

DISS. ETH NO. 29577

ALGORITHMIC FOUNDATIONS FOR SAFE AND
EFFICIENT REINFORCEMENT LEARNING
FROM HUMAN FEEDBACK

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

DAVID NIKOLAS LINDNER
M.Sc. ETH, ETH Zurich

born on 26 May 1995
citizen of Germany

accepted on the recommendation of
Prof. Dr. Andreas Krause, examiner
Dr. Katja Hofmann, co-examiner
Prof. Dr. Dorsa Sadigh, co-examiner

2023

David Nikolas Lindner: *Algorithmic Foundations for Safe and Efficient Reinforcement Learning from Human Feedback*, © 2023

DOI: [10.3929/ethz-b-000635156](https://doi.org/10.3929/ethz-b-000635156)

ABSTRACT

Reinforcement learning (RL) has shown remarkable success in applications with well-defined reward functions, such as maximizing the score in a video game or optimizing an algorithm’s run-time. However, in many real-world applications, there is no well-defined reward function. Instead, Reinforcement Learning from Human Feedback (RLHF) allows RL agents to learn from human-provided data, such as evaluations or rankings of trajectories. In many applications, human feedback is expensive to collect; therefore, learning robust policies from limited data is crucial. In this dissertation, we propose novel algorithms to enhance the sample efficiency and robustness of RLHF.

First, we propose active learning algorithms to improve the sample efficiency of RLHF by selecting the most informative data points for the user to label and by exploring the environment guided by uncertainty about the user’s preferences. Our approach provides conceptual clarity about active learning for RLHF and theoretical sample complexity results, drawing inspiration from multi-armed bandits and Bayesian optimization. Moreover, we provide extensive empirical evaluations in simulations that demonstrate the benefit of active learning for RLHF.

Second, we extend RLHF to learning constraints from human preferences instead of or in addition to rewards. We argue that constraints are a particularly natural representation of human preferences, particularly in safety-critical applications. We develop algorithms to learn constraints effectively from demonstrations with unknown rewards and actively learn constraints from human feedback. Our results suggest that representing human preferences as constraints can lead to safer policies and extend the potential applications for RLHF.

The proposed algorithms for reward and constraint learning serve as a foundation for future research to enhance the efficiency, safety, and applicability of RLHF.

ZUSAMMENFASSUNG

Reinforcement Learning (RL; dt. bestärkendes Lernen) hat bemerkenswerte Erfolge in Anwendungen mit klar definierten Belohnungsfunktionen erzielt. Beispiele hierfür sind, die Punktzahl in einem Videospiel zu maximieren oder die Laufzeit eines Algorithmus zu optimieren. In vielen praktischen Anwendungen gibt es jedoch keine klar definierte Belohnungsfunktion. Reinforcement Learning from Human Feedback (RLHF; dt. etwa "bestärkendes Lernen mit menschlichen Rückmeldungen") ermöglicht es RL-Agenten aus Daten zu lernen, die von Menschen zur Verfügung gestellt werden, wie z.B. Bewertungen von Trajektorien. Diese Art der Daten sind in vielen Anwendungen mit hohen Kosten verbunden. Daher ist es wichtig, aus wenigen Daten eine robuste Strategie zu lernen. In dieser Dissertation werden neue Algorithmen entwickelt, um die Effizienz und Robustheit von RLHF zu verbessern.

Im ersten Teil werden aktive Lernalgorithmen entwickelt, um die Effizienz von RLHF zu verbessern. Diese Methoden wählen die aussagekräftigsten Datenpunkte zur Auswertung aus und erkunden die Umgebung, abhängig von der Unsicherheit über Nutzerpräferenzen. Dieser Ansatz liefert neue Erkenntnisse über aktives Lernen für RLHF und theoretische Ergebnisse zur Effizienz von RLHF inspiriert von mehrarmigen Banditen und der Bayes'schen Optimierung. Der Nutzen des aktiven Lernens für RLHF wird durch umfangreiche empirische Evaluationen der entwickelten Algorithmen belegt.

Im zweiten Teil wird RLHF erweitert, sodass anstelle oder zusätzlich zu einer Belohnungsfunktion auch Nebenbedingungen gelernt werden können. Nebenbedingungen sind eine besonders natürliche Repräsentation menschlicher Präferenzen, insbesondere in sicherheitskritischen RL-Anwendungen. Daher werden sowohl Algorithmen zum effektiven Lernen von Nebenbedingungen aus Demonstrationen mit unbekanntem Belohnungsfunktionen als auch zum akti-

ven Lernen von Nebenbedingungen aus menschlichen Bewertungen entwickelt. Die Ergebnisse dieses Teils zeigen, dass durch die Modellierung menschlicher Präferenzen als Nebenbedingungen sicherere Strategien erlernt werden können und der Anwendungsbereich von RLHF erweitert werden kann.

Die entwickelten Algorithmen für das effiziente Lernen von Belohnungsfunktionen und Nebenbedingungen bilden eine Grundlage für zukünftige Forschung, um die Effizienz, die Sicherheit und die Anwendbarkeit von RLHF zu verbessern.

ACKNOWLEDGEMENTS

None of the work in this dissertation would have been possible without the support of a significant number of people.

First and foremost, thanks to my doctoral advisors, Andreas Krause and Katja Hofmann. Their complementary expertise was invaluable for the research leading up to this dissertation. More importantly, Andreas and Katja were (and are) inspiring scientific role models, and I feel incredibly fortunate to have worked with them.

I had the privilege of having a de facto *third* advisor, Sebastian Tschitschek. In addition to being an exceptional collaborator, Sebastian provided valuable practical advice on the day-to-day of doing research and being a doctoral student.

Thanks to multiple others that acted as mentors and role models over the last couple of years. Particularly thanks to Matteo Turchetta, Rohin Shah, Ethan Perez, and Vladimir Mikulik for showing me how to do good research.

Further, thanks to the many brilliant collaborators I was fortunate to work with, including Giorgia Ramponi, Kamil Ciosek, Hoda Heidari, Mennatallah El-Assady, Thomas McGrath, Jérémy Scheurer, and Cynthia Chen, among others.

Thanks to my long-time office mate, Andisheh Amrollahi, and office neighbor, Max Paulus, for making coming to work fun. Thanks to the entire LAS group for being a welcoming, supportive, and fun place to work. Being around so many brilliant colleagues was stimulating and inspiring.

Thanks to Rita Klute and the entire IML administrative team – my work would have been impossible without them.

Finally, my deepest gratitude goes to my parents, who gave me so many opportunities and let me choose my path, and to my partner, Tuna, who supported me on this path through the highs and lows.

CONTENTS

1	Introduction	1
1.1	Overview of the Dissertation	4
1.2	Prior Publications Relevant to the Dissertation	6
2	Background	7
2.1	Multi-Armed Bandits	8
2.2	Bayesian Optimization and Active Learning	10
2.2.1	Gaussian Processes	11
2.2.2	Acquisition Functions	12
2.3	Reinforcement Learning	13
2.3.1	Markov Decision Processes	14
2.3.2	Constrained Markov Decision Processes	17
2.3.3	Linear Programming for MDPs and CMDPs	18
2.3.4	Dynamic Programming	20
2.3.5	Q-Learning	21
2.3.6	Policy Optimization	23
2.4	Reward Learning	24
2.4.1	Inverse Reinforcement Learning	24
2.4.2	Preference Learning	27
2.5	Overview of Notation	29
1	Active Learning for Reward Learning	
3	Information Directed Reward Learning	35
3.1	Related Work	36
3.2	Problem Setting	38
3.3	Information Directed Reward Learning	38
3.4	IDRL for GP Reward Models	41
3.5	Connection to Multi-Armed Bandits	44
3.5.1	Linear Partial Monitoring	45
3.5.2	Transductive Linear Bandits	45
3.6	IDRL for Deep RL	48

3.7	Experiments	50
3.7.1	Baselines	52
3.7.2	Can IDRL Improve Sample Efficiency?	53
3.7.3	Can IDRL Learn From Comparisons?	54
3.7.4	Does IDRL Scale to Bigger Environments?	58
3.7.5	Does IDRL Scale to Deep RL?	60
3.8	Conclusion	61
4	Active Exploration for Inverse Reinforcement Learning	63
4.1	Related Work	64
4.2	Active Learning for IRL	67
4.2.1	Problem Definition	67
4.2.2	Feasible Rewards in Finite-Horizon MDPs	68
4.2.3	Error Propagation	69
4.3	Uniform Sampling With a Generative Model	70
4.3.1	Estimating Transition Model and Expert Policy	71
4.3.2	Uniform Sampling Strategy	71
4.4	Active Exploration for IRL	72
4.4.1	Uncertainty-Based Exploration for IRL	73
4.4.2	Adaptive Exploration	74
4.4.3	Sample Complexity of AceIRL	76
4.5	Experiments	77
4.6	Connection to Reward-Free Exploration	79
4.7	Conclusion	81
II	Constraint Learning for Reinforcement Learning	
5	Learning Constraints From Demonstrations Without Rewards	87
5.1	Related Work	89
5.2	Inferring Constraints With Unknown Rewards	90
5.2.1	Problem Setup	90
5.2.2	Limitation of IRL in CMDPs	91
5.3	Convex Constraint Learning for RL	93
5.3.1	Constructing a Convex Safe Set	93
5.3.2	Convergence Under (Approximate) Optimality	95
5.3.3	Estimating Feature Expectations	97
5.3.4	Practical Implementation of CoCoRL	99

5.4	Experiments	100
5.4.1	IRL-Based Constraint Learning Baselines	100
5.4.2	Single-State CMDPs	101
5.4.3	Tabular Environments	102
5.4.4	Driving Environment	106
5.5	Conclusion	108
6	Interactively Learning Constraints	111
6.1	Related Work	114
6.2	Linear Constrained Best-Arm Identification	115
6.2.1	Lower Bounds	116
6.2.2	Confidence Intervals for Linear Regression	118
6.2.3	Algorithms With Static Confidence Intervals	119
6.2.4	Algorithms With Adaptive Confidence Intervals	123
6.3	Experiments	125
6.3.1	Synthetic Experiments	125
6.3.2	Comparing ACOL to Regret Minimization	128
6.3.3	Preference Learning Experiments	130
6.4	Conclusion	131
7	Conclusion	135

Appendix

A	Information Directed Reward Learning	139
A.1	Proofs	139
A.2	Implementation Details	144
A.2.1	Baselines	144
A.2.2	IDRL for Deep RL	147
A.3	Experiment Details	150
A.3.1	Environments	150
A.3.2	Additional Results	154
B	Active Exploration for IRL	157
B.1	Proofs	157
B.1.1	Simulation Lemmas	157
B.1.2	Feasible Reward Set	163
B.1.3	Uniform Sampling IRL With Generative Model	165
B.1.4	AceIRL: Problem Independent Analysis	170

B.1.5	AceIRL: Problem Dependent Analysis	178
B.1.6	Computing the Exploration Policy	185
B.2	Experiment Details	187
B.2.1	Environments	187
B.2.2	Additional Results	189
C	Convex Constraint Learning for RL	191
C.1	Proofs	191
C.1.1	Limitations of IRL in CMDPs	191
C.1.2	Safety Guarantees	192
C.1.3	Optimality Guarantees	194
C.1.4	Estimating Feature Expectations	197
C.2	Implementation Details	203
C.2.1	Constructing the Convex Hull	204
C.2.2	Iteratively Adding Points	206
C.3	Experiment Details	207
C.3.1	Gridworld Experiments	207
C.3.2	Driving Experiments	208
D	Adaptive Constraint Learning	213
D.1	Proofs	213
D.1.1	Lower Bounds	213
D.1.2	Adaptive Constraint Learning	219
D.1.3	Oracle and G-Allocation	222
D.2	Experiment Details	225
D.2.1	Driving Environment	225
D.2.2	Additional Results	226
	Bibliography	231

INTRODUCTION

Reinforcement Learning (RL; [1]) aims to build AI systems that learn from experience. An RL agent interacts with an environment and improves its behavior through trial and error, guided by a reward signal. RL agents have been successful at many impressive tasks, including playing complex board games (e.g., Chess [2], Go [3], and Stratego [4]), video games (e.g., Dota [5] and Starcraft [6]), and optimizing data center cooling [7], video compression [8], and sorting algorithms [9]. All of these applications have a well-defined and measurable reward signal, e.g., whether a game is won or how much time or memory an algorithm uses.

However, specifying a reward function can be challenging in many practical tasks [10]. For example, consider designing a reward function for autonomous driving. Humans consider many different objectives when driving, including safety, comfort, and efficiency. A reward function for autonomous driving has to consider all of these factors while also providing a dense enough signal for the agent to learn from. Knox *et al.* [11] find that many reward functions proposed in the autonomous driving literature fail basic consistency checks and, for example, miss important attributes, have loopholes the agent can exploit, or have reward-shaping terms that can lead to unsafe behavior.

Reinforcement Learning from Human Feedback (RLHF; [12]) addresses the difficulty of designing reward functions. In RLHF, rather than relying solely on a predefined reward function, the agent uses human feedback, such as evaluations or rankings of the agent's behavior in previous situations (Figure 1.1). RLHF provides a more intuitive and flexible way to teach complex behaviors to RL agents and promises to make RL applicable to a broader range of tasks.

Recently, RLHF has shown particular success in natural language processing, where reward functions are difficult to specify. Building

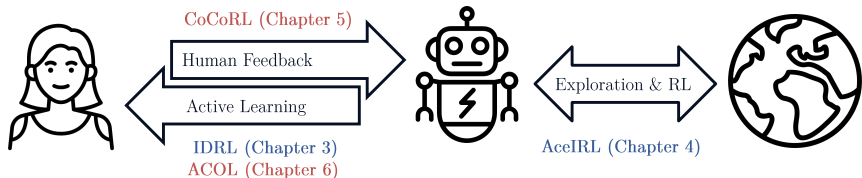


FIGURE 1.1: Schematic overview of Reinforcement Learning from Human Feedback (RLHF). In this dissertation, we study methods that represent human preferences as **reward models** as well as **constraint models**. In the first part, we study **reward learning** (shown in blue). We discuss active learning for choosing queries in Chapter 3 and active exploration in the environment in Chapter 4. In the second part, we focus on **constraint learning** (shown in red). We discuss learning constraints from demonstrations in Chapter 5 and introduce active learning for learning constraints in Chapter 6.

on large language models, agents trained using RLHF can summarize text [13], follow instructions [14], or act as full dialogue agents [15].

Despite its potential, RLHF faces challenges on multiple fronts (cf. the overview by Casper *et al.* [16]). These challenges encompass both modeling and algorithm design considerations (see, e.g., [17]), as well as human-centered factors (see, e.g., [18]). In this dissertation, we focus on the algorithmic challenges associated with the sample-efficiency and robustness of RLHF.

High-quality human feedback is expensive to collect, and current methods need a lot of feedback to learn robustly. The human feedback is invaluable for the learning process [13], but the time, effort, and expertise required to provide this feedback can often be prohibitive [19]. Therefore, the *first challenge* we consider is how to *maximize the value derived from limited human feedback*.

Current RLHF methods optimize for a single (learned) reward function, whereas humans often have multiple objectives and preferences [20]. Further, some preferences might act as constraints on the agent’s behavior rather than additional terms in a reward function.

For example, we could design a reward function for autonomous driving that simultaneously measures safety, comfort, and efficiency. However, we usually do not want to optimize all of these objectives but rather optimize one objective while ensuring that others are satisfied. For example, we might want to reach our destination as quickly as possible while driving safely. The observation that constraints can be a natural representation of human preferences motivates the *second challenge* we consider: how to *learn constraints from human feedback*.

We now arrived at the two primary research questions that we investigate in this dissertation:

- How can we make RLHF more sample efficient?
- How can we learn constraints from human feedback?

Our primary approach to address the first research question is active learning [21], i.e., selecting the most informative data points for humans to label. Prior work typically adapts standard active learning methods to RLHF (e.g., [22]). However, the situation in RL is different from supervised learning in two ways. First, in RL, we do not want to approximate the “true” reward function well but instead find a good policy, making the situation more similar to Bayesian optimization rather than active learning. Second, in RL, we must explore the environment to collect data for humans to label, which is unnecessary in supervised learning. These differences motivate us to define alternative objectives for active learning for RLHF, inspired by work on multi-armed bandits and Bayesian optimization.

Constraints are pivotal in many safety-critical RL applications, such as robotics. We argue that in such domains, we should learn constraint models from human feedback rather than only learning reward models. To enable this, we develop algorithms to learn constraints effectively, addressing our second research question. First, we focus on learning constraints from demonstrations with unknown rewards. Second, we combine this approach with active learning to develop an algorithm to actively learn constraints from human feedback.

By improving the algorithms we use to learn rewards and constraints from human feedback, we can expand the space of possible applications for RLHF. Moreover, learning better reward and constraint models could lead to more robust, reliable, and safe AI systems capable of handling complex tasks. The algorithms proposed in this dissertation can serve as a foundation for future research and development on making RLHF more efficient and safe.

1.1 OVERVIEW OF THE DISSERTATION

This dissertation is split into two parts. The first part focuses on learning reward models. We present a general approach to active reward learning that focuses on learning a good policy rather than only reducing approximation error (Chapter 3) and a method to actively explore the environment in order to collect data to query an expert about (Chapter 4). The second part focuses on learning constraint models. We argue that constraints might be a particularly useful representation for learning human preferences and propose methods for learning constraints from a set of demonstrations with unknown rewards (Chapter 5) as well as actively learning constraints from feedback about the safety of trajectories (Chapter 6).

Figure 1.1 provides an overview of how our contributions relate to the different parts of a typical RLHF setup. In the following, we summarize the contributions of the individual chapters.

- In Chapter 3, we propose *Information Directed Reward Learning* (IDRL), a general active reward learning approach for learning a model of the reward function from expensive feedback with the goal of *finding a good policy* rather than uniformly reducing the model’s error. IDRL can use arbitrary Bayesian reward models and arbitrary types of queries, making it more general than existing methods. We evaluate IDRL extensively in simulated environments, including a driving task and high-dimensional continuous control tasks in the MuJoCo simulator, and show it significantly outperforms prior methods.

- In Chapter 4, we focus on active exploration for Inverse Reinforcement Learning (IRL), i.e., inferring a reward from expert demonstrations. We propose the active IRL problem and characterize the necessary and sufficient conditions for solving it. Then, we propose *Active Exploration for Inverse Reinforcement Learning* (AceIRL), a novel algorithm to actively explore the environment and query the expert policy to infer a good reward function. We provide sample complexity guarantees for AceIRL and empirically evaluate it in simulated environments, demonstrating significantly better performance than more naive exploration strategies.
- In Chapter 5, we study the problem of inferring constraints in CMDPs from a set of demonstrations with unknown rewards. We propose *Convex Constraint Learning for Reinforcement Learning* (CoCoRL), a novel method for addressing this problem, and we prove that CoCoRL guarantees safety and asymptotic optimality. Further, our empirical evaluations of CoCoRL in tabular environments and a continuous driving task with multiple constraints show that CoCoRL learns constraints that lead to safe driving behavior and that can be transferred across different tasks and environments.
- In Chapter 6, we formalize the active learning problem for inferring constraints as a novel linear bandit problem (Section 6.2) which we call *constrained linear best-arm identification*. We provide an instance-dependent sample complexity lower bound and propose *Adaptive Constraint Learning* (ACOL), an algorithm that almost matches this lower bound. Our empirical evaluation shows that ACOL gets close to the performance of an oracle solution that has access to the true constraint function while outperforming a range of simpler baselines.

1.2 PRIOR PUBLICATIONS RELEVANT TO THE DISSERTATION

Parts of the work presented in this dissertation were published previously or made available as preprints. This dissertation is based on the following publications:

1. D. Lindner, M. Turchetta, S. Tschitschek, K. Ciosek, and A. Krause, “Information directed reward learning for reinforcement learning”, in *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
2. D. Lindner, A. Krause, and G. Ramponi, “Active exploration for inverse reinforcement learning”, in *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.
3. D. Lindner, S. Tschitschek, K. Hofmann, and A. Krause, “Interactively learning preference constraints in linear bandits”, in *International Conference on Machine Learning (ICML)*, 2022.
4. D. Lindner, X. Chen, S. Tschitschek, K. Hofmann, and A. Krause, “Learning safety constraints from demonstrations with unknown rewards”, *arXiv preprint arXiv:2305.16147*, 2023.

Chapters 1, 2 and 7 are based on all four publications. At the beginning of each of Chapters 3 to 6, we highlight the publication(s) relevant to the respective chapter.

BACKGROUND

This chapter lays the groundwork for our main contributions.

We start with an introduction to multi-armed bandits (MABs) in Section 2.1. MABs have become a key tool for studying learning under uncertainty. Chapter 6 is directly built on the MAB framework, and all other chapters use ideas from MABs indirectly.

Active learning and Bayesian optimization, discussed in Section 2.2, extend the ideas of learning and optimizing under uncertainty to more complex contexts. Understanding these principles will be essential as we move into studying RLHF. We use ideas from active learning and Bayesian optimization in Chapters 3, 4 and 6.

Section 2.3 introduces Reinforcement Learning (RL) and constrained RL, the main problems we study. RL focuses on sequential decision-making in an unknown environment. In contrast to multi-armed bandits, RL agents can take actions that influence the environment. We use the standard RL setup in Chapters 3 and 4 and constrained RL in Chapter 5 and to some extent in Chapter 6.

Next, in Section 2.4, we introduce Inverse Reinforcement Learning (IRL) and preference learning, two common approaches to RLHF. In IRL, the agent learns from expert demonstrations, whereas in preference learning, the agent learns from a human's expressed preferences. IRL is the focus of Chapters 4 and 5, and preference learning is the focus on Chapters 3 and 6.

Finally, Section 2.5 provides an overview of the notation used throughout this dissertation. We sometimes deviate from standard notation when combining ideas from different fields. Hence, Section 2.5 aims to be a helpful reference throughout reading this dissertation.

In summary, this chapter provides the context necessary to follow the later chapters. Each later chapter also includes a brief discussion

of related work to provide specific context for the contributions of that chapter.

2.1 MULTI-ARMED BANDITS

Multi-Armed Bandits (MABs) are a class of decision-making problems that have attracted significant attention due to their simplicity and applicability across various fields. We can view MAB problems as a restriction of RL. In MABs, the agent cannot influence the environment, which is a helpful restriction to make the problem significantly easier to analyze. The term “Multi-Armed Bandit” originates from a metaphorical scenario where a gambler is faced with several slot machines (also known as “one-armed bandits”) and has to decide which machine to play to maximize returns.

A MAB problem is a sequential decision-making task under uncertainty. The agent can choose between a set of “arms” (or actions) indexed by $i = 1, 2, \dots, K$. Each arm i has an associated unknown reward distribution with expected reward μ_i . At each time step $t = 1, 2, \dots, T$, the agent selects an arm a_t and receives a reward r_t sampled from the corresponding reward distribution. The agent’s goal is to select arms over time to maximize the accumulated reward, or equivalently, to minimize the *regret*.

