x, a) \mid x \in \{-2.4, -2.3, \cdots, 2.4\}, a \in \{0, 1\}\}$
- **Mujoco environments**: $\{z \mid z \in \{-5, -4.9, \cdots, 5\}\}$
- **HighD**: $\{(v, g) \mid v \in \{0, 1, \cdots, 40\}, g \in \{0, 2.5, \cdots, 200\}\}$ (v is in ms^{-1} and g is in m)
- **ExiD**: $\{(d, v) | d \in \{-10, -9.5, \cdots, 10\}, v \in \{-2.5, -2.4, \cdots, 2.5\}\}$ (here d is in m and v is in ms^{-1})

B.4 Expert dataset generation

B.4.1 ICL and IPCL

For synthetic experiments (Gridworld and CartPole) and robotics experiments (Mujoco), we simply perform CRL or PCRL for sufficient number of epochs until convergence. Afterwards, we generate trajectories using the learned policy.

For the HighD dataset, the data collection process is described in [63]. For this dataset, we choose one of the multiple scenarios in this dataset and use $\approx$ 100 trajectories from this scenario that go from the start region to the end region. For the ExiD dataset, the data collection process is similarly described in [95]. For this dataset, we randomly choose 5 scenarios, and filter tracks to get $\approx$ 1000 trajectories in each of which a vehicle performs a single lane change, with no vehicle in the target lane.

B.4.2 Feature selection ground truths

The ground truths for feature selection are produced manually through domain knowlege depending on the environment and the kind of constraint being learned.

B.5 Hyperparameters

B.5.1 Common Hyperparameters for learning expected constraints

Hyperparameters common to all the experiments for learning expected constraints are listed in Table B.1. Note that since HighD/ExiD/Ant-Constrained/HalfCheetah-Constrained are continuous action space environments, we use TANH activation for the policy, which outputs the mean and std. deviation of a learned Gaussian action distribution.

B.5.2 Hyperparameters for GAIL-Constraint and ICRL

Hyperparameters specific to the GAIL-Constraint method are listed in Table B.2. This method was adapted from Malik et al. [88]. Hyperparameters for ICRL [88] are listed in Table B.3.

B.5.3 Hyperparameters for learning expected constraints

The hyperparameters for ICL (proposed method to learn expected constraints) are listed in Table B.4. For some environments, we use a variant of ICL that uses PPO-Lagrange [113] as the forward constrained RL procedure. This implementation (provided by OpenAI) is highly optimized for constrained RL tasks, especially robotics tasks. We use this variant of ICL for our Mujoco and ExiD experiments (Table B.5).

Table B.1: Common Hyperparameters for ICL experiments

Hyperparameter	Value(s)
PPO learning rate (η_2)	5×10^{-4}
Constraint function learning rate (η_3)	5×10^{-4}
Constraint function hidden layers	64+ReLU, 64+ReLU
Constraint function final activation	Sigmoid
PPO policy layers	64+ReLU, 64+ReLU
PPO value fn. layers	64+ReLU, 64+ReLU
Minibatch size	64
PPO clip parameter (ϵ_{PPO})	0.1
PPO entropy coefficient (λ_{ent})	0.01
PPO updates per epoch	25
No. of trajectories for any dataset	50
Training seeds	1, 2, 3, 4, 5

Table B.2: GAIL-Constraint Hyperparameters

Hyperparameter	Environment				
	GA	GB	CMR	CMid	HighD
Discount factor (γ)	1.0	1.0	0.99	0.99	0.99
PPO steps per epoch	2000	2000	2000	2000	2000
PPO value fn. loss coefficient	0.5	0.5	0.5	0.5	0.5
PPO gradient clip value	0.5	0.5	0.5	0.5	0.5
PPO total steps	2M	2M	2M	2M	0.4M

Hyperparameter	Environment		
	Ant	HC	ExiD
Discount factor (γ)	0.99	0.99	0.99
PPO steps per epoch	4000	4000	1000
PPO value fn. loss coefficient	0.5	0.5	0.5
PPO gradient clip value	0.5	0.5	0.5
PPO total steps	3.5M	3.5M	3.5M

B.5.4 Hyperparameters for learning probabilistic constraints

The common hyperparameters for our experiments are listed in Table B.6, and the environment specific hyperparameters are listed in Table B.7. For all environments except Gridworld, we implemented a PPO-Penalty with GAE in order to learn a good policy in the probabilistic setting. Further, the reward and observations are also normalized for the Mujoco setting. For ICL, we use OpenAI's PPO Lagrange [113] for the forward PPO procedure for all Gym environments (Mujoco and Cartpole). For other environments, we use PPO Penalty [42].

B.5.5 Hyperparameters for UAICRL

The hyperparameters for UAICRL [134] are provided in Tables B.8 and B.9.