The regret \mathcal{R}_T is the difference between the expected reward of the optimal arm (the arm with the highest expected reward) and the expected reward of the arms chosen by the agent, summed over T time steps:

$$\mathcal{R}_T = \sum_{t=1}^T (\mu^* - \mu_{a_t}), \quad (2.1)$$

where $\mu^* = \max_i \mu_i$ is the expected reward of the optimal arm.

The fundamental challenge of the MAB problem is to balance *exploration*, i.e., trying out different actions to learn more about their reward distributions, and *exploitation*, choosing the action that provides the highest reward based on the information gathered so far.

In contrast to regret minimization, *pure exploration* provides an alternative objective for MAB problems [23]. In pure exploration, the agent is not concerned with the accumulated reward. Instead, the agent only aims to identify the arm with the highest expected reward. This setting is particularly relevant when the cost of suboptimal actions is low during learning but high during deployment, e.g., when learning in a simulation.

Several popular algorithms exist for regret minimization and pure exploration in MAB problems. For instance, the *Upper Confidence Bound* (UCB) algorithm constructs confidence bounds around the estimated expected reward of each arm. It always selects the arm with the highest upper bound, a principle called “optimism in the face of uncertainty. In each iteration, the basic version of UCB selects the arm that maximizes:

$$a_t = \arg \max_i \left(\hat{\mu}_i + \sqrt{\frac{2 \log(1/\delta)}{N_i(t)}} \right), \quad (2.2)$$

where $\hat{\mu}_i$ is the empirical mean reward of arm i and $N_i(t)$ is the number of times arm i has been selected up to time t . This approach balances exploration and exploitation because it selects arms with high uncertainty as long as they could be better than those with low uncertainty. Assuming fixed sub-Gaussian noise on the rewards, UCB achieves a regret on the order of $\mathcal{O}(\sqrt{nk \log(k)})$, where n is the number of iterations and k the number of arms. This regret bound is optimal up to logarithmic factors¹[25].

Many approaches to pure exploration problems use *uncertainty sampling*, i.e., they select the arm with the highest uncertainty about the estimated reward. Instead of balancing exploration and exploitation, these algorithms only focus on exploration, i.e., they prioritize learning about the arms over immediate reward.

¹ Other variants of the UCB algorithm using different confidence bounds obtain different regret bounds. See Chapter 7 of Lattimore and Szepesvári [24] for an analysis of the variant we present here.

In many practical situations, we have additional information about the individual arms, even before we choose them the first time. In particular, we can often expect “similar” arms to have similar reward distributions. For example, in recommender systems, we have data about the customers, and we want to generalize from one customer to another one. The standard MAB framework cannot capture this. *Contextual* bandits allow incorporating such side information by augmenting each arm with a *context*.

One example of such a model is the *linear* bandit, where the reward is a linear function of some known feature vector (the context) associated with each arm and an unknown parameter vector. If arm i has feature vector $x_i \in \mathbb{R}^d$ and the unknown parameter vector is $\theta \in \mathbb{R}^d$, the expected reward of arm i is $\mu_i = x_i^T \theta$. The *Linear Upper Confidence Bound* (LinUCB) algorithm [26] is a popular algorithm for linear bandits. LinUCB maintains a confidence interval for the unknown parameter vector θ and selects the arm with the highest upper confidence bound, extending the idea of UCB to linear bandits.

Thompson sampling is another popular algorithm for MAB problems, including contextual bandits. Thompson sampling maintains a posterior distribution over the reward distributions of each arm. At each time step, it samples a reward from each posterior distribution and selects the arm with the highest sampled reward. Thompson sampling is known to be asymptotically optimal for regret minimization in MAB problems [27].

The MAB problem captures the essence of many practical sequential decision-making problems, including clinical trials, recommendation systems, ad placement, and many others [28]. For an accessible introduction to the theory of multi-armed bandits, we refer to Lattimore and Szepesvári [24].

2.2 BAYESIAN OPTIMIZATION AND ACTIVE LEARNING

Bayesian optimization (BO; [29]) significantly extends MAB problems to continuous and high-dimensional situations. BO aims to maximize

a black-box function f that is expensive to evaluate using as few evaluations as possible. To this end, BO algorithms maintain a probabilistic model of f , often in the form of a Gaussian Process (GP), and actively select data to update this model via an acquisition function. We can view BO as an extension of pure exploration in MABs to continuous “arms”. In BO, we can evaluate $f(x)$ at any point $x \in \mathbb{R}^d$, whereas in MABs, we can only evaluate a discrete set of arms.

Like BO, Active learning [21] aims to efficiently allocate resources to learn about an unknown function. However, BO aims to find the maximum of the function, whereas active learning aims to understand the function as a whole or identify particular characteristics. Active learning typically involves querying the function where the model’s uncertainty is high. Thus, active learning and BO can be viewed as complementary approaches to learning about an unknown function, using similar methods but with different ultimate objectives.

2.2.1 Gaussian Processes

Gaussian Processes (GPs; [30]) offer a powerful tool for modeling unknown functions in active learning and Bayesian optimization. A GP defines a distribution over functions, and is fully specified by a mean function $m(x)$ and a covariance function $k(x, x')$, also called the *kernel* function. Given a dataset of points $X = \{x_1, x_2, \dots, x_n\}$, a GP induces a multivariate normal distribution on the function values

$$f(X) = (f(x_1), f(x_2), \dots, f(x_n))^T \sim \mathcal{N}(\mu, \Sigma)$$

with mean $\mu = m(X)$, i.e., $\mu_i = m(x_i)$, and covariance matrix $\Sigma = k(X, X)$, i.e., $\Sigma_{ij} = k(x_i, x_j)$. Importantly, GP models are convenient to deal with computationally. Suppose we model a function using a GP model, denoted by $f \sim \mathcal{GP}(m, k)$, and want to update the model using observed data. Then, the GP posterior conditioned on the data is still a GP, albeit with a new mean and covariance function.

The mean function $m(x)$ represents the expected value of the function at x and is often assumed to be zero *w.l.o.g.* The kernel

function $k(x, x')$ measures the similarity between points: larger values indicate that x and x' are more similar. Common choices for the kernel function include the squared exponential kernel, periodic kernels, and the Matérn kernel, among others. The choice of kernel function can significantly influence the performance of the GP and often requires prior knowledge about the problem at hand.

A key advantage of GPs is that they offer a measure of uncertainty estimate about each prediction. We use this uncertainty in active learning and BO to decide where to sample next.

2.2.2 Acquisition Functions

In Bayesian optimization, where to sample next is guided by an *acquisition function* $u(x, \mathcal{D})$, which uses the GP's posterior mean and covariance. Typically a BO algorithm maximizes the acquisition function over the input space to choose the next point to evaluate.

One well-known acquisition function is the *Expected Improvement* (EI; [31]) over the best previous observation under the GP posterior:

$$u_{\text{EI}}(x, \mathcal{D}) = \mathbb{E}_{f \sim \text{GP}}[\max(f(x) - f(x_{\max}), 0) | \mathcal{D}],$$

where x_{\max} is the best observation so far. The expectation is computed under the GP posterior. EI implements a natural trade-off between exploration and exploitation, favoring points where the GP mean is high (exploitation) or the GP uncertainty is high (exploration).

Another popular acquisition function is the Upper Confidence Bound (GP-UCB; [32]), which extends the UCB algorithm from MABs to BO. Using a GP model, the GP-UCB acquisition function is

$$u_{\text{UCB}}(x, \mathcal{D}) = \mu(x, \mathcal{D}) + \beta \cdot \sigma(x, \mathcal{D}),$$

where $\mu(x, \mathcal{D})$ and $\sigma(x, \mathcal{D})$ are the GP posterior mean and standard deviation at x , respectively, and β is a parameter that controls the trade-off between exploration and exploitation.

Acquisition functions are central to the performance of BO, and much prior work focuses on developing new acquisition functions and improving existing ones.

In active learning, acquisition functions based on *information gain* are common. Intuitively, the information gain between two random variables measures the amount of information we can obtain about one by observing the other. Formally, for two random variables X and Y with marginal distributions p_X, p_Y and joint distribution $p_{(X,Y)}$, the information gain (or mutual information) is

$$I(X; Y) = D_{\text{KL}}(p_{(X,Y)} \| p_X \cdot p_Y),$$

where $D_{\text{KL}}(\cdot \| \cdot)$ is the KL-divergence. Given a third random variable Z , the conditional information gain is defined as

$$I(X; Y | Z = z) = D_{\text{KL}}(p_{(X,Y)|Z=z} \| p_{X|Z=z} \cdot p_{Y|Z=z}).$$

To perform active learning about a variable of interest Y , we can choose the conditional information gain as an acquisition function:

$$u_{\text{IG}}(x, \mathcal{D}) = I(f(x); Y | \mathcal{D}).$$

This acquisition function chooses points x where the observation $f(x)$ is expected to provide maximal information about Y , given the data \mathcal{D} observed so far.

In summary, Bayesian optimization and active learning share many principles. Both aim to learn about an unknown function and use probabilistic models to represent uncertainty. The main difference is that Bayesian optimization aims to find the maximum of the unknown function, while active learning aims to learn about the function as a whole. We will use ideas from both approaches to develop algorithms for RLHF.

2.3 REINFORCEMENT LEARNING

Reinforcement Learning (RL; [1]) is a broad framework for sequential decision-making under uncertainty. RL involves an agent interacting with an environment through trial and error. The agent receives a reward signal and aims to maximize cumulative reward over time.

The many solution methods for RL problems include linear programming [33], dynamic programming [34], and policy gradient methods [35]. More recently, many of these methods have been combined with deep neural networks, an approach commonly called *Deep RL* [36].

Constrained RL [37] extends the basic RL framework to decision-making settings with additional constraints. This framework is particularly relevant to many practical problems where certain safety, fairness, or resource constraints must be respected.

This section introduces RL and constrained RL and reviews standard solution approaches. We refer the reader to Sutton and Barto [1] for a more detailed introduction to RL. We refer to Altman [37] for an introduction to constrained RL.

2.3.1 Markov Decision Processes

Markov Decision Processes (MDPs) provide a model of sequential decision-making in dynamical systems. We distinguish between finite-horizon and infinite-horizon MDPs.

FINITE-HORIZON MDPs. A finite-horizon (or episodic) MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, H, \gamma, \mu_0, r)$, where \mathcal{S} is the state space, \mathcal{A} is the action space, $P : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the transition model, H is the time horizon, $\gamma \in [0, 1]$ is the discount factor, and μ_0 is the initial state distribution.

We describe an agent's behavior with a (possibly stochastic) policy $\pi : \mathcal{S} \times [H] \rightarrow \Delta(\mathcal{A})$. The reward function $r : \mathcal{S} \times \mathcal{A} \times [H] \rightarrow [0, r_{\max}]$ maps state-action-timestep triplets to a reward. The agent's goal is to maximize the expected cumulative return:

$$G(\pi) = \mathbb{E}_{P, \pi, \mu_0} \left[\sum_{t=0}^H \gamma^t r_t(s_t, a_t) \right].$$

If the state and action space are finite, we call the MDP *tabular* and denote the number of states by $|\mathcal{S}|$ and the number of actions by $|\mathcal{A}|$.

We often omit the dependence of the reward on the timestep, writing $r(s, a)$ instead of $r_h(s, a)$ and assuming a fixed reward function over time. Note that *w.l.o.g.*, we can model any initial state distribution as a single initial state by modifying the transitions. We sometimes use this to simplify notation and assume that μ_0 is supported on a single state.

INFINITE-HORIZON MDPS. An infinite-horizon MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \gamma, \mu_0, r)$ does not have a finite time horizon, but requires a discount factor $\gamma < 1$. The policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ and the reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow [0, r_{\max}]$ are now independent of the timestep. The agent's goal is still to find a policy π that maximizes the *expected discounted return* $G(\pi) = \mathbb{E}_{P, \pi, \mu_0} [\sum_{t=0}^{\infty} \gamma^t r](s_t, a_t)$.

CONNECTION TO MAB PROBLEMS. Note that we can view a MAB problem as an MDP with a single state and one action per arm. The transition model is deterministic, and the reward function is stochastic. The agent's goal to maximize the expected cumulative reward is equivalent to minimizing the MAB regret. In this view, MABs are a special case of RL problems.

OCCUPANCY MEASURES. The (discounted) *occupancy measure* is helpful to describe an agent's behavior in an MDP. In an infinite-horizon MDP, we define it as:

$$\mu_{\pi}(s, a) = \mathbb{E}_{P, \pi, \mu_0} \left[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}_{\{s_t=s, a_t=a\}} \right],$$

where $\mathbb{1}_{\{\cdot\}}$ denotes an indicator function. In particular, the occupancy measure $\mu_{\pi}(s, a)$ is the expected discounted frequency of policy π visiting state s and taking action a . Note that for finite state and action spaces, the return is linear in μ_{π} , i.e.,

$$G_r(\pi) = \sum_{s, a} \mu_{\pi}(s, a) r(s, a).$$

In finite-horizon MDPs, there are several ways to define the occupancy measure. For example, we can define $\mu_{\mathcal{M},\pi}^{h,h'}(s, a|s_0)$ as the probability of being in state s and taking action a at time $h' \geq h$ following policy π in MDP \mathcal{M} starting in state s_0 at time h . We can compute it recursively:

$$\begin{aligned}\mu_{\mathcal{M},\pi}^{h,h}(s', a'|s) &:= \pi_h(a'|s') \mathbb{1}_{\{s'=s\}}, \\ \mu_{\mathcal{M},\pi}^{h,h'+1}(s', a'|s) &:= \sum_{\tilde{s}, \tilde{a}} \pi_{h'}(a'|s') P(s'|\tilde{s}, \tilde{a}) \mu_{\mathcal{M},\pi}^{h,h'}(\tilde{s}, \tilde{a}|s).\end{aligned}$$

Sometimes it is useful to define a state occupancy measure by marginalizing the actions. We also use μ to denote this quantity, writing $\mu_\pi(s)$ in infinite-horizon MDPs, and $\mu_{\mathcal{M},\pi}^{h,h'}(s'|s)$ in finite-horizon MDPs.

VALUE FUNCTION. The *value function* of a policy π describes the expected return of the policy when starting in a specific state s at time h . In infinite-horizon MDPs, we define it as:

$$V_{\mathcal{M}}^\pi(s) = \mathbb{E}_{P,\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s \right].$$

In finite-horizon MDPs, we define the value function as:

$$V_{\mathcal{M}}^{\pi,h}(s) = \mathbb{E}_{P,\pi} \left[\sum_{h'=h}^H \gamma^{h'-h} r_t(s_t, a_t) \mid s_h = s \right].$$

Q-FUNCTION. The *Q-function* (or action-value function) of a policy π describes the expected return of taking action a in state s at time h , and then following policy π .

In infinite-horizon MDPs, we define the Q-function as:

$$Q_{\mathcal{M}}^\pi(s, a) = \mathbb{E}_{P,\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right].$$

In finite-horizon MDPs, define the Q-function as:

$$Q_{\mathcal{M}}^{\pi,h}(s, a) = \mathbb{E}_{P, \pi} \left[\sum_{h'=h}^H \gamma^{h'-h} r_t(s_t, a_t) \mid s_h = s, a_h = a \right].$$

Q-function and value function are related by

$$V_{\mathcal{M}}^{\pi,h}(s) = \sum_a \pi_h(a|s) Q_{\mathcal{M}}^{\pi,h}(s, a).$$

The *advantage function* is the difference between Q-function and value function $A_{\mathcal{M}}^{\pi,h}(s, a) = Q_{\mathcal{M}}^{\pi,h}(s, a) - V_{\mathcal{M}}^{\pi,h}(s)$. A policy π is optimal if and only if $A_{\mathcal{M}}^{\pi,h}(s, a) \leq 0$ for each all h, s, a . We denote the set of optimal policies for the MDP \mathcal{M} with $\Pi_{\mathcal{M}}^*$.

BELLMAN EQUATIONS. The Bellman equation provides a recursive formulation for the Q-function, value function, or other quantities. It establishes a fundamental link between a state-action pair's value and its successor's value. The most common form of the Bellman equation in the infinite-horizon setting is:

$$Q_{\mathcal{M}}^{\pi}(s, a) = r(s, a) + \gamma \sum_{s', a'} \pi(a'|s') P(s'|s, a) Q_{\mathcal{M}}^{\pi}(s', a').$$

The Bellman equation expresses the Q-value as the immediate reward for the current action plus the discounted expectation of the Q-values for the next state-action pairs. We can write similar Bellman equations for the value function, the advantage function, and other cumulative quantities, in finite- and infinite-horizon MDPs. The recursive definition of the occupancy measure we introduced before is also a Bellman equation. Bellman equations form the theoretical foundation for many Reinforcement Learning algorithms by enabling recursive computation of value functions.

2.3.2 Constrained Markov Decision Processes

A Constrained Markov Decision Process (CMDP; [37]) augments a standard MDP with a set of cost functions and associated thresholds.

This model applies to situations where the agent must not only maximize a reward but also consider certain constraints or restrictions.

In a CMDP, we have a set of cost functions c_1, \dots, c_n , each of which is defined similarly to the reward function, mapping state-action pairs to a cost value $c_j : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. Each cost function has an associated threshold ξ_1, \dots, ξ_n , which sets a maximum limit on the acceptable cumulative cost under each function.

The agent's goal in a CMDP is to maximize the expected discounted return, $G_r(\pi)$, while ensuring that the expected discounted cumulative costs satisfy $J_j(\pi) \leq \xi_j$ for all j .

The cumulative cost of a policy π for a cost function c_j is defined similarly to the expected return:

$$J_j(\pi) = \mathbb{E}_{P, \pi, \mu_0} \left[\sum_{t=0}^{\infty} \gamma^t c_j(s_t, a_t) \right].$$

The cumulative costs are also linear in the occupancy measure:

$$J_j(\pi) = \sum_{s,a} \mu_\pi(s, a) c_j(s, a).$$

CMDPs provide a powerful extension to MDPs. While introducing constraints results in additional complexity in finding optimal policies, it allows for a more nuanced representation of decision-making problems where rewards and constraints are important.

2.3.3 Linear Programming for MDPs and CMDPs

Tabular MDPs and CMDPs with small enough state and action spaces can efficiently be solved using linear programming (LP; [38]). There are several ways to formulate MDPs as LPs. This section introduces a formulation for infinite-horizon MDPs based on the occupancy measure. We then extend this problem to solve CMDPs, by introducing additional constraints on the occupancy measure.

Using the state-action occupancy measure, we can derive the following Bellman-like equation:

$$\mu_{\pi}(s, a) = \mu_0(s, a) + \gamma \sum_{s', a'} P(s|s', a') \pi(a|s) \mu_{\pi}(s', a')$$

Optimizing the expected return, which is linear in the occupancy measure, under the constraint implied by the Bellman equation yields the following linear program:

$$\begin{aligned} & \underset{\mu_{\pi}(s, a)}{\text{maximize}} && \sum_{s, a} \mu_{\pi}(s, a) r(s, a) \\ & \text{subject to} && \sum_a \mu_{\pi}(s, a) = \mu_0(s) + \gamma \sum_{s', a'} P(s|s', a') \mu_{\pi}(s', a'), \\ & && \mu_{\pi}(s, a) \geq 0, \end{aligned}$$

Solving this linear program yields the optimal state-action occupancy measure μ_{π^*} . From this, we can determine the optimal policy π^* by normalizing μ_{π^*} :

$$\pi^*(a|s) = \frac{\mu_{\pi^*}(s, a)}{\sum_{a'} \mu_{\pi^*}(s, a')}.$$

It is straightforward to extend this linear program to solve CMDPs. The objective function and the Bellman constraint remain the same, but we add linear constraints to ensure that the expected cumulative costs do not exceed the corresponding thresholds. This results in the following LP:

$$\begin{aligned} & \underset{\mu_{\pi}(s, a)}{\text{maximize}} && \sum_{s, a} \mu_{\pi}(s, a) r(s, a) \\ & \text{subject to} && \sum_a \mu_{\pi}(s, a) = \mu_0(s) + \gamma \sum_{s', a'} P(s|s', a') \mu_{\pi}(s', a'), \\ & && \sum_{s, a} \mu_{\pi}(s, a) c_j(s, a) \leq \xi_j, \text{ for each constraint } j, \\ & && \mu_{\pi}(s, a) \geq 0. \end{aligned}$$

This LP approach is particularly useful for small state and action spaces, where the LP can be solved efficiently. However, this approach

becomes intractable for large or continuous state or action spaces. Fortunately, there are many alternative solution methods for MDPs and CMDPs, some of which we discuss next.

2.3.4 *Dynamic Programming*

Dynamic Programming (DP) is a powerful method for solving MDPs by iteratively improving estimates of the value function or Q-function and improving a policy based on these estimates. This section provides a brief overview of the most common dynamic programming methods in RL: *Policy Iteration* and *Value Iteration*. For a more detailed introduction to dynamic programming in the context of RL and optimal control, we refer to Bertsekas [34]. Extending dynamic programming methods to CMDPs is more difficult than extending LP methods, and we do not cover it here. Instead, we refer to Altman [37] for some discussion.

POLICY ITERATION. *Policy iteration* alternates between evaluating and improving the current policy based on a value estimate.

In the policy evaluation step, we estimate the value function for the current policy π . In a tabular environment with a known transitions function, we can do this exactly by solving a system of linear equations resulting from the Bellman equation

$$V_{\mathcal{M}}^{\pi}(s) = \sum_a \pi(a|s) \left(r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^{\pi}(s') \right),$$

for all states s .

In the policy improvement step, we update the policy to greedily choose actions that maximize the estimated value at each state. Specifically, the new policy π' is given by:

$$\pi'(a|s) = \operatorname{argmax}_a \left(r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^{\pi}(s') \right).$$

Policy iteration repeats these two steps until convergence. The algorithm is guaranteed to converge to an optimal policy and value function.

VALUE ITERATION. Value Iteration combines policy evaluation and policy improvement into a single step. Instead of computing a policy from the current value function estimate, Value Iteration updates the value function directly as:

$$V(s) \leftarrow \max_a \left(r(s, a) + \gamma \sum_{s'} P(s'|s, a) V(s') \right).$$

Value Iteration repeats this update for all states until the value function converges, and only then constructs a policy that acts greedily w.r.t. the value function estimate. Value Iteration is also guaranteed to converge to an optimal value function and policy.

2.3.5 Q-Learning

Linear programming and dynamic programming are *planning* algorithms that can solve MDPs with known transition dynamics. However, the transition dynamics can typically only be explored by interacting with the environment. In *model-based* RL, we estimate a transition model using interactions with the environment and then use a planning method with the estimated model.