Table B.3: ICRL Hyperparameters

Hyperparameter	**Environment**				
	GA	GB	CMR	CMid	HighD
Discount factor (γ)	1.0	1.0	0.99	0.99	0.99
PPO steps per epoch	2000	2000	2000	2000	2000
PPO value fn. loss coefficient	0.5	0.5	0.5	0.5	0.5
Constraint value fn. loss coefficient	0.5	0.5	0.5	0.5	0.5
PPO gradient clip value	0.5	0.5	0.5	0.5	0.5
Eval episodes	100	100	100	100	100
ICRL Iterations	200	200	200	200	40
Forward PPO timesteps	10000	10000	10000	10000	10000
PPO Budget	0	0	0	0	0
Penalty initial value (ν)	0.1	0.1	0.1	0.1	0.1
Penalty learning rate	0.01	0.01	0.01	0.01	0.01
Per step importance sampling	Yes	Yes	Yes	Yes	Yes
Forward KL (Old/New)	10	10	10	10	10
Backward KL (New/Old)	2.5	2.5	2.5	2.5	2.5
Backward iterations	5	5	5	5	5
Constraint network reg. coefficient	0.6	0.6	0.6	0.6	0.6

Hyperparameter	**Environment**		
	Ant	HC	ExiD
Discount factor (γ)	0.99	0.99	0.99
PPO steps per epoch	4000	4000	1000
PPO value fn. loss coefficient	0.5	0.5	0.5
Constraint value fn. loss coefficient	0.5	0.5	0.5
PPO gradient clip value	0.5	0.5	0.5
Eval episodes	100	100	100
ICRL Iterations	125	125	250
Forward PPO timesteps	10000	10000	10000
PPO Budget	0	0	0
Penalty initial value (ν)	0.1	0.1	0.1
Penalty learning rate	0.01	0.01	0.01
Per step importance sampling	Yes	Yes	Yes
Forward KL (Old/New)	10	10	10
Backward KL (New/Old)	2.5	2.5	2.5
Backward iterations	5	5	5
Constraint network reg. coefficient	0.6	0.6	0.6

Table B.4: ICL Hyperparameters

Hyperparameter	Environment				
	GA	GB	CMR	CMid	HighD
Discount factor (γ)	1.0	1.0	0.99	0.99	0.99
ICL Iterations (n)	10	10	10	10	3
Correction learning rate (η_1)	2.5×10^{-5}	2.5×10^{-5}	2.5×10^{-5}	2.5×10^{-5}	2.5×10^{-5}
PPO episodes per epoch	20	20	20	20	20
β	0.99	0.99	50	30	0.1
Constraint update epochs (e)	20	20	20	20	25
PPO epochs (m)	500	500	300	300	50
Soft loss coefficient λ	15	15	15	15	15

Table B.5: ICL Hyperparameters (with PPO-Lagrange from [113])

Hyperparameter	Environment		
	Ant	HC	ExiD
Discount factor (γ)	0.99	0.99	0.99
GAE Lambda (λ_{GAE})	0.97	0.97	0.97
ICL Iterations (n)	5	5	5
PPO steps per epoch	4000	4000	1000
β	5	15	5
Constraint update epochs (e)	25	25	25
PPO epochs (m)	50	50	50

Table B.6: List of common hyperparameters for IPCL experiments

Hyperparameter	Value(s)
PPO learning rate (η_2)	5×10^{-4}
Constraint function learning rate (η_3)	5×10^{-4}
Constraint function hidden layers	64+ReLU, 64+ReLU
Constraint function final activation	Sigmoid
PPO policy layers	64+ReLU, 64+ReLU
PPO value fn. layers	64+ReLU, 64+ReLU
Minibatch size	64
Training seeds	0, 1, 2, 3, 4

Table B.7: List of specific hyperparameters (note: CA = constraint adjustment)

Hyperparameter	Gridworld	CartPole	Ant	HalfCheetah	HighD
Expert δ	0.6	0.7	0.7	0.5	-
Expert β	0.99	20	15	15	-
ICL/IPCL β	0.99	20	15	15	0.1
ICL/IPCL δ	0.6	0.7	0.7	0.5	0.5,0.9
Discount factor (γ)	1.0	0.99	0.99	0.99	1.0
ICL/IPCL iterations (n)	10/10	5/15	5/10	10/10	3/3
PPO Iterations	500	300	100	100	50
Horizon	50	200	500	500	1000
Max steps per epoch	-	4000	4000	4000	20000
Feasibility learning rate (η_1)	2.5×10^{-5}	2.5×10^{-5}	5×10^{-2}	5×10^{-2}	2.5×10^{-5}
Episodes per epoch	20	-	-	-	-
PPO train subepochs	25	25	80	80	25
PPO Clipping	0.1	0.1	0.2	0.2	0.1
PPO Entropy Coefficient	0.01	0.01	0	0	0.01
GAE used	No	Yes	Yes	Yes	Yes
GAE lambda (λ_{GAE})	-	0.97	0.97	0.97	0.97
ICL/IPCL CA coefficient λ	15/15	15/15	15/1	15/1	15/15
ICL/IPCL CA iterations	20/20	20/20	20/10	20/5	20/20

Table B.8: List of common hyperparameters (UAICRL)

Hyperparameter	**Value(s)**
PPO learning rate	3×10^{-5}
Constraint function hidden layers	256+ReLU, 256+ReLU
Constraint function final activation	Sigmoid
PPO policy layers	64+ReLU, 64+ReLU
Training seeds	0, 1, 2, 3, 4

Table B.9: List of specific hyperparameters (UAICRL)

Hyperparameter	Gridworld	CartPole	Ant	HalfCheetah
Discount factor (γ)	1.0	0.99	0.99	0.99
UAICRL iterations	20	20	30	30
Forward steps	8000	10000	50000	50000
Budget (β)	0.99	20	15	15
Cost quantiles	60	70	70	50
N quantiles	100	100	100	100
Backward iterations	5	5	5	5