Model-free RL is an alternative approach that does not construct an explicit model of the environment. One popular model-free RL algorithm is *Q-learning* [39], which learns a Q-function directly by interacting with the environment.

Q-Learning collects trajectories in the environment using an exploration policy and then updates an estimated Q-function according to the following rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right).$$

where, $\alpha \in (0, 1)$ is a learning rate, and $\max_{a'} Q(s', a')$ represents the current estimate of the optimal future value.

Commonly the exploration policy is an “ ϵ -greedy” policy, which chooses a random action with probability ϵ and the greedy action w.r.t. the current Q-function estimate with probability $1 - \epsilon$.

Under some assumptions that ensure sufficient exploration and a decaying learning rate, Q-learning converges to the optimal Q-function [39]. We can then extract an optimal policy by acting greedily w.r.t. the learned Q-function.

FEATURE APPROXIMATION. In practice, the state and action spaces are often too large to represent the Q-function as a table. Function approximation can address this by parameterizing the Q-function $Q(s, a; \theta)$ by a set of parameters θ .

A popular choice for the parameterization is a deep neural network, which results in an algorithm called *Deep Q-Network* (DQN; [40]). In DQN, we use stochastic gradient descent to find the parameters θ that minimize the following loss function at each iteration i :

$$L_i(\theta_i) = \mathbb{E}_{P, \pi, \mu_0} [(y_i - Q(s, a; \theta_i))^2],$$

where y_i is the target for iteration i , defined as

$$y_i = \mathbb{E}_{P, \pi, \mu_0} \left[r(s, a) + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \right].$$

The expectations are typically approximated using Monte-Carlo samples, i.e., as the average over a finite number of trajectories from policy π in the environment. The target y_i depends on the previous iteration’s Q-value parameters θ_{i-1} , leading to an iterative updating scheme called *bootstrapping*. Simultaneously using function approximation and bootstrapping causes Q-learning to lose theoretical convergence guarantees [1]. However, DQN can work well in many domains in practice if combined with a few heuristic modifications to stabilize learning [40].

2.3.6 Policy Optimization

Policy optimization algorithms directly optimize the policy to maximize the expected return and have shown great success in complex, high-dimensional environments. Central to most policy optimization methods is the *policy gradient theorem* [35], which provides an analytical form for the gradient of the expected return w.r.t. the policy parameters. The theorem states that the gradient of the expected return is given by

$$\nabla_{\theta} G_r(\pi_{\theta}) = \mathbb{E}_{P, \pi, \mu_0} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q_{\mathcal{M}}^{\pi_{\theta}}(s, a)].$$

Here, $\pi_{\theta}(a|s)$ is the policy parameterized by θ , and $\mu_{\pi_{\theta}}$ is the stationary occupancy measure under π_{θ} . We can iteratively improve the policy by estimating the expectation using sampled trajectories and updating the policy parameter using the policy gradient. This “vanilla” policy gradient method is the foundation for many policy optimization methods. However, this basic algorithm can often be sample inefficient and unstable, which has led to a wide range of improved policy optimization methods.

Trust Region Policy Optimization (TRPO; [41]) optimizes the policy while constraining the amount by which the new policy can deviate from the old policy. This constraint stabilizes the learning by ensuring the policy does not change too drastically at each update.

Proximal Policy Optimization (PPO; [42]) is a simplification of TRPO that has similar performance but is more straightforward to implement and tune. Instead of using a constraint optimization problem to remain in a trust region, PPO clips the policy ratio to encourage small policy updates.

Constrained Policy Optimization (CPO; [43]) extends the idea of TRPO to constrained MDPs. It optimizes a policy to maximize the expected return while satisfying constraints.

There are many more algorithms based on policy optimization, and we refer the reader to Chapter 5 of François-Lavet *et al.* [36] for a more detailed overview.

2.4 REWARD LEARNING

In RL, we must specify a reward for each state-action pair to evaluate the agent’s performance. However, specifying such a reward function is often challenging. Rather than manually designing a reward function, *reward learning* involves learning a reward model from data [44]. The data can, for example, be expert demonstrations, feedback from human operators, or preferences between different actions or trajectories. We can then use the learned reward model to learn a policy with any RL algorithm.

Reinforcement Learning from Human Feedback (RLHF) refers to any RL method that learns from human feedback instead of an explicit reward function. Reward learning is the most common way to implement RLHF, and we use both terms interchangeably in this dissertation. In this section, we introduce two popular methods used for reward learning: Inverse Reinforcement Learning (IRL) and Preference Learning.

2.4.1 *Inverse Reinforcement Learning*

Inverse Reinforcement Learning (IRL) aims to infer a reward function from expert demonstrations. In IRL, we are given an MDP without a reward function \mathcal{M} (short $\text{MDP} \setminus \text{R}$) and a set of expert demonstrations \mathcal{D} provided by an expert policy π^E . We want to infer a reward function \hat{r} that explains the expert demonstrations.² In particular, we want the expert policy π^E to be optimal in the MDP augmented with the inferred reward function $\mathcal{M} \cup \hat{r}$. Solving this problem is equivalent to finding a reward function such that the optimal policy matches the expert policy’s state occupancy measure or feature expectations if the reward function is linear [45].

IRL is closely related to imitation learning. Whereas imitation learning directly tries to mimic the expert’s behavior, IRL aims to infer the reward function that guides the expert’s behavior. We can use IRL

² Sometimes, we assume full access to the expert policy π^E instead of demonstrations.

for imitation learning. However, by inferring a reward function, IRL has the potential to generalize beyond the expert’s demonstrations.

Crucially, the IRL problem is underspecified: infinitely many reward functions can explain a set of demonstrations. For instance, a reward function that assigns a constant reward to every state-action pair makes any given policy optimal. The various IRL methods proposed in the literature reflect different approaches to resolving this ambiguity, often by introducing additional assumptions or regularizations.

MAXIMUM MARGIN IRL. Ng and Russell [46] propose to resolve the ambiguity of IRL by finding the reward function \hat{r} that maximizes the *margin* by which the expert policy is optimal. To be optimal, the expert policy must satisfy $\hat{V}^{\pi^E}(s) \geq \hat{Q}^{\pi^E}(s, a)$ for all states s and actions a . Hence, we can define the optimality margin as $\sum_{s,a} (\hat{V}^{\pi^E}(s) - \hat{Q}^{\pi^E}(s, a))$. Here \hat{V}^{π^E} is the value function of the expert policy computed under inferred reward \hat{r} , and $\hat{Q}^{\pi^E}(s, a)$ is the corresponding Q-function. We can formalize the goal of maximum margin IRL as solving the optimization problem

$$\begin{aligned} & \underset{\hat{r}, \zeta}{\text{maximize}} && \sum_{s,a} \zeta(s, a) \\ & \text{subject to} && \hat{V}^{\pi^E}(s) - \hat{Q}^{\pi^E}(s, a) \geq \zeta(s, a), \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A} \\ & && 0 \leq \hat{r}(s, a) \leq r_{\max} \end{aligned} \quad (2.3)$$

where $\zeta(s, a)$ are the optimization variables representing the optimality margins. For tabular MDPs, this is a linear program because both \hat{V}^{π^E} and \hat{Q}^{π^E} are linear in the state-action occupancy vectors.

MAXIMUM (CAUSAL) ENTROPY IRL. Ziebart *et al.* [47] propose an IRL algorithm based on the maximum entropy principle, a well-established idea in statistical physics and information theory. The maximum entropy principle states that the most appropriate distribution to estimate under given constraints is the one with the maximum entropy, intuitively the “most uncertain” distribution.

We can use the maximum entropy principle to resolve the ambiguity of the IRL problem by choosing the reward function \hat{r} that maximizes the entropy of the expert. The goal is to infer a reward function that makes the optimal policy occupancy measure match the expert’s occupancy measure and maximizes the entropy over trajectories. More formally, Maximum Entropy RL consists of solving the following optimization problem:

$$\begin{aligned} & \underset{\hat{r}}{\text{maximize}} && H(\hat{\pi}) = \mathbb{E}_{\tau \sim \hat{\pi}} [-\log P(\tau)] \\ & \text{subject to} && \mu_{\hat{\pi}}(s, a) = \mu_{\pi^E}(s, a) \quad \forall s, a \end{aligned}$$

where $\hat{\pi}$ is the optimal policy for \hat{r} and $P(\tau)$ is the probability distribution over trajectories implied by rolling out $\hat{\pi}$ in the environment.

This formulation implicitly assumes that the expert, while trying to optimize the underlying reward function, also exhibits a preference for more stochastic trajectories that have higher entropy. This assumption can be problematic in stochastic environments, where it implies the expert seeks out more random transitions [48].

Maximum Causal Entropy IRL is a more nuanced way of using the maximum entropy principle in IRL. It avoids issues with stochastic transitions by considering the causality of the decision-making process. Instead of maximizing the entropy over trajectories, Maximum Causal Entropy IRL maximizes the entropy of the future action choices conditioned on the past at each timestep. This leads to a slightly modified optimization problem:

$$\begin{aligned} & \underset{\hat{r}}{\text{maximize}} && H_{\text{causal}}(\hat{\pi}) = \mathbb{E}_{P, \pi, \mu_0} \left[- \sum_{t=0}^T \log \hat{\pi}(a_t | s_t) \right] \\ & \text{subject to} && \mu_{\hat{\pi}}(s, a) = \mu_{\pi^E}(s, a) \quad \forall s, a \end{aligned}$$

The key difference to Maximum Entropy IRL is that we only consider the entropy of the policy, i.e., the agent’s choices, and exclude all uncertainty due to the environment dynamics. Maximum Entropy IRL and Maximum Causal Entropy IRL are equivalent in determin-

istic environments. In stochastic environments, the causal entropy variant often yields superior results.

We can solve both Maximum Entropy IRL and Maximum Causal Entropy IRL by maintaining a reward function and a policy estimate and alternating between updating them. First, we update the policy to be optimal for the current reward estimate. For Maximum Entropy IRL, we can use any standard RL algorithm in this step. Second, we update the reward estimate based on the discrepancy between the expert and the current policy. For a tabular environment, we can update the reward function according to:

$$\hat{r}(s, a) \leftarrow \hat{r}(s, a) + \alpha (\mu_{\hat{\pi}}(s, a) - \mu_{\pi^E}(s, a))$$

where α is a learning rate. We can straightforwardly extend this gradient update to non-tabular environments with linear reward functions, the most common setting in which Maximum Entropy IRL is applied. Repeating both steps until convergence solves the Maximum Entropy IRL problem. We can use a similar algorithm for Maximum Causal Entropy IRL but need to modify the policy update. For details on this, we refer to Ziebart *et al.* [47] or Gleave and Toyer [48].

2.4.2 Preference Learning

IRL is a powerful approach to learning reward functions from human behavior, but it is limited to situations where expert demonstrations are readily available. Demonstrating a task can be challenging or even impossible in many cases, especially in high-dimensional or complex environments or when the task requires specialized expertise. Furthermore, the quality of the inferred reward function is bounded by the quality of the demonstrations provided, making it difficult to exceed human-level performance using IRL.

Alternatively, we can learn directly from evaluative feedback on agent behavior, a paradigm we refer to as preference learning. This feedback can be of different types, including numerical ratings, verbal feedback, or comparisons between possible actions or trajectories.

HUMAN REINFORCEMENT. The most direct way to learn from human feedback is to directly *shape* a policy using human reinforcement. The feedback comes as a positive or negative *reinforcement* signal provided by humans that rewards or punishes specific actions. *TAMER* [49] assumes that human reinforcement directly indicates the long-term value of actions in a short time window before the signal. This assumption makes credit assignment easier and allows for sample efficient learning. However, it can be violated in practice, as humans tend to give feedback depending on the current policy of the agent [50]. Instead of direct policy shaping, most recent approaches to RLHF learn reward models from human feedback. Reward models are often particularly compact representations of the desired behavior and tend to require less feedback than directly shaping a policy.

EVALUATIONS. Similar to direct human reinforcement, we can learn reward models from evaluations of agent behavior, sometimes called *critiques* [51]. Evaluations include, for example, numerical evaluations (e.g., [52]) or binary labels indicating whether the behavior was good or bad (e.g., [53]). We can learn a reward model from such evaluations using standard supervised learning techniques. However, interpreting the ratings can be challenging: the scale used by the rater is often arbitrary, and it can shift over time, making it difficult to compare ratings or use them to train an agent.

PAIRWISE COMPARISONS. A promising alternative is to learn from pairwise comparisons [54]. Rather than providing an absolute rating, the human evaluator only indicates which of two actions or trajectories they prefer. This approach can simplify the task for the human evaluator, and the relative nature of the feedback can mitigate some of the challenges associated with interpreting ratings. Wirth *et al.* [55] provide a comprehensive overview of preference-based RL methods. Christiano *et al.* [12] popularized learning a reward model

from pairwise comparisons. They use the Bradley-Terry model [56] for the probability of a human preferring one of two trajectories:

$$p(\tau_1 > \tau_2 | \theta) = \frac{\exp(r_\theta(\tau_1))}{\exp(r_\theta(\tau_1)) + \exp(r_\theta(\tau_2))}.$$

where τ_1 and τ_2 are two trajectories or trajectory segments to compare and $r(\tau_1)$ and $r_\theta(\tau_2)$ denote the sum of rewards along the trajectories under the reward model parameterized by θ . Based on this model, Christiano *et al.* [12] learn a deep neural network to represent the reward function from human data via stochastic gradient descent. They manage to learn complex behaviors in simulated robotics environments using learned reward models. Since then, learning reward models from pairwise comparisons has shown great promise in a variety of domains, including robotics [57], video games [58], and large language models [14].

2.5 OVERVIEW OF NOTATION

In this dissertation, we build on work from multiple fields, such as multi-armed bandits, Bayesian optimization, and Reinforcement Learning, among others. Unfortunately, there are sometimes clashes between the standard notation in these fields, forcing us to choose non-standard notation. As a reference, Table 2.1 provides an overview of notation shared throughout the thesis, and Tables 2.2 to 2.5 provide an overview of notation specific to each chapter.

Symbol	Description
\mathcal{M}	Markov Decision Process
\mathcal{C}	Constrained Markov Decision Process
\mathcal{S}	State space
$ \mathcal{S} $	Number of states (for finite state space)
s, s_t	Single state (at time t)
\mathcal{A}	Action space
$ \mathcal{A} $	Number of actions (for finite action space)
a, a_t	Single action (at time t)
r	Reward function
r_{\max}	Upper bound on reward
P	Transition probability function
μ_0	Initial state distribution
γ	Discount factor
H	Time horizon
π	Policy
$G(\pi), G_r(\pi)$	Expected return of policy π (computed using reward function r)
c	Cost function
ξ	Cost threshold
$J(\pi)$	Expected cumulative cost of policy π
Q	Q-function
V	Value function
A	Advantage function
τ	Trajectory
μ	Occupancy measure

TABLE 2.1: General notation for Reinforcement Learning.

Symbol	Description
Π_c	Candidate policy set
\mathcal{Q}_c	Candidate query set
$\mathbf{f}(\pi)$	(Estimated) State occupancy measure of policy π
\mathcal{D}	Dataset of observations
\hat{r}	Belief about the reward function
$\hat{G}(\pi)$	Belief about the expected return of policy π

TABLE 2.2: Specific notation for Chapter 3.

Symbol	Name
$\mathcal{M} \cup r$	MDP with reward r
$\widehat{\mathcal{M}}$	Estimated MDP
\hat{r}	Estimated reward function
π^E	Expert policy
C	Reward confidence interval
$\hat{\Pi}$	Policy confidence interval
$\mathcal{R}_{\mathfrak{B}}$	Exact feasible set
$\mathcal{R}_{\hat{\mathfrak{B}}}$	Approximate feasible set

TABLE 2.3: Specific notation for Chapter 4.

Symbol	Description
\mathcal{F}	True safe set
S	Safe set
θ	Reward parameter
ϕ	Constraint parameter
r_{eval}	Evaluation reward
\mathbf{f}	Feature function
\mathcal{R}	Regret measure
k	Number of demonstrations
n	Number of constraints
m	Number of inferred constraints
n_{traj}	Number of trajectories

TABLE 2.4: Specific notation for Chapter 5.

Symbol	Description
ν	CBAI problem instance
θ	Reward parameter
ϕ	Constraint parameter
x, \mathcal{X}	Action, Action set
T	Exploration budget
$\mathcal{X}_{\theta}^{\geq}, \mathcal{X}_{\theta}^{>}$	(Strictly) better action set w.r.t reward parameter
λ, A_{λ}	Design, Design matrix
τ	Stopping time
\mathcal{U}	Uncertain set
S	Safe set

TABLE 2.5: Specific notation for Chapter 6.

Part I

ACTIVE LEARNING FOR REWARD LEARNING

INFORMATION DIRECTED REWARD LEARNING

We start our investigation of algorithms for RLHF by studying active learning for reward models. Since human feedback is expensive, *active reward learning* aims to minimize the number of samples necessary to learn a good reward model. Prior work typically focuses on approximating the reward function uniformly well, which may not be aligned with the original goal of RL: *finding an optimal policy* (see Figure 3.1). Moreover, prior work is often tailored to specific types of queries, such as comparisons of two trajectories (e.g., [22]) or numerical evaluations of trajectories (e.g., [52]), limiting its applicability.

In this chapter, we address these limitations. In particular:

- We propose *Information Directed Reward Learning* (IDRL), a general active reward learning approach with the goal of *finding a good policy* rather than uniformly reducing the reward model's error. IDRL can use arbitrary Bayesian reward models and arbitrary types of queries (Section 3.3).
- We describe an exact and efficient implementation of IDRL using Gaussian process (GP) reward models (Section 3.4) and different types of queries.
- We show there are close connections between IDRL with a GP reward model and provably optimal algorithms in multi-armed bandits (Section 3.5).
- We propose an approximation of IDRL that uses a deep neural network reward model and a state-of-the-art policy optimization algorithm to learn from comparison queries (Section 3.6).
- We evaluate IDRL extensively in simulated environments (Section 6.3), including a driving task and high-dimensional continuous control tasks in the MuJoCo simulator. We find that

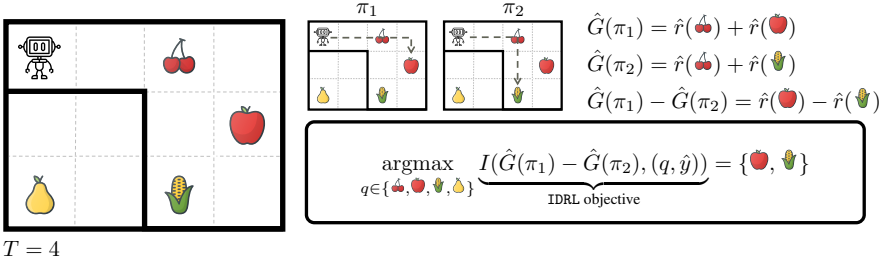


FIGURE 3.1: Illustration of the active reward learning setting considered in this chapter. The robot wants to collect food for a human. It can only move 4 timesteps in the gridworld, cannot pass through the black walls, and collecting more food is always better. The robot does not know the human’s preferences but can ask for food ratings. Common active learning methods aim to learn the reward uniformly and query all items similarly often. In contrast, IDRL considers only the two plausibly optimal policies π_1 and π_2 . Since both policies collect the cherry and do not collect the pear, the robot only needs to learn about the apple and the corn. IDRL can solve the task with 2 queries instead of 4.

both implementations of IDRL significantly outperform prior methods.

Overall, we demonstrate that IDRL is a promising approach to active learning for RLHF beyond basic uncertainty sampling.

The contents of this chapter are in large parts based on:

D. Lindner, M. Turchetta, S. Tschitschek, K. Ciosek, and A. Krause, “Information directed reward learning for reinforcement learning”, in *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

3.1 RELATED WORK

Most work learning reward models for reinforcement learning uses simple heuristics to select queries to make (e.g., [12, 58]). In contrast,

	Non-linear Rewards	Single-state Queries	Trajectory Queries	Numerical Queries	Comparison Queries	Considers Env. Dynamics
Sadigh <i>et al.</i> [22]	✗	✗	✓	✗	✓	✗
Bıyık <i>et al.</i> [59]	✗	✗*	✓	✗*	✓	✗
Bıyık <i>et al.</i> [60]	✓	✗*	✓	✗*	✓	✗
Daniel <i>et al.</i> [52]	✓	✗*	✓	✓	✗*	✓
Wilde <i>et al.</i> [61]	✗	✗	✓	✗	✓	✓
IDRL (ours)	✓	✓	✓	✓	✓	✓

*The authors do not consider this setting, but we provide an extension to their method in Section 6.3 and Appendix A.2.

TABLE 3.1: In contrast to most prior work on active reward learning for RL, IDRL can handle non-linear reward functions and different query types, in particular numerical evaluations and comparisons of individual states and (partial) trajectories. Further, IDRL takes the environment dynamics into account to achieve better sample efficiency (cf. Figure 3.1).

active reward learning aims to select the most informative queries in a principled way. For linear reward functions, Sadigh *et al.* [22] ask the expert to compare trajectories synthesized to maximize the volume removed from a hypothesis space. Bıyık *et al.* [59] argue that maximizing information gain leads to better sample efficiency and queries that are easier to answer than volume removal. Bıyık *et al.* [60] generalize maximizing information gain to non-linear reward functions using a GP model. We also use an information gain objective to select queries; however, our approach focuses on finding an optimal policy instead of uniformly reducing the error of the reward model. With a similar motivation, Wilde *et al.* [61] aim to capture how informative a query is for distinguishing policies. However, their method is limited to comparisons between potentially optimal policies. Daniel *et al.* [52] also introduce an acquisition function to measure how informative a query is for learning a good policy. However, their setting is restricted to observing the cumulative reward of a trajectory, and their acquisition function is computationally expensive. Table 3.1 gives an overview of how our method compares to this prior work.

3.2 PROBLEM SETTING

We focus on MDPs where the reward function is not readily available. Instead, the agent can query an expert for information about the reward. In iteration i , the agent makes a query q_i to the expert and receives a response y_i . For example, q_i could ask the expert to compare two trajectories or judge a single trajectory, and y_i could indicate which of the two trajectories is better or provide the return of a single trajectory. We assume that the agent can interact with the environment cheaply, but queries to the expert are *expensive*. Hence, the agent has to find a policy π that *maximizes the expected return* $G(\pi)$ using *as few expert queries as possible*.

We approach this problem by learning a model of the reward function, i.e., a model that predicts the reward of a given state,¹ and computing a policy that maximizes the return induced by the model. Importantly, we want to learn a reward model such that the induced optimal policy achieves a high return under the true reward function. Note that we can use any RL algorithm to find the policy. Hence, the problem reduces to selecting a model for the reward function and deciding which queries to make. Our key insight is that queries that help most to find a good policy might differ from those that uniformly reduce the model’s uncertainty.

3.3 INFORMATION DIRECTED REWARD LEARNING

This section introduces Information Directed Reward Learning (IDRL) for a general Bayesian model of the reward and discusses how to select queries q_i , making no assumptions on their form nor on the responses y_i .

REWARD MODEL. To select informative queries, we need to quantify uncertainty; hence, we use a Bayesian model of the reward

¹ Our approach also works for reward functions that depend on state-action pairs or transitions. We focus on state-dependent rewards for simplicity of exposition.

function. From a Bayesian perspective, it is important to distinguish between the agent’s belief about a quantity and its “actual” value that is unknown to the agent. We denote the belief about the reward in state s with $\hat{r}(s)$ and its actual value with $r(s)$.

QUERY SELECTION. To select informative queries, we have to consider that the responses might only give *indirect* information about the set of optimal policies. For example, assume the agent can ask the expert to quantify the reward of individual states. These rewards provide information about the expected return of a policy but may yield no information about the set of optimal policies. For example, if a state is visited similarly often by every plausibly optimal policy, knowing its reward does not help decide between the policies (e.g., the cherry in Figure 3.1). Therefore, any approach that only aims to reduce the uncertainty of the reward model may waste expensive queries that do not help find an optimal policy.

Intuitively, we want to select instead queries that *help identify the optimal policy*. In the language of information theory, we want to maximize the *information gain* of a query about *the identity of the optimal policy*. More formally, if $\mathcal{D} = \{(q_1, y_1), \dots, (q_t, y_t)\}$ is a dataset of past queries and responses, let us denote with $P(\hat{\pi}^* | \mathcal{D})$ the agent’s belief about the optimal policy, induced by our belief about the reward function \hat{r} . Also, let \mathcal{Q}_c be a set of candidate queries the agent can make. Then, we could select queries

$$q^* \in \operatorname{argmax}_{q \in \mathcal{Q}_c} I(\hat{\pi}^*; (q, \hat{y}) | \mathcal{D}),$$

where \hat{y} is the agent’s belief about the response it will get to query q , and I denotes the information gain. Unfortunately, this objective has two undesirable properties. First, the agent has to keep track of a distribution over all possible policies to compute it, which is intractable in general. Second, reducing uncertainty about the optimal policy matters only if there are significant differences in the return of plausibly optimal policies. For example, suppose the agent identifies a set of plausibly optimal policies with similar returns. In that case,

we care less about identifying which of these policies is optimal than when the plausibly optimal policies have very different returns.

To address the first challenge, we obtain a finite set of *candidate policies* Π_c that are *plausibly optimal* according to our Bayesian reward model. To address the second challenge, we select the most informative query for distinguishing policies in terms of their value.

Let us first discuss how to select queries, assuming a set of plausibly optimal policies Π_c is available. We can exploit that the belief about the reward function \hat{r} induces a belief about the expected return of policy $\pi \in \Pi_c$, denoted as $\hat{G}(\pi)$. In a tabular environment, we can compute the expected return of a policy as the scalar product $G(\pi) = \mathbf{f}(\pi)^T \mathbf{r}$, where $\mathbf{f}(\pi)$ is a vector of the (discounted) state occupancy measure of policy π and \mathbf{r} is a vector of rewards of the corresponding states. In practice, we can estimate $\mathbf{f}(\pi)$ from trajectories sampled using policy π . This approach also allows us to compute $\hat{G}(\pi)$ from \hat{r} .

Given $\hat{G}(\pi)$, IDRL proceeds in two steps. It first selects two policies that maximize the model’s uncertainty about the difference in their expected returns:

$$\pi_1, \pi_2 \in \underset{\pi, \pi' \in \Pi_c}{\operatorname{argmax}} H(\hat{G}(\pi) - \hat{G}(\pi') | \mathcal{D}), \quad (3.1)$$

where H is the entropy of the belief conditioned on past queries. To gather information to distinguish between π_1 and π_2 , IDRL then selects queries that maximize the information gain about the difference in expected return between π_1 and π_2 :

$$q^* \in \underset{q \in \mathcal{Q}_c}{\operatorname{argmax}} I(\hat{G}(\pi_1) - \hat{G}(\pi_2); (q, \hat{y}) | \mathcal{D}). \quad (3.2)$$

Note that this is *not* the same as jointly maximizing the information gain about $\hat{G}(\pi_1)$ and $\hat{G}(\pi_2)$. Equation (3.2) prefers queries that help to distinguish π_1 and π_2 over queries that help to determine the exact value of $\hat{G}(\pi_1)$ and $\hat{G}(\pi_2)$. Assuming Π_c contains an optimal policy, reducing the uncertainty about the difference in returns within Π_c will help quickly find an optimal policy. In particular, if there is

no remaining uncertainty about the differences in return, we can unambiguously identify an optimal policy.

Let us now discuss how to obtain a set of candidate policies Π_c . For IDRL to select informative queries, Π_c has to reflect the agent’s current belief about optimal policies. We use Thompson sampling (TS; [62]) as a flexible way to create Π_c . We implement TS by repeatedly sampling a reward function from the posterior reward model and finding an optimal policy for this sampled reward function. This approach approximates sampling from the posterior distribution over optimal policies. Since TS is demanding, in our experiments, we investigate two ways to alleviate its computational burden: (1) we update Π_c in regular intervals rather than at every step, and (2) we start from the candidates computed in the previous steps rather than starting the policy optimization from scratch.

Algorithm 1 shows the full IDRL algorithm. In each iteration, IDRL identifies two plausibly optimal policies with high uncertainty about their difference in return and then aims to reduce this uncertainty. We can stop the algorithm after a fixed number of queries or by checking a convergence criterion and return a policy $\bar{\pi}^*$ that is optimal for the current reward model.

Note that IDRL is agnostic to how the candidate queries \mathcal{Q}_c are generated. Different applications might require different approaches to generating \mathcal{Q}_c . In our experiments, for example, we consider: (1) using all possible queries in small environments; (2) choosing states or trajectories to query from rollouts of the currently optimal policy $\bar{\pi}^*$; (3) selecting queries from rollouts of the candidate policies; and (4) selecting queries from trajectories of a pre-defined explorations policy. Importantly, all of these, and others, are compatible with IDRL.

3.4 IDRL FOR GP REWARD MODELS

Next, we describe our first concrete implementation of IDRL, which uses a Gaussian process (GP) reward model and linear query types.

Algorithm 1 *Information Directed Reward Learning (IDRL)*. The algorithm requires a set of candidate queries \mathcal{Q}_c , a Bayesian model of the reward function, and an RL algorithm that returns a policy given a reward function. $\widehat{G}(\pi)$ is the belief about the expected return of policy π , induced by the reward model $P(\hat{r}|\mathcal{D})$, and \hat{r} is the belief about the reward function.

```

1:  $\mathcal{D} \leftarrow \{\}$ 
2:  $\Pi_c \leftarrow$  initialize candidate policies
3: Initialize reward model with prior distribution  $P(\hat{r})$ 
4: while not converged do
5:   Select a query:
6:      $\pi_1, \pi_2 \in \operatorname{argmax}_{\pi, \pi' \in \Pi_c} H(\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D})$ 
7:      $q^* \in \operatorname{argmax}_{q \in \mathcal{Q}_c} I(\widehat{G}(\pi_1) - \widehat{G}(\pi_2); (q, \hat{y})|\mathcal{D})$ 
8:   Make query and update reward model:
9:      $y^* \leftarrow$  Response to query  $q^*$ 
10:    Update model:  $P(\hat{r}|\mathcal{D} \cup \{(q^*, y^*)\}) \propto P(y^*|\hat{r}, \mathcal{D}, q^*)P(\hat{r}|\mathcal{D})$ 
11:     $\mathcal{D} \leftarrow \mathcal{D} \cup \{(q^*, y^*)\}$ 
12:   Optionally update candidate policies  $\Pi_c$ 
13:  $\bar{r} \leftarrow$  mean estimate of the reward model
14:  $\bar{\pi}^* \leftarrow$  RL( $\bar{r}$ )
15: return  $\bar{\pi}^*$ 

```

These choices allow us to compute equations (3.1) and (3.2) exactly and efficiently.

REWARD MODEL. We model the reward function as a GP with (*w.l.o.g.*) a zero-mean prior distribution $\hat{r}(s) \sim \mathcal{GP}(0, k(s, s'))$ using a kernel k which measures the similarity of states.

QUERY SELECTION. We first show how to compute equations (3.1) and (3.2) if the posterior belief about the reward function is Gaussian. Then, we discuss a family of practically relevant query types that satisfy this assumption. We provide proofs for all results in Appendix A.1.

Proposition 3.4.1. If $\hat{r}(s)|\mathcal{D}$ is a GP, then $P(\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D})$ is Gaussian and:

$$\begin{aligned} \operatorname{argmax}_{\pi, \pi' \in \Pi_c} H(\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D}) &= \operatorname{argmax}_{\pi, \pi' \in \Pi_c} \operatorname{Var}[\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D}] \\ \operatorname{argmax}_{q \in \mathcal{Q}_c} I(\widehat{G}(\pi_1) - \widehat{G}(\pi_2); (q, \hat{y})|\mathcal{D}) &= \operatorname{argmin}_{q \in \mathcal{Q}_c} \operatorname{Var}[\widehat{G}(\pi_1) - \widehat{G}(\pi_2)|\mathcal{D} \cup \{(q, \hat{y})\}] \end{aligned}$$

We can compute both variances analytically, enabling exact implementation of equations (3.1) and (3.2).

QUERY TYPES. To apply this result, we need $\hat{r}(s)|\mathcal{D}$ to be a GP, which is not the case for general observations (q_i, y_i) . If the queries are individual states, i.e., $q_i = s_i$ and $y_i = r(s_i)$, the problem is standard GP regression, and $\hat{r}(s)|\mathcal{D}$ is a GP. More generally, we can show a similar statement if the observations are *linear* combinations of rewards.

Definition 3.4.1. We call $q = (X, C)$ a *linear reward query*, if it consists of states $X = \{s_1, \dots, s_N\}$ and linear weights $C = \{c_1, \dots, c_N\}$, and the response to query q is a linear combination of rewards $y = \sum_{j=1}^N c_j r(s_j) + \eta$, with Gaussian noise $\eta \sim \mathcal{N}(0, \sigma_n^2)$.

Proposition 3.4.2. Let q be a linear reward query. If the prior belief about the reward $\hat{r}(s)$ is a GP, then the posterior belief about the reward $\hat{r}(s)|(q, y)$ is also a GP.

Linear reward queries result in a particularly efficient implementation of IDRL. Of course, IDRL with a GP model could be extended to non-linear observations using approximate inference, similar to Büyük *et al.* [60]. However, many commonly used query types can be modeled as linear reward queries, including the return of trajectories or comparisons of trajectories. Let us discuss a few query types in more detail.

SINGLE STATE REWARDS. If $N = 1$, a linear query consists of a single state $q_i = s_i \in \mathcal{S}$, for which the expert provides a noisy reward.

RETURN OF TRAJECTORIES. For $N > 1$ and all $c_i = 1$, the agent observes the sum of rewards of states. The set X could contain the states in a trajectory or a sub-sequence of it. Then, the queries ask about the return, i.e., the sum of rewards, of this sequence of states.

COMPARISONS OF STATES AND TRAJECTORIES. We can model a comparison of the reward in states s_a and s_b by defining $X = \{s_a, s_b\}$ and defining $C = \{1, -1\}$. Then the agent might observe $y = r(s_a) - r(s_b)$. In practice, comparison queries usually result in binary feedback, i.e., the expert states that either s_a or s_b is preferred. We can model this, e.g., with a Bernoulli distribution $P(y = 1) = (1 + r(s_a) - r(s_b))/2$, if all rewards are between 0 and 1. The observations from this distribution have expectation $r(s_a) - r(s_b)$ and the noise model is sub-Gaussian, which we can approximate with a Gaussian noise distribution (cf. [63]). Hence, we can model such comparison queries as linear reward queries. We can model comparisons between two sets of states, e.g., between two trajectories, analogously. Other observation models for comparisons have been proposed in the literature, such as softmax [22], probit [60] or Bernoulli distributions with constant probability [61]. IDRL can be extended to these alternatives using approximate inference.

3.5 CONNECTION TO MULTI-ARMED BANDITS

So far, we motivated IDRL from information-theoretic considerations. However, there are close connections to related algorithms in multi-armed bandits (MAB; cf. Section 2.1) that can serve as additional motivation. Recent work successfully uses decision criteria based on information gain in various MAB problems [64]. However, our setting differs from standard MAB problems because we do not directly observe the quantity we are optimizing for, i.e., the return of a policy. In this section, we discuss two settings that are more closely related to our setting: the *linear partial monitoring* problem and *transductive linear bandits*.

3.5.1 Linear Partial Monitoring

Partial monitoring problems generalize the standard MAB problem to cases where the agent’s observations provide only indirect information about the reward [65]. For tabular MDPs, we can interpret our reward learning setting as a linear partial monitoring problem. Let \mathbf{r} be a vector of all rewards in a tabular MDP. We consider observations that are a linear function of the rewards $r(s_i) = \mathbf{c}^T \mathbf{r}$, and the optimization target is also a linear function of the reward vector $G(\pi) = \mathbf{f}(\pi)^T \mathbf{r}$. Kirschner *et al.* [63] analyze linear partial monitoring problems and propose criteria for selecting observations based on information gain. One criterion they propose to measure information gain, called *directed information gain*, is equivalent to the information gain criterion IDRL uses (App. B.2 in [63]). However, they consider cumulative regret minimization, and, therefore, their algorithm has to trade off the information gain of an observation with its expected regret. In our setting, minimizing cumulative regret would correspond to maximizing $\sum_{t=1}^T G(\bar{\pi}_t)$, where $\bar{\pi}_t$ is the policy that IDRL returns if it is stopped after t iterations. Instead, we evaluate the final policy and aim to maximize $G(\bar{\pi}_T)$. Consequently, our algorithm directly uses directed information gain as a selection criterion.

3.5.2 Transductive Linear Bandits

Our setting is a pure exploration problem: We evaluate only the performance of the final policy after a fixed budget of queries and not the intermediary policies. Fiez *et al.* [66] study pure exploration in *transductive linear bandits* where the goal is to maximize a linear reward function in a set \mathcal{Z} by making queries in a potentially different set \mathcal{X} . In this section, we show that for a tabular MDP, our problem is a special case of the transductive linear bandit problem. We can understand IDRL as an adaptive version of the RAGE algorithm introduced by Fiez *et al.* [66]. To see this connection, let us first define the transductive linear bandit problem.

Definition 3.5.1 (Fiez *et al.* [66]). A *transductive linear bandit problem* is defined by two sets $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Z} \subset \mathbb{R}^d$, where the goal is to find $\operatorname{argmax}_{\mathbf{z} \in \mathcal{Z}} \theta^T \mathbf{z}$ for some hidden parameter vector $\theta \in \mathbb{R}^d$. However, instead of observing this objective directly, the learning agent interacts with the bandit at each time-step by selecting an arm $\mathbf{x} \in \mathcal{X}$ to play and then observing $\theta^T \mathbf{x} + \eta$ where η is independent, zero-mean, sub-Gaussian noise. The agent’s goal is to find the maximum in \mathcal{Z} by making as few queries in \mathcal{X} as possible.

Proposition 3.5.1. For a tabular MDP, a fixed set of candidate policies Π_c and a set of linear reward queries \mathcal{Q}_c , our reward learning problem is a transductive linear bandit problem with $\mathcal{Z} = \{\mathbf{f}(\pi) | \pi \in \Pi_c\} \subset \mathbb{R}^{|\mathcal{S}|}$ and $\mathcal{X} = \{\mathbf{c}_i | i \in \{1, \dots, |\mathcal{Q}_c|\}\} \subset \mathbb{R}^{|\mathcal{S}|}$ a set of linear observations.

To see this, note that our goal is to maximize $G(\pi) = \mathbf{f}(\pi)^T \mathbf{r}$ and we query linear combinations of rewards in each round $\mathbf{c}_{i_t}^T \mathbf{r}$. Here i_t is the index of the query that the agent selects at time t , and \mathbf{c}_{i_t} is a vector of linear weights that defines query q_{i_t} .

To understand the connection between IDRL and the RAGE algorithm proposed by Fiez *et al.* [66], let us assume that the reward function is a linear function of some features of the state, and to use a linear kernel for the GP model, which is equivalent to Bayesian linear regression.

Let $\phi : \mathcal{S} \rightarrow \mathbb{R}^d$ be a feature function, and the true reward function $r(s) = \phi(s)^T \theta^*$. Similarly, we can define a feature vector for each query $q \in \mathcal{Q}_c$ and overload the notation $\phi(q) = \sum_{i=1}^N C_i \phi(s_i)$. Also, we can write the expected return of a policy as $G(\pi) = \mathbf{f}(\pi)_\phi^T \theta^*$ with $\mathbf{f}(\pi)_\phi = \mathbf{f}(\pi)^T \Phi$ and $\Phi = (\phi(s_1), \dots, \phi(s_{|\mathcal{S}|}))^T$.

The RAGE algorithm proceeds in multiple rounds. In each round, it selects arms using the allocation

$$\lambda_t^* = \operatorname{argmin}_{\lambda \in \Delta_{\mathcal{X}}} \max_{\pi_1, \pi_2 \in \widehat{\mathcal{Z}}_t} \|\mathbf{f}(\pi_1)_\phi - \mathbf{f}(\pi_2)_\phi\|_{A_\lambda}^2 \quad (3.3)$$

where $A_\lambda = \sum_{q_i \in \mathcal{Q}_c} \lambda_i \phi(q_i) \phi(q_i)^T$, and where $\Delta_{\mathcal{X}}$ is the probability simplex over candidate queries. This rule selects query q_i at round

t with probability $(\lambda_t^*)_i$. Additionally, RAGE keeps track of a set of plausibly optimal arms $\hat{\mathcal{Z}}_t$, i.e., plausibly optimal policies in our case. RAGE ensures that the suboptimality gap of arms in this set shrinks exponentially as the algorithm proceeds.

The following proposition provides an alternative notation for IDRL that shows a formal similarity to RAGE.

Proposition 3.5.2. Assume we estimate $\hat{\theta}$ with Bayesian linear regression with noise variance σ^2 , and prior $\hat{\theta} \sim \mathcal{N}(0, \alpha^{-1}I)$ after collecting data

$$\mathcal{D} = ((\boldsymbol{\phi}(q_{i_1}), y_{i_1}), \dots, (\boldsymbol{\phi}(q_{i_{t-1}}), y_{i_{t-1}})).$$

Also, assume an infinitely wide prior $\alpha^{-1} \rightarrow \infty$.

We can then write the maximization in the first step of IDRL as

$$\operatorname{argmax}_{\pi, \pi' \in \Pi_c} H(\widehat{G}(\pi) - \widehat{G}(\pi') | \mathcal{D}) = \operatorname{argmax}_{\pi, \pi' \in \Pi_c} \|\mathbf{f}(\pi)_\phi - \mathbf{f}(\pi')_\phi\|_{A_{\mathcal{D}}}^2$$

where $A_{\mathcal{D}} = \sum_{q_i \in \mathcal{Q}_c} N_i \boldsymbol{\phi}(q_i) \boldsymbol{\phi}(q_i)^T$.

Furthermore, for a given pair of policies, π_1 and π_2 , we can write the maximization in the second step of IDRL as

$$\begin{aligned} & \operatorname{argmax}_{q \in \mathcal{Q}_c} I(\widehat{G}(\pi_1) - \widehat{G}(\pi_2); (q, \hat{y}) | \mathcal{D}) \\ &= \operatorname{argmin}_{q \in \mathcal{Q}_c} \|\mathbf{f}(\pi_1)_\phi - \mathbf{f}(\pi_2)_\phi\|_{A_{\mathcal{D}, q}}^2 \end{aligned}$$

where $A_{\mathcal{D}, q} = \boldsymbol{\phi}(q) \boldsymbol{\phi}(q)^T + \sum_{q_i \in \mathcal{Q}_c} N_i \boldsymbol{\phi}(q_i) \boldsymbol{\phi}(q_i)^T$ and N_i is the number of times q_i occurs in \mathcal{D} .

Comparing this proposition with equation (3.3) shows a formal similarity between both algorithms. In particular, we can understand IDRL as a version of equation (3.3) that adapts to the data seen so far and selects the next observation that would minimize this objective. Instead of the matrix A_λ induced by the allocation λ , IDRL computes the variances using $A_{\mathcal{D}}$, i.e., based on data observed in the past, and using $A_{\mathcal{D}, q}$, i.e., evaluating the effect of an additional observation. Additionally, IDRL performs two separate optimizations,

which we can consider as an approximation to the min-max problem in equation (3.3), which would be infeasible to evaluate in our setting. Similar to RAGE, IDRL keeps track of a set of plausibly optimal policies. However, IDRL uses Thompson sampling, while RAGE uses suboptimality gaps to build this set.

3.6 IDRL FOR DEEP RL

GP models provide a convenient way to implement IDRL exactly. But can IDRL also be used if we can not model the reward function as a GP? Moreover, can we scale it to large environments on the scale of typical Deep RL applications?

To address these questions, we propose a second implementation of IDRL using a deep neural network (DNN) reward model. To scale IDRL to large Deep RL scenarios, we integrate it into a policy optimization algorithm, similar to Christiano *et al.* [12].² In our experiments, we focus on comparison queries, but extending the algorithm to other query types is straightforward.

REWARD MODEL. We use adaptive basis function regression with DNNs, similar to Snoek *et al.* [67]. Concretely, we represent the reward function as a function of observations o and actions a , using a DNN model $\mu_{\hat{r}}(o, a) = \theta^T f_{\phi}(o, a)$. We conceptually separate the model into a feature representation $f_{\phi}(o, a)$ parameterized by weights ϕ and a linear function θ . In practice, θ is the last layer of the DNN, and we train ϕ and θ jointly. We train the model from comparisons of short clips of the agent’s behavior using the Bradley-Terry model (see Section 2.4) and ℓ_2 -regularization. Because the Bradley-Terry model is invariant to shifting the reward function, we use DNN layers without biases. Additionally, we normalize the model’s output when using it to train policies.

To compute the IDRL objective, we need a Bayesian posterior. We treat the learned representation $f_{\phi}(o, a)$ as a basis function and the

² We compare our setup to Christiano *et al.* [12] in more detail in Appendix A.2.2.

final layer of the DNN as a *maximum a posteriori* (MAP) estimate of the parameters of a Bayesian logistic regression model. Finally, we approximate the full posterior using a Laplace approximation:

$$p(\theta|\mathcal{D}, \hat{\phi}) \approx \mathcal{N}(\hat{\theta}, H^{-1})$$

$$H = [\nabla^2 \log p(\theta|\mathcal{D}, \hat{\phi})] \Big|_{\theta=\hat{\theta}}$$

where H is the Hessian of the log-likelihood at point $\hat{\theta}$. The Laplace approximation is a basic technique for approximate inference; however, it is convenient because it approximates the posterior as a Gaussian distribution. Hence, we can easily compute this distribution’s entropy and information gain, similar to a GP model.

QUERY SELECTION. Because of the Laplace approximation, the posterior distribution of $\hat{r}(s)|\mathcal{D}$ is Gaussian, and we can compute equations (3.1) and (3.2) the same way we did for a GP reward model.

QUERY TYPES. Similar to Christiano *et al.* [12], we consider queries $q_i = (\sigma_i^1, \sigma_i^2)$ that compare two segments of trajectories σ_i^1 and σ_i^2 , where the user responds with their preference $y_i \in \{-1, 1\}$.

CANDIDATE POLICIES. In large environments, training new policies from scratch during the Thompson sampling step is infeasible. To avoid this, we maintain a fixed set of policies that we update regularly instead of training new policies from scratch whenever we receive new samples.

CANDIDATE QUERIES. We generate candidate queries by rolling out the current policy optimized for the mean estimate of the reward model, as well as the candidate policies and uniformly sampling pairs of segments from the resulting trajectories.

FULL ALGORITHM. We use the soft actor-critic algorithm (SAC; [68]) to train a policy for the current reward model and the candidate

policies. Similar to Christiano *et al.* [12], the agent queries comparisons following a fixed schedule in which the number of samples is proportional to $\frac{1}{T}$, where T is the number of policy training steps, i.e., we provide more samples early during training and less later on. Algorithm 2 shows the full pseudocode for our Deep RL implementation of IDRL; for details on hyperparameters, see Appendix A.2.2.

3.7 EXPERIMENTS

We empirically test IDRL in several environments, ranging from Gridworlds to complex continuous control tasks, and for several query types, including numerical evaluations and comparisons of trajectories. Our evaluation covers most scenarios from prior work and shows that IDRL attains comparable or superior performance to methods designed for specific scenarios.

In all experiments, we simulate expert feedback based on an underlying true reward function unknown to the agent. We usually evaluate the *regret* of a policy π trained using the reward model, i.e., $G(\pi^*) - G(\pi)$ for an optimal policy π^* . If we do not know π^* , we approximate it with a policy trained on the true reward function.

We first validate that GP-based IDRL improves sample efficiency in simple environments for numerical and comparison queries (Section 3.7.2). Next, we consider the most common setup in the literature: learning from comparisons of trajectories. We compare GP-based IDRL against alternative approaches in a driving simulator, proposed in prior work (Section 3.7.3). Then, we study another natural feedback type: ratings of clips of the agent’s behavior. In this setting, we demonstrate how GP-based IDRL can be scaled up to bigger environments in the MuJoCo simulator (Section 3.7.4). Finally, we further demonstrate scalability by considering the Deep RL implementation of IDRL to learn standard MuJoCo tasks from comparisons of clips of trajectories, similar to Christiano *et al.* [12] (Section 3.7.5).

We choose our setup for each environment to be close to prior work to promote a fair comparison. Consequently, our choice of

Algorithm 2 *Information Directed Reward Learning (IDRL)* using neural networks. The main training loop alternates between updating the policy $\bar{\pi}^*$, querying new samples, and updating the candidate policies. The algorithm selects queries using the IDRL objective, where line 13 corresponds to equation (3.1) and line 19 corresponds to equation (3.2). Line 4 updates the feature representation and estimates the MAP of the posterior, and line 5 approximates the posterior using a Laplace approximation.

```

1: Initialize policy  $\bar{\pi}^*$ , candidates  $\Pi_c \leftarrow \{\pi_1^c, \dots, \pi_n^c\}$ ,  $\mathcal{D} \leftarrow \{\}$ 
2: Initialize DNN reward model  $\mu_{\hat{r}}(o, a) = \theta^T f_\phi(o, a)$ 
3: while not done do
4:   Update DNN parameters,  $\phi$  and  $\theta$ , on  $\mathcal{D}$ 
5:    $H \leftarrow [\nabla^2 \log p(\theta | \mathcal{D}, \phi)] \Big|_{\theta}$ 
6:   Train policy  $\bar{\pi}^*$  on reward  $r_i(o, a) = \theta^T f_\phi(o, a)$  using SAC
7:   if update candidate policies then
8:     Sample  $\omega_1, \dots, \omega_n$  from  $\mathcal{N}(\theta, H^{-1})$ 
9:     for  $i \in \{1, \dots, n\}$  do
10:      Train  $\pi_i^c$  on reward  $r_i(o, a) = \omega_i^T f_\phi(o, a)$  using SAC
11:      Estimate occupancy measure  $\mathbf{f}(\pi_i^c)$  using rollouts
12:   if query samples then
13:      $\pi_1, \pi_2 \in \operatorname{argmax}_{\pi, \pi' \in \Pi_c} (\mathbf{f}(\pi) - \mathbf{f}(\pi'))^T H^{-1} (\mathbf{f}(\pi) - \mathbf{f}(\pi'))$ 
14:      $\mathbf{v}(\pi_1, \pi_2) \leftarrow \mathbf{f}(\pi_1) - \mathbf{f}(\pi_2)$ 
15:     Roll out policy  $\bar{\pi}^*$  and collect candidate queries  $\mathcal{Q}_c$ 
16:     for  $q_i \in \mathcal{Q}_c$  do
17:       Compute Hessian with the expected response to  $q_i$ :
18:        $H_{q_i} \leftarrow [\nabla^2 \log p(\theta | \mathcal{D} \cup \{(q_i, \hat{y}_i)\}, \phi)] \Big|_{\theta}$ 
19:        $u(q_i, \mathcal{D}) \leftarrow -\mathbf{v}(\pi_1, \pi_2)^T H_{q_i}^{-1} \mathbf{v}(\pi_1, \pi_2)$ 
20:       Sort queries by  $u(q_i, \mathcal{D})$  and select top  $k$ :  $\{q_1, \dots, q_k\}$ 
21:       Make queries  $\{q_1, \dots, q_k\}$  and observe  $\{y_1, \dots, y_k\}$ 
22:        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(q_1, y_1), \dots, (q_k, y_k)\}$ 
23: return  $\bar{\pi}^*$ 

```

RL solver and query types differ between environments. As a side effect, this highlights IDRL’s generality. Appendix A.2 provides some additional details on our implementation and experiments.

3.7.1 Baselines

We consider five baselines:

- *Uniform sampling* selects queries from \mathcal{Q}_c with equal probability.
- *Information gain on the reward* (IGR) selects queries that maximize information gain about the reward $I((q, \hat{y}); \hat{r}|\mathcal{D})$. For a GP model, this is equivalent to maximizing $\text{Var}[\hat{y}|\mathcal{D}, q]$. Bıyık *et al.* [59] use IGR to learn rewards from comparisons of trajectories; however, it can be extended to other query types.
- *Expected improvement on the reward* (EIR) maximizes the improvement in the value of a query compared to the best observation so far, in *expectation*, and is a common acquisition function in BO [31]. EIR can not be applied to comparison queries.
- *Expected policy divergence* (EPD) is an active reward learning method introduced by Daniel *et al.* [52], which makes queries that maximally change the current policy. Since EPD updates the policy for each potential observation, it is prohibitively expensive for large \mathcal{Q}_c . While EPD was introduced to query the return of trajectories, we extend it to other query types (cf. Appendix A.2).
- *Maximum regret* (MR) is an acquisition function proposed by Wilde *et al.* [61]. It assumes access to a set of candidate reward functions and corresponding optimal policies. MR compares policies that perform well according to one reward function but poorly according to a different one. It can only be used with comparisons of full trajectories.

We also tested *expected volume removal* (EVR; [22]) for comparison queries; however, we found it to get stuck often, which confirms the

findings of Bıyık *et al.* [59]. Note that IGR and EIR reduce uncertainty uniformly over the state space, while EPD and MR consider the environment dynamics.

3.7.2 Can IDRL Improve Sample Efficiency?

We first validate our hypothesis that IDRL improves sample efficiency in small environments. We study a set of *Gridworlds* similar to Figure 3.1, and two toy environments, we call *Chain* and *Junction*, that isolate specific reasons we expect IDRL to perform well.

TOY ENVIRONMENTS. Figure 3.2 shows the *Chain* and *Junction* MDPs. Importantly, in both environments, there are states in which the agent’s actions do not change the transitions. The rewards of these transitions are not as decision-relevant as the rewards of transitions the agent can influence. Hence, we expect IDRL to focus on the latter.

GRIDWORLDS. We consider 10×10 *Gridworlds* with randomly placed walls and objects with different rewards. The agent has to find the object with the largest reward, similar to Figure 3.1.

QUERY TYPES. We consider queries about the reward of individual states, i.e., $q_i = s_i \in \mathcal{S}$ and $y_i = r(s_i)$, and comparison queries with $q_i = (s_{i1}, s_{i2})$ and $y_i \in \{-1, 1\}$. The candidate queries \mathcal{Q}_c either consist of all states or all pairs of states. We use GP-based IDRL, with a kernel that encodes distance between states in the *Chain* and *Junction* and the similarity of different squares in the *Gridworld* which objects are the same and which are different.

RESULTS. Figure 3.3 shows the regret of a policy trained on the reward model after certain numbers of queries. IDRL finds better policies than the baselines with a limited number of queries because it focuses on regions of the state space relevant for finding the optimal policy. As shown in Figure 3.1, this improves sample efficiency over

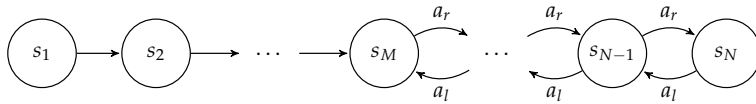
methods that uniformly reduce uncertainty, such as IGR and EIR. IDRL also outperforms EPD because EPD’s goal of selecting queries that maximally change the current policy differs from the goal of finding an optimal policy. We investigate EPD’s specific failure modes in Appendix A.3.2.

3.7.3 *Can IDRL Learn From Comparisons?*

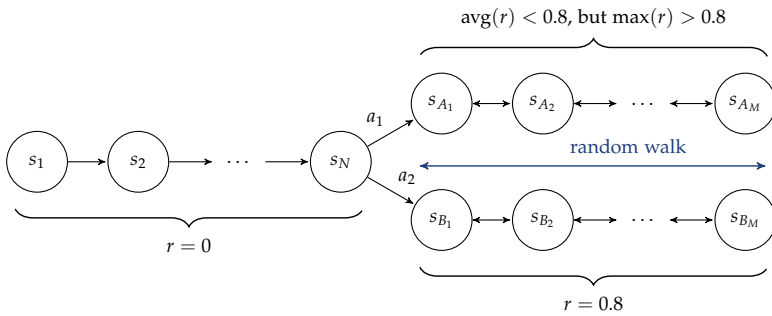
Most prior work studies reward learning from comparisons of trajectories. To evaluate IDRL in this setting, we consider the 2-dimensional, continuous *Driver* environment by Sadigh *et al.* [22].

SETUP. In the *Driver* environment, the agent controls a car on a highway with another car driving on a fixed trajectory (cf. Figure 3.4a). We randomly sample an underlying (linear) reward function for each experiment to describe the desired driving behavior. We use code by Sadigh *et al.* [22] to simulate and solve the environment but adapt it to our setting. In contrast to Sadigh *et al.* [22], we do not synthesize queries. Instead, we sample a fixed set of 200 reward functions from a Gaussian prior distribution. We then optimize a policy for each reward function and, similarly to Wilde *et al.* [61], consider all pairs of policies as potential queries. Moreover, we assume a linear observation model, whereas Sadigh *et al.* [22] and Wilde *et al.* [61] choose different non-linear observation models. Each experiment runs for less than 24 hours on a single CPU.

RESULTS. Figure 3.4a shows the regret curves for the learned policy and the cosine similarity for the learned reward function weights. IDRL outperforms the baselines and finds a better policy with fewer queries. However, the difference to pure information gain is small in this simple environment.



(a) Chain



(b) Junction

FIGURE 3.2: Illustration of the *Chain* and *Junction* MDPs. In the *Chain* MDP (a), the agent moves right with probability 1 from state s_1 to s_M , and the agent can move deterministically left and right from state s_M to state s_N with $N > M$. The reward function is sampled from the GP prior with a square-exponential kernel. We choose $M = 10$ and $N = 20$. In the *Junction* MDP (b), the agent moves right with probability 1 from state s_1 to s_N . In s_N , the agent can take the upper or lower path, denoted with A and B , respectively. The agent moves left or right along both paths with probability 0.5. In the lower path, the reward of all states is 0.8. In the upper path, the highest reward is greater than 0.8, but the average reward is less than 0.8. We choose $N = 15$ and $M = 5$. For both environments, the initial state distribution is uniform over the state space.

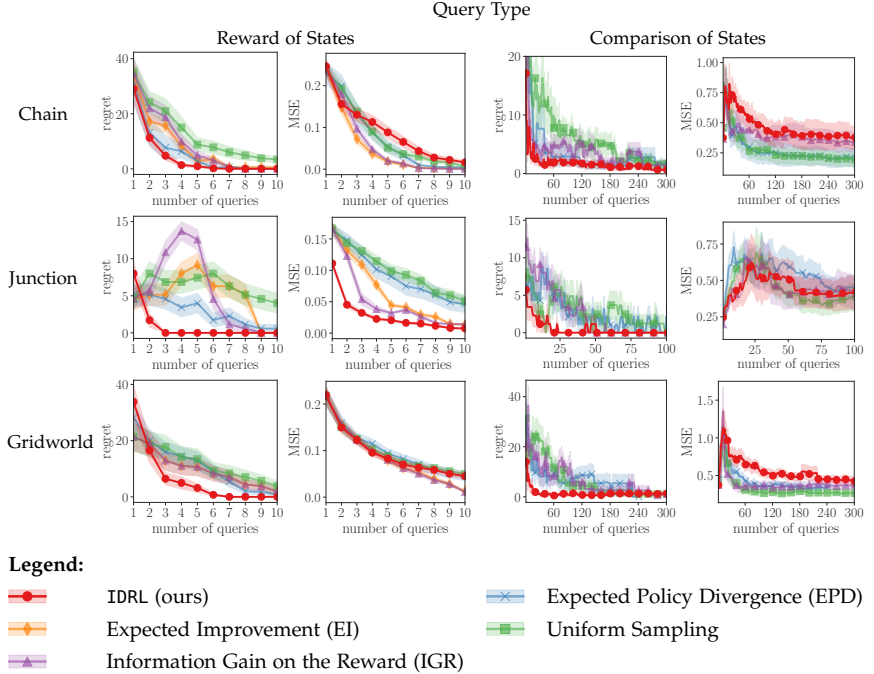


FIGURE 3.3: Learning curves for the *Chain*, *Junction*, and *Gridworld* environments for two query types: the reward of individual states and comparisons of states. For each setting, one plot shows the regret of a policy trained using the reward model, and a second plot shows the mean squared error (MSE) of the reward model over the whole state space. The plots show mean and standard error across 30 random seeds. Across all environments, IDRL learns a better policy with fewer queries. However, the MSE measured on the entire state space is usually worse because IDRL focuses on regions of the state space that are informative about the optimal policy.

Legend:

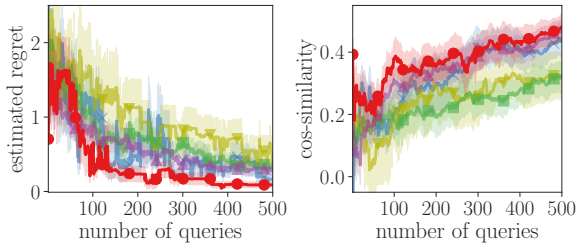
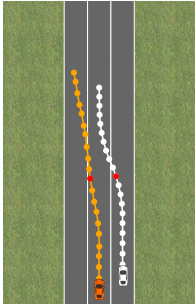
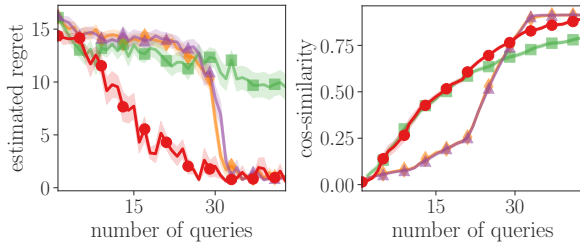
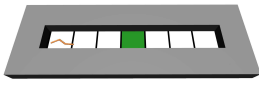
(a) *Driver* (comparison of trajectories)(b) *Swimmer-Corridor* (evaluation of trajectory clips)

FIGURE 3.4: Results in the (a) *Driver* and (b) *Swimmer-Corridor* environments (shown on the left). We show the regret of a policy trained on the reward model compared to a policy trained on the true reward function as a function of the number of queries (middle plot). We report the cosine similarity between the learned and the true reward function (right plot). The plots show mean and standard error over 30 random seeds. IDRL finds significantly better policies while not necessarily learning an overall more accurate model of the reward function. Appendix A.3.2 contains a similar plot for *Ant-Corridor*.

	<i>Swimmer-Corridor</i>	<i>Ant-Corridor</i>
Uniform Sampling	11.8 ± 0.9	15 ± 1
IGR	13.3 ± 0.6	17.2 ± 0.5
EIR	12.0 ± 0.8	17.5 ± 0.8
IDRL (20 updates)	2.4 ± 0.8	2.2 ± 0.8
IDRL (4 updates)	2.8 ± 0.7	5 ± 1
IDRL (2 updates)	5.1 ± 0.6	8 ± 1
IDRL (1 update)	7.6 ± 0.8	12 ± 1

TABLE 3.2: Results comparing IDRL for different update frequencies of the candidate policies in the *Corridor* environments. The table shows the estimated regret of a policy trained using 20 queries about the reward function.

3.7.4 Does IDRL Scale to Bigger Environments?

To demonstrate that GP-based IDRL scales to larger environments, we use the MuJoCo simulator [69], which provides challenging environments commonly used as benchmarks for RL. However, its standard locomotion tasks are very easy to learn for a GP model because the reward is directly proportional to the agent’s velocity in the x-direction. Instead, we propose a task where the reward function is harder to learn.

SETUP. In our *Corridor* environments (Figure 3.4b), a robot (*Swimmer*, or *Ant*) has to move forward and stop at a goal position. The simulated expert provides ratings for trajectory clips according to a reward function proportional to the velocity in the direction of the goal. This reward function is linear in a set of state features, as described in Appendix A.3.1. We use *augmented random search* [70] as an RL algorithm. For the *Swimmer-Corridor*, we learn a linear policy; for the *Ant-Corridor*, we learn a hierarchical policy on top of pre-

trained policies moving in four different directions. We use a fixed, noisy exploration policy that moves along the corridor to generate candidate queries. Unfortunately, EPD is too expensive to evaluate in this environment, and MR is not suited to numerical evaluation queries, so that our baselines are limited.

RESULTS. Figure 3.4b shows that IDRL needs significantly fewer queries to find a good policy than any of the baselines. IDRL adapts its queries to the policies that the current reward model induces: it initially samples clips in which the robot moves close to its starting position and shifts its focus to other regions as the reward model improves, and the learned policy starts to move. In contrast, the baselines make queries in the whole reachable space similarly often and, therefore, waste queries in regions that are not directly relevant to improving the policy.

The computationally most expensive part of this implementation of IDRL is updating the candidate policies in each iteration. Updating them less often reduces the computational cost at the expense of potentially reducing the sample efficiency. Table 3.2 studies this trade-off and shows that IDRL outperforms the baselines even when the policies are updated only once at the beginning of training. In this extreme case, we reduce IDRL’s total runtime from about 40 hours to about 20 hours in *Swimmer-Corridor*, and from about 40 hours to about 10 hours in *Ant-Corridor*. The benefits are larger when solving the RL problem is more expensive. Nonetheless, the baseline algorithms are still faster and run for only 2 – 3 hours. This is because they do not require the additional inference steps necessary to optimize Equation (3.2). These results indicate that IDRL using full Thompson sampling to generate candidate policies can trade off computational cost and sample efficiency, which allows it to be applied to large environments.

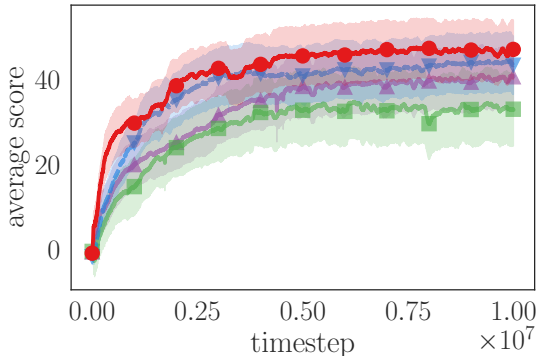


FIGURE 3.5: The normalized score of policies learned from 1400 (synthetic) comparisons of clips of the agent’s behavior, averaged over all MuJoCo environments (higher is better). We show the mean and standard error of the score averaged over 5 random seeds per environment. The plot compares IDRL (—●—) to IGR (—▲—) and uniform sampling (—■—), as well as an ablation of IDRL that does not use the candidate policies to generate additional candidate queries (—▼—). EPD is too expensive, and MR is unsuitable for queries consisting of clips of trajectories.

3.7.5 Does IDRL Scale to Deep RL?

Finally, we consider the Deep RL implementation of IDRL from Section 3.6, using the Soft Actor-Critic algorithm (SAC; [68]). We test it on standard MuJoCo locomotion tasks, which are harder to learn with a DNN than a GP model because the former encodes less prior information.

SETUP. We consider a suite of standard tasks in MuJoCo implemented in OpenAI Gym [71]: *HalfCheetah-v3*, *Walker2d-v3*, *Hopper-v3*, *Ant-v3*, *Swimmer-v3*, *InvertedPendulum-v2*, *InvertedDoublePendulum-v2*, *Reacher-v2*. Similar to Christiano *et al.* [12], we modify some environments to remove the termination conditions. Our environments differ slightly from Christiano *et al.* [12]; see Appendix A.2.2 for details. Our evaluation metric is a normalized score averaged over

all environments. A score of 0 corresponds to a random policy, and a score of 100 is the performance of a policy trained on the true reward function. We provide results for the individual environments in Appendix A.3.2. Since IDRL tracks the candidate policies, it generates the candidate queries rolling out the currently optimal policy *and* the candidate policies. However, the baselines cannot access the candidate policies and therefore consider a smaller set of potential queries. For a fair comparison, we perform an ablation where IDRL does not consider the candidate policies to generate candidate queries. Since IDRL maintains 3 (additional) candidate policies, it is roughly 4 times slower (about 80 hours on a single GPU) than the baselines (about 20 hours on a single GPU).

RESULTS. Figure 3.5 shows that, on average, IDRL learns good policies significantly faster than the baselines. The individual results in each environment (in Appendix A.3.2) are more nuanced. IDRL outperforms the baselines in some environments (e.g., *Hopper-v3*), performs comparable in other environments (e.g., *Walker2d-v3*), and performs worse than uniform sampling in a few environments (e.g., *HalfCheetah-v3*). Also, while mostly using the candidate policy rollouts improves the performance of IDRL, this is not always the case (e.g., in *Swimmer-v3* the ablation performs better). This indicates that much of the variance might be caused by the candidate queries, which we could improve by using other exploration strategies than rolling out the candidate policies. Crucially, these experiments demonstrate that IDRL is scalable to high-dimensional, complex tasks and improves sample efficiency over existing methods.

3.8 CONCLUSION

In this chapter, we studied the problem of actively learning reward function models using as few expert queries as possible. We introduced *Information Directed Reward Learning* (IDRL), a novel information-theoretic algorithm that focuses on learning a good policy rather than

attaining a low approximation error of the reward. We show that it needs significantly fewer queries than prior methods and that it scales to complex environments.

The main practical limitation of IDRL is its computational cost. We demonstrated how to scale IDRL to complex environments, increasing the runtime by only a constant factor. While IDRL is still more demanding than simpler algorithms, it is preferable when better sample efficiency is more important than low computational cost. Our problem setup also has conceptual limitations. We assume that interactions with the environment are cheap, which is not always the case. We partially address this problem in Chapter 4 where we focus on exploration. Moreover, we assume that the goal of RL is to learn a good policy in a single environment, which does not consider generalizing to other environments. To address this, future versions of IDRL could aim to learn a reward model that leads to good policies over a *distribution* of environments instead of a single environment.

While IDRL provides a principled method for choosing which reward queries to make from a fixed set of candidate queries \mathcal{Q}_c , it does not address choosing the set of candidate queries. Commonly the candidate queries will be states or trajectories obtained by exploring the environment. The next chapter studies actively exploring the environment to learn a reward model efficiently.

ACTIVE EXPLORATION FOR INVERSE REINFORCEMENT LEARNING

In this chapter, we turn to study the problem of efficiently exploring the environment to learn a reward model. Instead of studying preference learning with a general feedback type, in this chapter, we focus on Inverse Reinforcement Learning (IRL), i.e., observing actions from an expert policy. This restriction allows us to provide a more formal problem setup and to develop an algorithm with explicit sample complexity guarantees. Nevertheless, the key principles are the same as in Chapter 3, and the analysis in the present chapter could also be extended to other types of feedback than expert demonstrations.

Most existing and well-studied IRL algorithms assume that the transition model, and in some cases, the expert’s policy, are *known*. In many real-world applications, this is not true, and the agent needs to estimate the transition dynamics and the expert policy from samples. Figure 4.1 shows an illustrative example where the agent can choose between different paths that have different properties, e.g., walking speeds, and lead to different goals.

Hence, we consider IRL with unknown transition dynamics and expert policy and focus on exploring the environment in order to recover the expert’s reward function efficiently. To the best of our knowledge, we present the first sample complexity guarantees for the active IRL problem without access to a generative model. In particular, this chapter presents the following key contributions:

- We propose the active IRL problem in a finite-horizon, undiscounted Markov Decision Process (MDP) and characterize necessary and sufficient conditions for solving it (Section 4.2.1).
- We analyze how the estimation errors of the transition model and the expert policy contribute to the estimation error of

the reward function and a policy that optimizes the recovered reward function (Section 4.2.2).

- We propose Active Exploration for Inverse Reinforcement Learning (AceIRL), a novel algorithm that actively explores the environment and the expert policy to infer a reward function. AceIRL constructs exploration policies based on the estimation error of the recovered reward function (Section 4.4).
- We consider two different exploration strategies for AceIRL. The first, more straightforward strategy provides a sample complexity similar to the algorithm proposed by Metelli *et al.* [72], which has access to a generative model (Section 4.4.1). The second strategy takes the expected reduction in uncertainty into account (Section 4.4.2), which yields a tighter, problem-dependent sample complexity bound (Section 4.4.3).
- We evaluate AceIRL empirically in simulated environments and demonstrate that it performs significantly better than more naive exploration strategies (Section 6.3).

The contents of this chapter are based on:

D. Lindner, A. Krause, and G. Ramponi, “Active exploration for inverse reinforcement learning”, in *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

4.1 RELATED WORK

Most IRL algorithms assume that the underlying transition model is known [47, 73–75]. However, the transition model usually needs to be estimated from samples, which induces an error in the recovered reward function that most papers do not study. Metelli *et al.* [72] analyze this error and the sample complexity of IRL in a tabular setting with a generative model. They propose an algorithm called TRAVEL designed to transfer the learned reward function to a fully

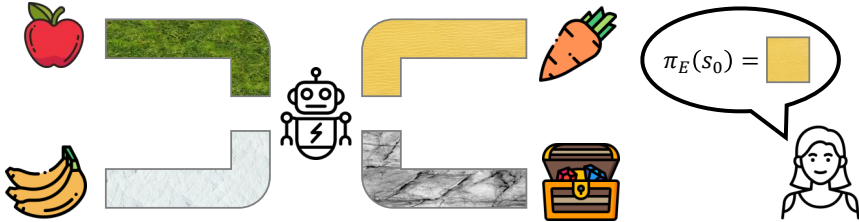


FIGURE 4.1: Illustration of active IRL. The agent can choose between four paths that lead to different objects. It can get action recommendations from an expert but does not know about the properties of the different paths (the transition dynamics) or the value of different items (the reward function). The agent’s goal is to infer which reward functions explain the expert’s recommendations. Only observing the expert actions is not enough to do that. Instead, the agent has to explore the environment and learn about the dynamics. The human might prefer to find the treasure over the carrot but still recommend the yellow path because the treasure is very difficult to reach. To explore efficiently, the agent has to combine its uncertainty about the expert policy with its uncertainty about the environment to choose where to explore. AceIRL implements an exploration strategy that aims to infer which reward functions are consistent with the expert’s recommendations as quickly as possible. We present experiments on a version of this environment in Section 6.3.

known target environment. TRAVEL selects which queries to make to the expert, assuming access to a generative model of the source environment and a fully known target environment. Dexter *et al.* [76] provides a similar analysis in continuous state spaces and discrete action spaces, but they still require a generative model of the environment. Their algorithm also requires a generative model of the environment, but in contrast to Metelli *et al.* [72] it does not consider transferring the reward function to a new environment. In contrast to both works, we *do not* assume access to a generative model and thus need to tackle the exploration problem in IRL.

Some prior work studies active learning algorithms for IRL in a Bayesian framework but without providing theoretical guarantees. Lopes *et al.* [77] propose an active learning algorithm for IRL that estimates a posterior distribution over reward functions from demonstrations, requiring a prior distribution and full knowledge of the environment dynamics. Relatedly, Cohn *et al.* [78] consider a Bayesian IRL setting with a semi-autonomous agent that asks an expert for advice if it is uncertain about the reward. Cohn *et al.* [78] consider an agent who acts autonomously when confident and asks a human expert for advice otherwise. They consider a Bayesian IRL setting to define these acquisition functions and assume full knowledge of the environment dynamics, similar to Lopes *et al.* [77]. Brown *et al.* [79] empirically study active IRL in several safety-critical environments, selecting queries using value at risk. Kulick *et al.* [57] consider active learning for a robotic manipulation task, asking a human expert for advice in situations with the highest predictive uncertainty. Similarly, Losey and O'Malley [80] propose a method to learn uncertainty estimates from human corrections in robotics. All these papers assume a Bayesian framework and do not provide theoretical guarantees. In contrast, our setup does not require a prior over reward functions, and we provide sample complexity guarantees for our algorithm.

A separate line of work studies sample complexity in *imitation learning* where the goal is to imitate an expert policy rather than infer a reward function [81, 82]. In particular, Abbeel and Ng [83] also focus on exploration and propose to use the expert policy to explore

relevant regions, whereas Shani *et al.* [84] use an upper-confidence approach to exploration. Our setting is different because we focus on IRL instead of imitation learning, and we aim to explore to infer a reward function as efficiently as possible.

4.2 ACTIVE LEARNING FOR IRL

In this section, we first introduce the active Inverse Reinforcement Learning problem with and without a generative model (Section 4.2.1). Then, we define the feasible reward set for finite-horizon MDPs (Section 4.2.2) and characterize the error propagation on the reward function and the value function (Section 4.2.3), extending results by Metelli *et al.* [72] to the finite-horizon setting.

4.2.1 Problem Definition

Our goal is to design an exploration strategy to construct a dataset of demonstrations \mathcal{D} such that an arbitrary IRL algorithm can recover a *good* reward function from it. To be agnostic to the choice of IRL algorithm, we consider the set of all feasible reward functions for a specific expert policy. Formally, we consider IRL problems (\mathcal{M}, π^E) consisting of an MDP \mathcal{M} and an expert policy π^E , and we define the feasible reward set as follows.

Definition 4.2.1 (Feasible Reward Set). A reward function r is *feasible* for an IRL problem (\mathcal{M}, π^E) , if and only if the expert policy π^E is optimal in $\mathcal{M} \cup r$. We call the set of all feasible reward functions $\mathcal{R}_{\mathcal{M} \cup \pi^E}$ the *feasible reward set*. If we estimate the transition model and expert policy from samples, we refer to the *recovered feasible set* $\mathcal{R}_{\hat{\mathcal{M}} \cup \hat{\pi}^E}$ in contrast to the *exact feasible set* $\mathcal{R}_{\mathcal{M} \cup \pi^E}$.

Now, we can formalize the goal of Active IRL as finding an exploration strategy that satisfies the following PAC optimality criterion.

Definition 4.2.2 (Optimality Criterion). Let $\mathcal{R}_{\mathfrak{B}}$ be the exact feasible set and $\widehat{\mathcal{R}}_{\mathfrak{B}}$ be the feasible set recovered after observing $n \geq 0$ samples collected from \mathcal{M} and π^E . We say that an algorithm for Active IRL is (ϵ, δ, n) -correct if after n iterations with probability at least $1 - \delta$ it holds that:

$$\inf_{\hat{r} \in \widehat{\mathcal{R}}_{\mathfrak{B}}} \sup_{\hat{\pi}^* \in \Pi_{\widehat{\mathcal{M}} \cup \hat{r}}^*} \max_a \left| Q_{\mathcal{M} \cup r}^{\pi^*, 0}(s_0, a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, 0}(s_0, a) \right| \leq \epsilon \quad \text{for each } r \in \mathcal{R}_{\mathfrak{B}},$$

$$\inf_{r \in \mathcal{R}_{\mathfrak{B}}} \sup_{\pi^* \in \Pi_{\mathcal{M} \cup r}^*} \max_a \left| Q_{\mathcal{M} \cup r}^{\pi^*, 0}(s_0, a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, 0}(s_0, a) \right| \leq \epsilon \quad \text{for each } \hat{r} \in \widehat{\mathcal{R}}_{\mathfrak{B}},$$

where π^* is an optimal policy in $\mathcal{M} \cup r$ and $\hat{\pi}^*$ is an optimal policy in $\widehat{\mathcal{M}} \cup \hat{r}$.

The first condition states that for each reward in the exact feasible set, the best reward we could estimate in the recovered feasible set has a low error everywhere. This condition is a type of “recall”: every possible true reward function must be captured by the recovered feasible set. The second condition ensures that there is a possible true reward function with a low error for every possible recovered reward function. This avoids an unnecessarily large recovered feasible set. This condition is a type of “precision”: if we recover a reward function, it has to be close to a possible true reward function. Note that Metelli *et al.* [72] consider a similar optimality criterion in their Definition 5.1. However, they consider a known target environment; hence, our Definition 4.2.2 is a stronger requirement.

4.2.2 Feasible Rewards in Finite-Horizon MDPs

Ng and Russell [46] characterize the feasible reward set implicitly in the infinite-horizon setting, whereas Metelli *et al.* [72] characterize it explicitly. In this section, we provide similar results for a finite horizon. Appendix B.1 contains full proofs of all results.

Lemma 4.2.1 (Feasible Reward Set Implicit). A reward function r is feasible if and only if for all s, a, h it holds that: $A_{\mathcal{M} \cup r}^{\pi, h}(s, a) = 0$

if $\pi_h^E(a|s) \geq 0$ and $A_{\mathcal{M} \cup r}^{\pi, h}(s, a) \leq 0$ if $\pi_h^E(a|s) = 0$. Moreover, if the second inequality is strict, π^E is uniquely optimal, i.e., $\Pi_{\mathcal{M} \cup r}^* = \{\pi^E\}$.

Lemma 4.2.2 (Feasible Reward Set Explicit). A reward function r is feasible if and only if there exists an $\{A_h \in \mathbb{R}_{\geq 0}^{|\mathcal{S}| \times |\mathcal{A}|}\}_{h \in [H]}$ and $\{V_h \in \mathbb{R}^{|\mathcal{S}|}\}_{h \in [H]}$ such that for all s, a, h it holds that:

$$r_h(s, a) = -A_h(s, a) \mathbb{1}_{\{\pi_h^E(a|s)=0\}} + V_h(s) + \sum_{s'} P(s'|s, a) V_{h+1}(s')$$

Here, the **first term** ensures there is an advantage function for π^E , and it is 0 for actions the expert takes and $A_h(s, a)$ for actions the expert does not take. The **second term** corresponds to reward-shaping by the value function.

4.2.3 Error Propagation

Next, we study the error propagation of estimating the transition model P with \widehat{P} and the expert policy π^E with $\widehat{\pi}^E$. In particular, we bound the estimation error on the reward as a function of the estimation errors of \widehat{P} and $\widehat{\pi}^E$, extending a result by Metelli *et al.* [72] to the finite-horizon setting.

Theorem 4.2.1 (Error Propagation). Let (\mathcal{M}, π^E) and $(\widehat{\mathcal{M}}, \widehat{\pi}^E)$ be two IRL problems. Then, for any $r \in \mathcal{R}_{(\mathcal{M}, \pi^E)}$ there exists $\widehat{r} \in \widehat{\mathcal{R}}_{(\widehat{\mathcal{M}}, \widehat{\pi}^E)}$ such that:

$$\begin{aligned} |r_h(s, a) - \widehat{r}_h(s, a)| &\leq A_h(s, a) |\pi_h^E(a|s) - \widehat{\pi}_h^E(a|s)| \\ &\quad + \sum_{s'} V_{h+1}(s') |P(s'|s, a) - \widehat{P}(s'|s, a)| \end{aligned}$$

and we can bound $V_h \leq (H - h)r_{\max}$ and $A_h \leq (H - h)r_{\max}$.

It provides a bound on the distance between each reward function in the real feasible reward set $\mathcal{R}_{\mathfrak{B}}$ to the closest one in the estimated one $\widehat{\mathcal{R}}_{\mathfrak{B}}$. The error depends on the two estimated components and

this is reflected as the sum of two terms, one depending on the estimation of the expert policy and the other of the transition model.

In IRL, we cannot hope to recover the expert’s reward function perfectly. Instead, we aim to estimate a reward function that leads to an optimal policy with performance close to the expert’s policy under the (unknown) true reward function. For example, suppose a specific state s is difficult to reach in the environment. In that case, the error on the reward function $r(s, \cdot)$ will not impact the performance of the induced policy much. Formally, we are interested in studying the error propagation to the optimal value function. The following lemma will be crucial for analyzing this.

Lemma 4.2.3. Let \mathcal{M} be an MDP $\setminus R$, r, \hat{r} two reward functions with optimal policies $\pi^*, \hat{\pi}^*$. Then,

$$\begin{aligned} & Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, h}(s, a) \\ & \leq \sum_{h'=h}^H \sum_{s', a'} \left(\mu_{\mathcal{M}, \pi^*}^{h, h'}(s', a' | s, a) - \mu_{\mathcal{M}, \hat{\pi}^*}^{h, h'}(s', a' | s, a) \right) (r_{h'}(s', a') - \hat{r}_{h'}(s', a')) \end{aligned}$$

By combining this with Theorem 4.2.1, we can decompose the error in the value function and Q-function into the error in estimating the transition model and the error in estimating the expert policy.

4.3 UNIFORM SAMPLING WITH A GENERATIVE MODEL

As a warmup, let us first study the sample complexity of a simple *uniform sampling* strategy with access to the generative model of \mathcal{M} . We assume we can query a generative model about a state-action pair (s, a) to receive a next state $s' \sim P(\cdot | s, a)$ and an expert action $a_E \sim \pi^E(\cdot | s)$. This setting allows us to introduce key ideas and serves as a baseline to compare later results to. We adapt the infinite-horizon results by Metelli *et al.* [72] to the finite-horizon setting and our stronger PAC requirement in Definition 4.2.2. We first discuss how we can estimate the transition model and the policy (Section 4.3.1)

before analyzing the sample complexity of the uniform sampling strategy (Section 4.3.2).

4.3.1 Estimating Transition Model and Expert Policy

In each iteration k , let $n_k^h(s, a, s')$ be the number of times we observed the transitions (s, a, s') at time h up to iteration k . Also, we define $n_k^h(s, a) = \sum_{s'} n_k^h(s, a, s')$, and $n_k^h(s) = \sum_a n_k^h(s, a)$. Then we can estimate the transition model and expert policy by

$$\hat{P}_k(s'|s, a) = \frac{\sum_{h=1}^H n_k^h(s, a, s')}{\max(1, \sum_{h=1}^H n_k^h(s, a))} \quad \hat{\pi}_{k,h}^E(a|s) = \frac{n_k^h(s, a)}{\max(1, n_k^h(s))}.$$

In Appendix B.1.3, we derive Hoeffding's confidence intervals for the transition model and the expert policy. Combining these with Theorem 4.2.1, we can compute the uncertainty on the recovered reward as:

$$C_k^h(s, a) = (H - h)r_{\max} \min \left(1, 2\sqrt{\frac{2\ell_k^h(s, a)}{n_k^h(s, a)}} \right),$$

where $\ell_k^h(s, a) = \log(24|\mathcal{S}||\mathcal{A}|H(n_k^h(s, a))^2/\delta)$. We can show that for any pair of reward functions $r \in \mathcal{R}_{\mathfrak{R}}$ and $\hat{r} \in \mathcal{R}_{\hat{\mathfrak{R}}}$, the difference $|r_h(s, a) - \hat{r}_{k,h}(s, a)| \leq C_k^h(s, a)$ (see Appendix B.1). This uncertainty estimate will be a key component of our theoretical analysis.

4.3.2 Uniform Sampling Strategy

In each iteration k , the *uniform sampling* strategy allocates n_{\max} samples uniformly over $[H] \times \mathcal{S} \times \mathcal{A}$. It estimates the reward uncertainty and stops as soon as $H \max_{h,s,a} C_k^h(s, a) \leq \epsilon$. The following theorem characterizes the sample complexity of uniform sampling with a generative model.

Theorem 4.3.1 (Sample Complexity of Uniform Sampling IRL). The uniform sampling strategy fulfills Definition 4.2.2 with a number of samples upper bounded by:

$$n \leq \tilde{O}\left(H^5 r_{\max}^2 |\mathcal{S}| |\mathcal{A}| / \epsilon^2\right),$$

where \tilde{O} suppresses logarithmic terms.

This sample complexity bound appears slightly worse than the one in Metelli *et al.* [72], who find $(1 - \gamma)^{-4}$ which would translate to H^4 in our finite-horizon setting. The discrepancy, however, is due to us considering reward functions that can depend on the timestep h . If we assume the reward function does not depend on h , we gain a factor of H , obtaining the same result as Metelli *et al.* [72].

4.4 ACTIVE EXPLORATION FOR IRL

Let us now turn to our original problem: recovering the expert’s reward function in an unknown environment *without* a generative model. This problem is more challenging since we need to create an exploration strategy to acquire the desired information about the environment. We now propose a novel sample-efficient exploration algorithm for IRL that we call *Active Exploration for Inverse Reinforcement Learning* (AceIRL). The algorithm takes inspiration from recent works on (reward-free) exploration in RL [85, 86]. We divide the explanation of the algorithm into two parts. First, we introduce a simplified version of the algorithm, which comes with a problem independent sample complexity result (Section 4.4.1). Next, we introduce the full algorithm, which considers the expected reduction of uncertainty in the next iteration to improve exploration and maintains a confidence set of plausibly optimal policies to focus on the most relevant regions (Section 4.4.2). The full algorithm provides a tighter, problem-dependent sample complexity bound (Section 4.4.3). Algorithm 3 contains pseudo-code for AceIRL, and Appendix B.1 contains the detailed theoretical analysis, including proofs of all results discussed here.

4.4.1 Uncertainty-Based Exploration for IRL

The first idea of AceIRL is similar to reward-free UCRL [85]. Our goal is to fulfill the PAC requirement in Definition 4.2.2. Hence, we start from an upper bound on the estimation error between the performance of the optimal policy $\hat{\pi}^*$ for a reward $\hat{r} \in \mathcal{R}_{\mathfrak{B}}$ in the recovered feasible set and the optimal policy π^* for a reward function $r \in \mathcal{R}_{\mathfrak{B}}$ in the true MDP \mathcal{M} . We will then use this upper bound to drive the exploration. For each timestep h and iteration k , we define the error:

$$\hat{e}_k^h(s, a; \pi^*, \hat{\pi}^*) = \left| Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, h}(s, a) \right|. \quad (4.1)$$

We can define an upper bound on these errors recursively with $C_k^H(s, a) = 0$ and

$$E_k^h(s, a) = \min \left((H - h)r_{\max}, C_k^h(s, a) + \sum_{s'} \hat{P}(s'|s, a) \max_{a' \in \mathcal{A}} E_k^{h+1}(s', a') \right). \quad (\text{EB1})$$

It is straightforward to show that $\hat{e}_k^h(s, a; \pi^*, \hat{\pi}^*) \leq E_k^h(s, a)$ for any two policies $\pi^*, \hat{\pi}^*$. Using this error bound, we can introduce a simplified version of AceIRL that explores greedily w.r.t. $E_k^h(s, a)$. We call this algorithm “AceIRL Greedy”. Note that this is equivalent to solving the RL problem defined by $\mathcal{M} \cup C_k^h$; hence, we can use any RL solver to find the exploration policy in practice. If we explore with this greedy policy, we can stop if:

$$4 \max_a E_k^0(s_0, a) \leq \epsilon. \quad (\text{SP1})$$

We can show that when this stopping condition holds, the solution fulfills the PAC requirement 4.2.2. Furthermore, we show in Appendix B.1.4 that AceIRL Greedy achieves a sample complexity on order $\tilde{O}(H^5 r_{\max}^2 |\mathcal{S}| |\mathcal{A}| / \epsilon^2)$, which matches the sample complexity of uniform sampling *with a generative model*. This is already a strong result implying that we do not need a generative model to achieve a good sample complexity for IRL. However, we can improve the algorithm further.

Algorithm 3 AceIRL algorithm for IRL in an unknown environment.

Require: significance $\delta \in (0, 1)$, target accuracy ϵ ,

- 1: IRL algorithm \mathcal{A} , number of episodes N_E
 - 2: Initialize $k \leftarrow 0$, $\epsilon_0 \leftarrow H/10$
 - 3: **while** $\epsilon_k > \epsilon/4$ **do**
 - 4: Solve (convex) optimization problem (ACE) to obtain π_k
 - 5: Explore with policy π_k for N_E episodes
 - 6: During exploration, observe transitions and expert actions
 - 7: $k \leftarrow k + 1$
 - 8: Update $\hat{P}_k, \hat{\pi}_k, C_k^h$, and $\hat{r}_k \leftarrow \mathcal{A}(\mathcal{R}, \mathfrak{B})$
 - 9: Update accuracy $\epsilon_k \leftarrow \max_a \hat{E}_k^0(s_0, a)$
 - 10: **return** estimated reward function \hat{r}_k
-

4.4.2 Adaptive Exploration

AceIRL Greedy is limited in two ways: (i) it explores states that have high uncertainty so far, whereas our goal is to reduce *future* uncertainty, and (ii) it explores to reduce the uncertainty about all policies, whereas our goal is to reduce the uncertainty primarily about *plausibly optimal* policies. To address these limitations, we propose two modifications that yield the full AceIRL algorithm.

REDUCING FUTURE UNCERTAINTY. The greedy policy w.r.t. E_k^h explores states where the estimation error on the Q-functions is large. However, note that this is not exactly what we want, namely, to reduce the uncertainty the most. In particular, if we explore for more than one episode before updating the exploration policy, we should choose an exploration policy that considers how the uncertainty will reduce during exploration. Ideally, we would explore with a policy that minimizes E_{k+1}^h . However, we cannot compute this quantity exactly. Instead, we can approximate it using our current estimate of

the transition model. Concretely, if we have an exploration policy π , we can estimate the reward uncertainty at the next iteration as:

$$\hat{C}_{k+1}^h(s, a) = (H - h)r_{\max} \min \left(1, 2\sqrt{\frac{2\ell_k^h(s, a)}{n_k^h(s, a) + \hat{n}_\pi^h(s, a)}} \right),$$

where $\hat{n}_\pi^h(s, a) = N_E \cdot \mu_{\mathcal{M}, \pi}^{0, h}(s, a | s_0)$ is the expected number of times π visits (s, a) at time h and N_E is the number of episodes we will explore with π . We can use this estimate to find a policy that minimizes our estimate of E_k^{h+1} . While our original approach was akin to “uncertainty sampling”, we now have a better way to measure the “informativeness” of choosing an exploration policy. This is a common pattern when designing query strategies in active learning [21]. Note that this argument does not rely on the IRL problem and can be used to independently improve algorithms for reward-free exploration (cf. Section 4.6).

FOCUSING ON PLAUSIBLY OPTIMAL POLICIES. By exploring greedily w.r.t. E_k^h , we reduce the estimation error of all policies. However, we are primarily interested in estimating the distance between policies $\pi^* \in \Pi_{\mathcal{M} \cup r}^*$ and $\hat{\pi}^* \in \Pi_{\mathcal{M} \cup \hat{r}}^*$ with $r \in \mathcal{R}_{\mathfrak{B}}$ and $\hat{r} \in \mathcal{R}_{\hat{\mathfrak{B}}}$. Of course, we do not know these sets, so we cannot use them directly to target the exploration. Instead, assume we can construct a set of plausibly optimal policies $\hat{\Pi}_k$ that contains all π^* and $\hat{\pi}_k^*$ with high probability. Then, we can redefine our upper bounds recursively as $\hat{E}_k^H(s, a) = 0$ and:

$$\hat{E}_k^h(s, a) = \min \left((H - h)r_{\max}, C_k^h(s, a) + \sum_{s'} \hat{P}(s' | s, a) \max_{\pi \in \hat{\Pi}_{k-1}} \pi(a' | s') \hat{E}_k^{h+1}(s', a') \right), \quad (\text{EB2})$$

In contrast to (EB1), we maximize over policies in $\hat{\Pi}_k$ rather than all actions. Acting greedily w.r.t. $\hat{E}_k^h(s, a)$ is equivalent to finding the optimal policy $\pi_k \in \hat{\Pi}_k$ for the RL problem defined by $\mathcal{M} \cup C_k^h$. We use an arbitrary IRL algorithm \mathcal{A} to construct the set of

plausibly optimal policies. We only assume that \mathcal{A} will return a reward function $\hat{r}_k \in \mathcal{R}_{\mathfrak{B}}$. Then, we can construct a set of plausibly optimal policies as $\hat{\Pi}_k = \{\pi \mid V_{\mathcal{M} \cup \hat{r}_k}^*(s_0) - V_{\mathcal{M} \cup \hat{r}_k}^{\pi}(s_0) \leq 10\epsilon_k\}$. We show in Appendix B.1.5 that $\hat{\Pi}_k$ contains both π^* and $\hat{\pi}_k^*$ with high probability. This choice is based on ideas by Zanette *et al.* [87].

We can define a stopping condition analogously to (SP1):

$$4 \max_a \hat{E}_k^0(s_0, a) \leq \epsilon. \quad (\text{SP2})$$

Again, we can prove that if the algorithm stops due to (SP2), then $\mathcal{R}_{\mathfrak{B}}$ respects Definition 4.2.2.

IMPLEMENTING ACEIRL. To implement the full algorithm, we need to solve an optimization problem:

$$\pi_k \in \underset{\pi}{\operatorname{argmin}} \max_{\hat{\pi} \in \hat{\Pi}_{k-1}} \hat{E}_{k+1}^0(s_0, \hat{\pi}(s_0)) \quad (\text{ACE})$$

The solution to this problem is the exploration policy that minimizes the uncertainty at the next iteration about plausibly optimal policies. This combines both modifications we just discussed. This problem might seem difficult to solve at first, but, perhaps surprisingly, it can be formulated as a convex optimization problem solvable with standard techniques (cf. Appendix B.1.6).

4.4.3 Sample Complexity of AceIRL

In this section, we present our main result about the sample complexity of AceIRL. The result is problem-dependent and, in particular, depends on the advantage function $A_{\mathcal{M} \cup r}^{*,h}(s, a)$, where r is the reward function in the exact feasible set $\mathcal{R}_{\mathfrak{B}}$ closest to the reward function \hat{r}_k which belongs to the estimated feasible set $\mathcal{R}_{\mathfrak{B}}$. The advantage function $A_{\mathcal{M} \cup r}^{*,h}(s, a)$ acts similarly to a suboptimality gap: the closer the value of the second best action is to the best action, the harder it is to identify the best action and infer the correct reward function.

Theorem 4.4.1. [AceIRL Sample Complexity] AceIRL returns a (ϵ, δ, n) -correct solution with

$$n \leq \tilde{O} \left(\min \left[\frac{H^5 r_{\max}^2 |\mathcal{S}| |\mathcal{A}|}{\epsilon^2}, \frac{H^4 r_{\max}^2 |\mathcal{S}| |\mathcal{A}| \epsilon_{\tau-1}^2}{\min_{s,a,h} (A_{\mathcal{M} \cup r}^{*,h}(s,a))^2 \epsilon^2} \right] \right)$$

where $\epsilon_{\tau-1}$ depends on the choice of N_E , the number of episodes of exploration in each iteration. $A_{\mathcal{M} \cup r}^{*,h}(s,a)$ is the advantage function of $r \in \operatorname{argmin}_{r \in \mathcal{R}_{\mathfrak{B}}} \max_{h,s,a} (r_h(s,a) - \hat{r}_{k,h}(s,a))$, the reward function from the feasible set $\mathcal{R}_{\mathfrak{B}}$ closest to the estimated reward function \hat{r}_k .

This result is a minimum of two terms. The first term is problem independent, and it is achieved both by AceIRL Greedy and the full AceIRL. This bound matches the bound we saw previously with a generative model. Hence, AceIRL achieves the same results without access to the generative model. Using (ACE) can yield a better sample complexity, represented by the second term in the minimum. This bound depends on two main components: the ratio $\epsilon_{\tau-1}/\epsilon$ and the advantage function $A_{\mathcal{M} \cup r}^{*,h}(s,a)$. The ratio depends on the choice of N_E , the number of exploration episodes per iteration. If N_E is small, then the ϵ -ratio will also be small. If N_E is large, the algorithm will perform similarly to a uniform sampling strategy. Appendix B.1.5 provides the full proof of this theorem.

4.5 EXPERIMENTS

We perform a series of simulation experiments to evaluate AceIRL. We simulate a (deterministic) expert policy using an underlying true reward function and compare it to the recovered reward functions.

Our main evaluation metric is the *normalized regret*:

$$\frac{V_{\mathcal{M} \cup r}^{\pi^*,0}(s_0) - V_{\mathcal{M} \cup r}^{\hat{\pi}^*,0}(s_0)}{V_{\mathcal{M} \cup r}^{\pi^*,0}(s_0) - V_{\mathcal{M} \cup r}^{\bar{\pi}^*,0}(s_0)},$$

where π^* is the optimal policy for $\mathcal{M} \cup r$, $\hat{\pi}^*$ is the optimal policy for $\widehat{\mathcal{M}} \cup \hat{r}$, and $\bar{\pi}^*$ is the worst possible policy for r , i.e., the optimal policy for $\mathcal{M} \cup (-r)$.

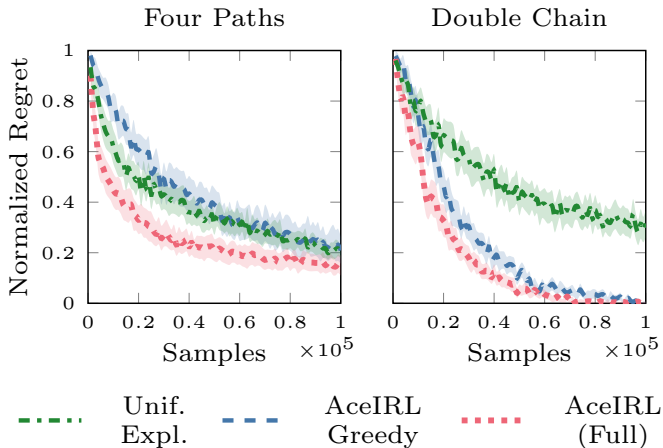


FIGURE 4.2: Normalized regret (lower is better) of the policy optimized for the inferred reward in the estimated MDP as a function of the number of samples. The plots show mean and 95% confidence intervals computed using 50 random seeds ($N_E = 50$).

We introduce the *Four Paths* environment shown in Figure 4.1, which consists of four chains of states with different randomly sampled transition probabilities. One path has a goal with reward 1; all other rewards are 0. We also evaluate on *Random MDPs* with uniformly sampled transition dynamics and reward functions, the *Double Chain* environment proposed by Kaufmann *et al.* [85], and the *Chain* and *Gridworld* environments proposed by Metelli *et al.* [72]. Appendix B.2.1 provides details on the transition dynamics and rewards of all environments.

We compare AceIRL and AceIRL Greedy to a uniformly random exploration policy as a naive exploration strategy. Further, we consider uniform sampling with a generative model and TRAVEL [72], which can be more sample efficient because they do not need to explore the environment. Note that TRAVEL is designed to learn a reward to be transferred to a known target environment. Instead, we use a modified version that uses the estimated MDP as a target.

Table 4.1 shows the sample efficiency of all algorithms in all environments, measured as the number of samples needed to achieve a regret threshold of 0.4 (different thresholds yield similar conclusions; see Appendix B.2). AceIRL is the most sample efficient exploration strategy without access to a generative model, but the relative differences between the methods depend on the environment. In some cases, AceIRL even performs comparably to methods using a generative model, as the theory predicts.

In the *Four Paths* and *Double Chain* environments, we also vary the N_E parameter. AceIRL performs better for small values at the computational cost of updating the exploration policy more often. If N_E is too large, using AceIRL can be as bad as a uniformly random exploration policy. Increasing N_E hurts the performance of AceIRL Greedy more severely, which does not consider N_E explicitly. Figure 4.2 shows the normalized regret as a function of the number of samples in *Four Paths* and *Double Chain*. In both cases AceIRL performs best. However, AceIRL Greedy is worse than random exploration in the *Four Paths* environment. Hence, we find that the problem dependent exploration strategy of the full algorithm significantly improves the sample efficiency.

4.6 CONNECTION TO REWARD-FREE EXPLORATION

In the *reward-free exploration* problem, introduced by Jin *et al.* [88], the agent explores an MDP $\setminus R$ to learn a transition model. In each iteration, the agent chooses a new exploration policy based on previous data. The goal is to ensure that if the agent is given a reward function r after the exploration phase, it can find a good policy using its transition model. Jin *et al.* [88] formalize this goal as reducing the error:

$$V_{\mathcal{M} \cup r}^{\hat{\pi}^*, 0}(s_0) - V_{\mathcal{M} \cup r}^{\hat{\pi}^*, 0}$$

where $\hat{\pi}^*$ is the optimal policy in the estimated MDP $\widehat{\mathcal{M}} \cup r$. Note the striking similarity between this problem and our active IRL problem. In active IRL we want to reduce a similar error (Definition 4.2.2), but

	Uniform sampling (gener. model)	TRAVEL (gener. model) [72]	Random Exploration	AceIRL Greedy	AceIRL (Full)
Four Paths (Figure 4.1)	1900 ± 71		17840 ± 1886		
– $N_E = 50$		1560 ± 76		24180 ± 1747	10780 ± 1369
– $N_E = 100$		2000 ± 0		32760 ± 2172	14080 ± 1603
– $N_E = 200$		4000 ± 0		52000 ± 4057	16160 ± 2033
Double Chain [85]	1980 ± 66		23640 ± 2195		
– $N_E = 50$		1120 ± 46		16240 ± 842	11580 ± 870
– $N_E = 100$		2000 ± 0		22200 ± 1329	15440 ± 1031
– $N_E = 200$		4000 ± 0		37200 ± 1664	20400 ± 1629
Metelli <i>et al.</i> [72]:					
Random MDPs ($N_E = 1$)	22 ± 1	27 ± 1	22 ± 1	23 ± 1	21 ± 1
Chain ($N_E = 1$)	78 ± 2	76 ± 4	161 ± 8	153 ± 8	142 ± 9
Gridworld ($N_E = 1$)	43 ± 2	35 ± 2	45 ± 2	46 ± 3	48 ± 2

TABLE 4.1: Sample complexity of AceIRL compared to random exploration and methods that use a generative model. We show the number of samples necessary to find a policy with normalized regret of less than 0.4. We report the means and standard errors computed over 50 random seeds each. For each environment, we highlight in **bold** the method that achieves the best performance without access to a generative model. If multiple methods are within one standard error distance, we highlight all of them. Overall, AceIRL is the most sample efficient method without a generative model if N_E is chosen small enough. In Appendix B.2.2, we show learning curves for all individual experiments.

we have additional information about the reward in the form of the expert policy.

The *Reward-free UCRL* algorithm, proposed by Kaufmann *et al.* [85], is similar to AceIRL Greedy (Section 4.4.1). Reward-free UCRL explores greedily w.r.t. an upper bound on the value function error. However, the exploration policy needs to be updated after each episode to adapt to the new uncertainty estimates. This might be expensive or not possible in practice. Instead, we could consider a *batched* version of reward-free exploration, where the agent explores for N_E episodes in each iteration, similar to our active IRL problem. In this setting, a greedy policy w.r.t. uncertainty is suboptimal because it does not adapt to the reduced uncertainty over the N_E episodes.

Instead, we can consider reducing the expected uncertainty at the next iteration, similar to our discussion in Section 4.4.2. If our error estimate is denoted by $E_k(s, a)$, we no longer act greedily w.r.t. E_k . Instead, we try to estimate the error at the next iteration $\hat{E}_{k+1}(s, a|\pi)$ as a function of the policy and try to select the policy that reduces this error. In the tabular case, we can formulate this as a convex optimization problem, analogous to Appendix B.1.6. We call this adaptation of AceIRL to the reward-free exploration problem *Ace-RF*.

Figure 4.3 shows illustrative results of this algorithm in the batched reward-free exploration setting in the *Double Chain* environment. We find that choosing an exploration policy that reduces future uncertainty is significantly better for larger batch sizes than reward-free UCRL.

4.7 CONCLUSION

We considered active Inverse Reinforcement Learning (IRL) with unknown transition dynamics and expert policy and introduced AceIRL, an efficient exploration strategy to learn about both the dynamics and the expert policy with the goal of inferring the reward function as efficiently as possible.

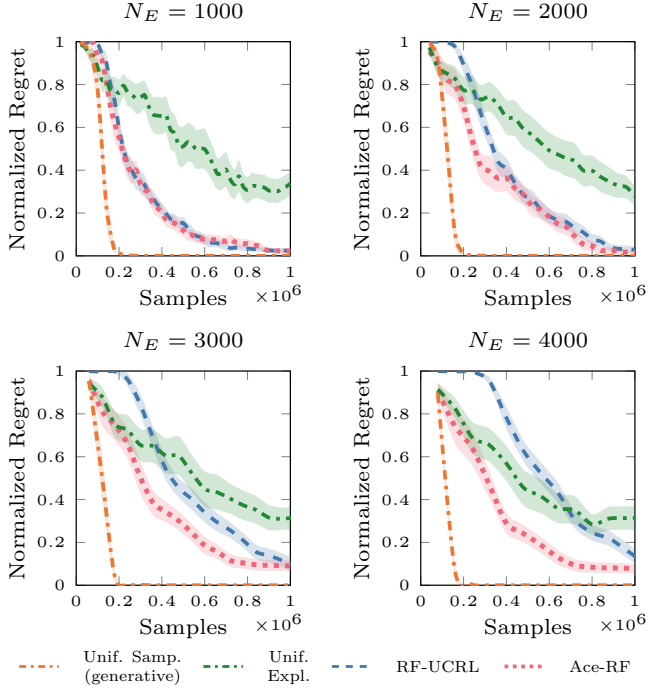


FIGURE 4.3: Illustrative experiments for reward-free exploration in the *Double Chain* environment proposed by Kaufmann *et al.* [85]. The difference to our active IRL setting is that the agent does not have access to the expert policy during exploration but still tries to learn a good model of the environment. During testing, it then gets access to the reward function, and the regret measures the suboptimality of the policy trained in the agent’s transition model. We find that the ideas used in AceIRL are also useful for batched reward-free exploration with large N_E .

Our approach is a crucial step towards IRL algorithms with theoretical guarantees, but future work is needed to move to more practical settings. In particular, it would be interesting to extend the approach to continuous state and action spaces (e.g., using function approximation) and to obtain an efficient algorithm that does not require solving convex optimization problems. From a theoretical perspective, it would be useful to derive a lower bound on the sample complexity of the IRL problem to understand if the IRL problem is inherently more difficult than usual RL.

While this chapter focused on IRL and formal analysis of the sample complexity, the ideas we used when designing AceIRL are similar to those we encountered in Chapter 3. Both IDRL and AceIRL determine a set of plausibly optimal policies and then try to reduce the uncertainty about the difference in policy returns. This suggests that future work could use similar ideas to scale up AceIRL to more practical settings or even combine both approaches to develop an active learning algorithm that explores the environment first and then selects the most informative experiences to ask for human feedback.

Part II

CONSTRAINT LEARNING FOR REINFORCEMENT LEARNING

LEARNING CONSTRAINTS FROM DEMONSTRATIONS WITHOUT REWARDS

So far, we studied learning reward models from human feedback. In the second part of this dissertation, we propose constraints as an alternative representation of human preferences.

Constrained Markov Decision Problems (CMDPs; [37]) integrate safety constraints with Reinforcement Learning (RL). However, similar to reward functions, it can be difficult to specify constraint functions manually. To tackle this issue, recent work proposes learning models of the constraints (e.g., [89, 90]) analogously to reward models [44]. However, existing approaches to constraint inference rely on demonstrations with *known reward functions*, which is often unrealistic in real-world applications.

For instance, consider the domain of autonomous driving (Figure 5.1), where specifying rewards/constraints is particularly difficult [11]. We can gather diverse human driving trajectories that satisfy shared constraints, such as keeping an appropriate distance to other cars and avoiding crashes. But the trajectories will usually have different (unknown) routes or driving style preferences, which we can model as different reward functions. In such scenarios, we aim to infer constraints from demonstrations with *shared constraints* but *unknown rewards*.

This example motivates the problem setting we study in this chapter. In particular:

- We introduce the problem of inferring constraints in CMDPs from demonstrations with unknown rewards (Section 5.2.1).
- We introduce *Convex Constraint Learning for Reinforcement Learning* (CoCoRL), to address this problem (Section 5.3)

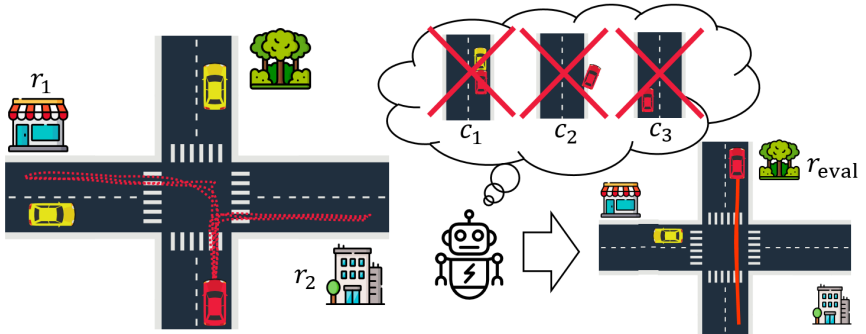


FIGURE 5.1: Illustrative example of constraint learning for autonomous driving. CoCoRL can learn safe driving behavior from diverse driving trajectories with different unknown goals. It infers constraints c_1, c_2, c_3 describing desirable driving behavior from demonstrations without knowledge of the specific reward functions r_1, r_2 . These inferred constraints allow to optimize for a new reward function r_{eval} , ensuring safe driving behavior even in new situations without demonstrations.

- We prove that CoCoRL guarantees safety (Section 5.3.1), and, for (approximately) optimal demonstrations, also guarantees asymptotic optimality (Section 5.3.2), while IRL provably cannot guarantee safety (Section 5.2.2).
- We conduct a comprehensive empirical evaluation of CoCoRL in tabular environments and a continuous driving task with multiple constraints (Section 6.3). We find that CoCoRL learns constraints that lead to safe driving behavior and that can be transferred across different tasks and environments, whereas IRL-based methods often perform poorly.

The contents of this chapter are based on:

D. Lindner, X. Chen, S. Tschitschek, K. Hofmann, and A. Krause, “Learning safety constraints from demonstrations with unknown rewards”, *arXiv preprint arXiv:2305.16147*, 2023.

5.1 RELATED WORK

CONSTRAINT LEARNING IN RL. Previous work on safe RL typically assumes fully known safety constraints (e.g., see the review by Garcia and Fernández [91]). Recently, there has been a line of research on constraint learning to address this limitation. Scobee and Sastry [89] and Stocking *et al.* [92] use maximum likelihood estimation to learn constraints from demonstrations with known rewards in tabular and continuous environments, respectively. Malik *et al.* [90] propose a scalable algorithm based on maximum entropy IRL, and extensions to using maximum causal entropy have been explored [93–95]. Papadimitriou *et al.* [96] adapt Bayesian IRL to learning constraints via a Lagrangian formulation. However, all of these approaches assume knowledge of the demonstrations’ reward functions. Chou *et al.* [97] use a mixed-integer linear program based on the Karush–Kuhn–Tucker (KKT) conditions to infer constraints from safe demonstrations. Their method allows for parametric uncertainty on the rewards, but it requires fully known environment dynamics and is not applicable to general CMDPs.

MULTI-TASK IRL. Multi-task IRL addresses the problem of learning from demonstrations in different but related tasks. Some methods reduce multi-task IRL to multiple single-task IRL problems [98, 99], but they do not fully leverage the similarity between demonstrations. Others treat multi-task IRL as a meta-learning problem, focusing on quickly adapting to new tasks [100–103]. However, IRL-based methods are difficult to adapt to CMDPs where safety is crucial.

LEARNING CONSTRAINTS FOR DRIVING BEHAVIOR. In the domain of autonomous driving, inferring constraints from demonstrations has attracted significant attention due to safety concerns. Rezaee and Yadmellat [104] extend the method by Scobee and Sastry [89] with a VAE-based gradient descent optimization to learn driving constraints. Liu *et al.* [105] propose a benchmark for constraint inference with known rewards based on the highD dataset, which provides

highway driving trajectories [106]. Gaurav *et al.* [107] also evaluate their method for learning constraints on the highD dataset. However, these approaches, again, assume known reward functions, which is often unrealistic beyond basic highway driving scenarios.

BANDITS & LEARNING THEORY. Our approach draws inspiration from constraint inference in other fields. For instance, learning safety constraints has been studied in best-arm identification in linear bandits [108]. Our problem is also related to one-class classification (e.g., [109]), the study of random polytopes (e.g., [110]), and PAC-learning of convex polytopes (e.g., [111]). However, none of these methods are directly applicable to the structure of our problem.

5.2 INFERRING CONSTRAINTS WITH UNKNOWN REWARDS

In this section, we introduce the problem of inferring constraints from demonstrations in a CMDP. Then, we discuss why naive solutions based on IRL cannot solve this problem.

5.2.1 Problem Setup

We consider an infinite-horizon discounted CMDP without reward function and cost functions, denoted by $(\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma)$. We have access to demonstrations from k safe policies $\mathcal{D} = \{\pi_1^*, \dots, \pi_k^*\}$, that have k unknown reward functions r_1, \dots, r_k , but share the same constraints defined by unknown cost functions c_1, \dots, c_n and thresholds ξ_1, \dots, ξ_n . We assume policy π_i^* is safe in the CMDP $(\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r_i, \{c_j\}_{j=1}^n, \{\xi_j\}_{j=1}^n)$. Generally, we assume policy π_i^* belongs to reward r_i , but it does not need to be optimal.

In our driving example (Figure 5.1), the rewards of the demonstrations can correspond to different routes and driver preferences, whereas the shared constraints describe driving rules and safety-critical behaviors, such as maintaining an appropriate distance to other cars or staying in the correct lane.

We assume that reward and cost functions are defined on a d -dimensional feature space, represented by $\mathbf{f} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]^d$. Instead of having full policies, we have access to their discounted feature expectations (slightly overloading notation) defined as $\mathbf{f}(\pi) = \mathbb{E}_{P, \pi}[\sum_{t=0}^{\infty} \gamma^t \mathbf{f}(s_t, a_t)]$. The reward and cost functions linear in the feature space, given by $r_i(s, a) = \theta_i^T \mathbf{f}(s, a)$ and $c_j(s, a) = \phi_j^T \mathbf{f}(s, a)$, where $\theta_i \in \mathbb{R}^d$ and $\phi_j \in \mathbb{R}^d$ are the respective parameter vectors. For now, we assume we know the exact feature expectations of the demonstrations. In Section 5.3.3, we discuss sample-based estimation.

Importantly, in the case of a finite state and action space, assuming a feature space is *w.l.o.g.*, because we can use the state-action occupancy measure $\mu_\pi(s, a)$ as features (as we did in Chapter 3). Specifically, we can define $\mathbf{f}(s, a) = (\mathbb{1}_{\{s=s_1, a_1\}}, \mathbb{1}_{\{s=s_1, a_2\}}, \dots)$ as an indicator vector with dimension $d = |\mathcal{S}| |\mathcal{A}|$. Then, $\mathbf{f}(\pi) \in \mathbb{R}^{|\mathcal{S}| |\mathcal{A}|}$ is a vector that contains the state-action occupancy values μ_π , and we have $G_r(\pi) = \theta^T \mathbf{f}(\pi) = \sum_{s, a} r(s, a) \mu_\pi(s, a)$, i.e., the reward parameter θ is a vector of the state-action rewards. Similarly, the constraint parameters ϕ_j are vectors of state-action costs c_j .

For evaluation, we receive a new reward function r_{eval} and aim to find an optimal policy for the CMDP $(\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r_{\text{eval}}, \{c_j\}_{j=1}^n, \{\tilde{\zeta}_j\}_{j=1}^n)$ where the constraints are unknown. Our goal is to ensure safety while achieving a good reward under r_{eval} .

5.2.2 Limitation of IRL in CMDPs

Initially, we might disregard the CMDP nature of the problem and try to apply standard IRL methods to infer a reward from demonstrations. This approach poses at least two key problems: (1) IRL cannot guarantee safety, and (2) it is unclear how to learn a transferable representation of the constraints.

To elaborate, let us first assume we have a single expert demonstration from a CMDP and apply IRL to infer a reward function, assuming no constraints. Unfortunately, we can show that, in some

CMDPs, all reward functions obtained through IRL can yield unsafe optimal policies.

Proposition 5.2.1 (IRL can be unsafe). There are CMDPs $\mathcal{C} = (\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r, \{c_j\}_{j=1}^n, \{\tilde{\zeta}_j\}_{j=1}^n)$ such that for any optimal policy π^* in \mathcal{C} and any reward function r_{IRL} that could be returned by an IRL algorithm, the resulting MDP $(\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r_{\text{IRL}})$ has optimal policies that are unsafe in \mathcal{C} .

This follows because CMDPs can have only stochastic optimal policies, whereas MDPs always have a deterministic optimal policy [33, 37]. See Appendix C.1 for a full proof of this and other theoretical results in this chapter.

Furthermore, even if we manage to learn a reward function that represents the constraints, it remains unclear how to transfer it to a new task represented by the reward function r_{eval} . One potential approach is to model the constraints as a *shared* reward penalty. Unfortunately, it turns out that learning this shared penalty can be infeasible, even when the rewards are known.

Proposition 5.2.2. Let $\mathcal{C} = (\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, \{c_j\}_{j=1}^n, \{\tilde{\zeta}_j\}_{j=1}^n)$ be a CMDP without reward. Let r_1, r_2 be two reward functions and π_1^* and π_2^* corresponding optimal policies in $\mathcal{C} \cup \{r_1\}$ and $\mathcal{C} \cup \{r_2\}$. Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma)$ be the corresponding MDP without reward. Without additional assumptions, we cannot guarantee the existence of a function $\hat{c} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ such that π_1^* is optimal in the MDP $\mathcal{M} \cup \{r_1 + \hat{c}\}$ and π_2^* is optimal in the MDP $\mathcal{M} \cup \{r_2 + \hat{c}\}$.

Similar to Proposition 5.2.1, this result stems from the fundamental difference between MDPs and CMDPs. The main challenge is to appropriately scale the reward penalty \hat{c} for a specific reward function. While it is always possible to find a suitable \hat{c} to describe a single expert policy with a single known reward function, the situation becomes more complex when dealing with multiple expert policies π_1^*, π_2^* and reward functions r_1, r_2 . In such cases, differently scaled penalties may be required, and the problem of learning a shared \hat{c}

can be infeasible. These findings strongly suggest that IRL is not suitable for learning from demonstrations from a CMDP. Our empirical findings in Section 6.3 confirm this.

5.3 CONVEX CONSTRAINT LEARNING FOR RL

We are now ready to introduce the CoCoRL algorithm. We construct the full algorithm step-by-step: (1) we introduce the key idea of using convexity to construct provably conservative safe set (Section 5.3.1); (2) we establish convergence results under (approximately) optimal demonstrations (Section 5.3.2); (3) we introduce approaches for using estimated feature expectations (Section 5.3.3); and, (4), we discuss practical considerations for implementing CoCoRL (Section 5.3.4).

Importantly, when constructing the conservative safe set, we only require access to safe demonstrations. For convergence, we make certain optimality assumptions, either exact optimality or Boltzmann-rationality. Appendix C.1 contains all omitted proofs.

5.3.1 Constructing a Convex Safe Set

Let the *true safe set* $\mathcal{F} = \{\pi | \forall j : J_j(\pi) \leq \xi_j\}$ be the set of all policies that satisfy the constraints. CoCoRL is based on the key observation that \mathcal{F} is convex.

Lemma 5.3.1. For any CMDP, suppose $\pi_1, \pi_2 \in \mathcal{F} = \{\pi | \forall j : J_j(\pi) \leq \xi_j\}$, and let $\bar{\pi}_{12}$ be a policy such that $\mathbf{f}(\bar{\pi}_{12}) = \lambda \mathbf{f}(\pi_1) + (1 - \lambda) \mathbf{f}(\pi_2)$ with $\lambda \in [0, 1]$. Then $\bar{\pi}_{12} \in \mathcal{F}$.

Proof. $\pi_1, \pi_2 \in \mathcal{F}$ means for any j , we have $J_j(\pi_1) \leq \xi_j$ and $J_j(\pi_2) \leq \xi_j$. Then, also for any j , we have $J_j(\bar{\pi}_{12}) = \phi_j^T \mathbf{f}(\bar{\pi}_{12}) = \lambda \phi_j^T \mathbf{f}(\pi_1) + (1 - \lambda) \phi_j^T \mathbf{f}(\pi_2) \leq \xi_j$. Thus, $\bar{\pi}_{12} \in \mathcal{F}$. \square

We know that all demonstrations are safe in the original CMDP and convex combinations of their feature expectations are safe. This insight leads us to a natural approach for constructing a conservative

safe set: create the convex hull of the feature expectations of the demonstrations, i.e.,

$$S = \text{conv}(\mathcal{D}) := \left\{ \mathbf{x} = \sum_{i=1}^k \lambda_i \mathbf{f}(\pi_i^*) \mid \lambda_i \geq 0 \text{ and } \sum_i \lambda_i = 1 \right\}.$$

To simplify notation, we will sometimes write $\pi \in S$ to mean $\mathbf{f}(\pi) \in S$. As \mathbf{f} is fixed, this notation is unambiguous. Thanks to the convexity of \mathcal{F} , we can now guarantee safety for all policies $\pi \in S$.

Theorem 5.3.1 (Estimated safe set). Any policy $\pi \in S$ is safe, i.e., we have $\pi \in \mathcal{F}$.

Proof. Our demonstrations π_1^*, \dots, π_k^* are all in the true safe set \mathcal{F} , and any $\pi \in S$ is a convex combination of them. Given that \mathcal{F} is convex (Lemma 5.3.1), we have $S \subseteq \mathcal{F}$. \square

During evaluation, we are given a new reward r_{eval} , and we want to find $\pi^* \in \text{argmax}_{\pi \in S} G_{r_{\text{eval}}}(\pi)$. Conveniently, we can reduce this problem to solving a new CMDP with identical dynamics. Specifically, we can find a set of new cost functions parameterized by $\hat{\phi}_1, \dots, \hat{\phi}_m$ and thresholds $\hat{\zeta}_1, \dots, \hat{\zeta}_m$, such that $S = \{x \mid \hat{\phi}_j^T x \leq \hat{\zeta}_j \text{ for all } j\}$. This result follows from standard convex analysis: S is a convex polyhedron, which is the solution to a set of linear equations. Constructing a convex hull from a set of points and identifying the linear equations is a well-studied problem in polyhedral combinatorics (e.g., [112]).

Consequently, we can optimize within our safe set S by identifying and solving an “inferred” CMDP.

Theorem 5.3.2 (Inferred CMDP). We can find cost functions and thresholds such that for any reward function r_{eval} , solving the inferred CMDP $(\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r_{\text{eval}}, \{\hat{c}_j\}_{j=1}^m, \{\hat{\zeta}_j\}_{j=1}^m)$ is equivalent to solving $\pi^* \in \text{argmax}_{\pi \in S} G_{r_{\text{eval}}}(\pi)$. Consequently, if an optimal policy for the true CMDP is in S , solving the inferred CMDP will find an optimal policy for the true CMDP.

Note that the number of inferred constraints m is not necessarily equal to the number of true constraints n , but we do not assume to

know n . It immediately follows from Theorem 5.3.2 that our safe set is worst-case optimal because, without additional assumptions, we cannot dismiss the possibility that the true CMDP coincides with our inferred CMDP.

Corollary 5.3.1 (S is maximal). If $\pi \notin S$, there exist r_1, \dots, r_k and c_1, \dots, c_n , such that the expert policies π_1^*, \dots, π_k^* are optimal in the CMDPs $(S, \mathcal{A}, P, \mu_0, \gamma, r_i, \{c_j\}_{j=1}^n, \{\xi_j\}_{j=1}^n)$, but $\pi \notin \mathcal{F}$.

In essence, S is the largest set we can choose while ensuring safety.

5.3.2 Convergence Under (Approximate) Optimality

So far, we studied the safety of CoCoRL. In this section, we establish its convergence to optimality under two conditions: (1) when the safe demonstrations are exactly optimal, and (2) when they are approximately optimal according to a Boltzmann model.

Given a new reward function r_{eval} , we focus on comparing our safe solution $\max_{\pi \in S} G_r(\pi)$ to the optimal safe solution $\max_{\pi \in \mathcal{F}} G_r(\pi)$. In particular, we define the *policy regret*

$$\mathcal{R}(r, S) = \max_{\pi \in \mathcal{F}} G_r(\pi) - \max_{\pi \in S} G_r(\pi).$$

Let us assume the reward functions of the demonstrations and the reward functions during evaluation follow the same distribution $P(r)$. After observing k demonstrations, we construct the safe set S_k , and consider the expectation $\mathbb{E}_r[\mathcal{R}(r, S_k)]$ under the reward distribution. To begin with, let us consider exactly optimal demonstrations.

Assumption 5.3.1 (Exact optimality). Each demonstration π_i^* is optimal in the CMDP $(S, \mathcal{A}, P, \mu_0, \gamma, r_i, \{c_j\}_j, \{\xi_j\}_j)$.

Theorem 5.3.3 (Convergence, exact optimality). Under Assumption 5.3.1, for any $\delta > 0$, after $k \geq \log(\delta / f_v(d, n)) / \log(1 - \delta / f_v(d, n))$, we have $P(\mathcal{R}(r, S_k) > 0) \leq \delta$, where $f_v(d, n)$ is an upper bound on the number of vertices of the true safe set. In particular,

$$\lim_{k \rightarrow \infty} \mathbb{E}_r[\mathcal{R}(r, S_k)] = 0.$$

Due to both the true safe set and the estimated safe set being convex polyhedra, we have a finite hypothesis class consisting of all convex polyhedra with vertices that are a subset of the true safe set's vertices. The proof builds on the insight that all vertices supported by the distribution induced by $P(r)$ will eventually be observed, while unsupported vertices do not contribute to the regret. The number of vertices is typically on the order $f_v(d, n) \in \mathcal{O}(n^{\lfloor d/2 \rfloor})$. A tighter bound is given by the McMullen upper bound theorem [113].

In practical settings, it is unrealistic to assume perfectly optimal demonstrations. Thus, we relax this assumption and consider the case where demonstrations are only approximately optimal.

Assumption 5.3.2 (Boltzmann-rationality). We observe demonstrations following a Boltzmann policy distribution

$$\pi_i^* \sim \exp(\beta G_{r_i}(\pi)) \mathbb{1}_{\{J_1(\pi) \leq \xi_1, \dots, J_n(\pi) \leq \xi_n\}} / Z_i,$$

where $G_{r_i}(\pi)$ is the expected return computed w.r.t. the reward function r_i , and Z_i is a normalization constant.

Note that Theorem 5.3.1 and Corollary 5.3.1 still hold under Assumption 5.3.2, as they only depend on all demonstrations being safe. Also, we can still establish asymptotic optimality.

Theorem 5.3.4 (Convergence, Boltzmann-rationality). Under Assumption 5.3.2, for any $\delta > 0$, after $k \geq \log(\delta / f_v(d, n)) / \log(1 - \exp(-\beta d / (1 - \gamma)))$, we have $P(\mathcal{R}(r, S_k) > 0) \leq \delta$, where $f_v(d, n)$ is an upper bound on the number of vertices of the true safe set. In particular,

$$\lim_{k \rightarrow \infty} \mathbb{E}_r [\mathcal{R}(r, S_k)] = 0.$$

This result relies on the Boltzmann distribution having a fixed lower bound over the bounded set of policy features. By enumerating the vertices of the true safe set, we can establish an upper bound on the probability of the “bad event” where a vertex is not adequately covered by the set of k demonstrations. This shows how the regret must decrease as k increases.

5.3.3 Estimating Feature Expectations

So far, we assumed access to the true feature expectations of the demonstrations $\mathbf{f}(\pi_i^*)$. However, in practice, we often rely on samples from the environment to estimate the feature expectations. Given n_{traj} samples from policy π_i^* , a natural way to estimate the feature expectations is by taking the sample mean: $\hat{\mathbf{f}}(\pi_i^*) = \frac{1}{n_{\text{traj}}} \sum_{j=1}^{n_{\text{traj}}} \mathbf{f}(\tau_j^i)$, where $\mathbf{f}(\tau_j^i) = \sum_{t=0}^{\infty} \gamma^t \mathbf{f}(s_t, a_t)$ and $\tau_j^i = (s_0, a_0, s_1, a_1, \dots)$ represents a trajectory from rolling out π_i^* in the environment. This estimate is unbiased, i.e., $\mathbb{E}[\hat{\mathbf{f}}(\pi_i^*)] = \mathbf{f}(\pi_i^*)$, and we can use standard concentration inequalities to measure its confidence. For instance, by using McDiarmid's inequality, we can ensure ϵ -safety when using the estimated feature expectations.

Theorem 5.3.5 (ϵ -safety with estimated feature expectations). Suppose we estimate the feature expectations of each policy π_i^* using at least $n_{\text{traj}} > d \log(nk/\delta)/(2\epsilon^2(1-\gamma))$ samples, and construct the estimated safe set $\hat{S} = \text{conv}(\hat{\mathbf{f}}(\pi_1^*), \dots, \hat{\mathbf{f}}(\pi_k^*))$. Then, we have $P(\max_j (J_j(\pi) - \xi_j) > \epsilon | \pi \in \hat{S}) < \delta$.

We can extend our analysis to derive convergence bounds for exactly optimal or Boltzmann-rational demonstrations, similar to Theorem 5.3.3 and Theorem 5.3.4. The exact bounds are provided in Appendix C.1.4. Importantly, we have no regret asymptotically:

$$\lim_{\min(k, n_{\text{traj}}) \rightarrow \infty} \mathbb{E}_r[\mathcal{R}(r, S_k)] = 0,$$

in both cases.

If we need to ensure exact safety, we can, alternatively, use confidence intervals around estimated feature expectations to construct a conservative safe set, i.e., a set $\hat{S} \subset S$ that still guarantees *exact* safety with high probability (w.h.p.).

Suppose we have k confidence sets $\mathcal{C}_i \subseteq \mathbb{R}^d$, such that we know $\mathbf{f}(\pi_i^*) \in \mathcal{C}_i$ (w.h.p.). Now we want to construct a conservative convex hull, i.e., a set of points in which any point certainly in the convex

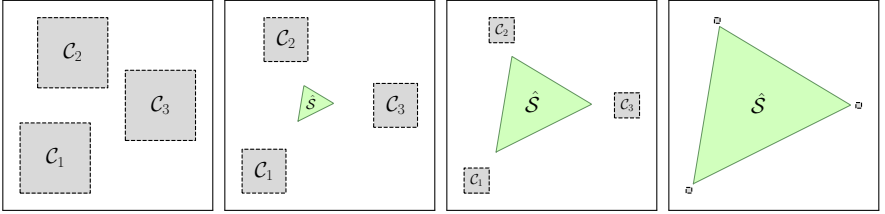


FIGURE 5.2: Illustration of the *guaranteed hull* in 2 dimensions. The gray areas depict confidence sets $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$, and the green area is the conservative safe set \hat{S} constructed using the guaranteed hull. For large confidence sets, the guaranteed hull can be empty (left); for small confidence sets, it approaches the safe set constructed using the exact feature expectations (right).

hull of $\mathbf{f}(\pi_1^*), \dots, \mathbf{f}(\pi_k^*)$. In other words, we want to construct the intersection of all possible convex hulls with points from $\mathcal{C}_1, \dots, \mathcal{C}_k$:

$$\hat{S} = \bigcap_{x_1, \dots, x_k \in \mathcal{C}_1 \times \dots \times \mathcal{C}_k} \text{conv}(x_1, \dots, x_k).$$

This set is sometimes called the *guaranteed hull* [114], or the *interior convex hull* [115]. Efficient algorithms are known to construct guaranteed hulls in 2 or 3 dimensions if \mathcal{C} has a simple shape, such as a cube or a disk (e.g., see [114]). However, we need to construct a guaranteed hull in d dimensions.

Edalat *et al.* [116] propose to reduce constructing a guaranteed hull when \mathcal{C}_i are (hyper-)rectangles to computing the intersection of convex hulls constructed from combinations of the corners of the rectangle. In $d \leq 3$ dimensions, this is possible with $\mathcal{O}(k \log k)$ complexity, but in higher dimensions it requires $\mathcal{O}(k^{\lfloor d/2 \rfloor})$. If we use a concentration inequality for each coordinate independently, e.g., by using Lemma C.1.3 with unit basis vectors, then the \mathcal{C}_i 's are rectangles, and we can use this approach. However, because we have to construct many convex hulls, it can get expensive as k and d grow.

Figure 5.2 illustrates the guaranteed hull growing as the confidence regions shrink. The guaranteed hull is a conservative estimate and can be empty if the \mathcal{C}_i 's are large. Hence, \hat{S} might sometimes be too

conservative, especially if we only observe a single trajectory from each policy. In such cases, we need additional assumptions about the environment (e.g., it is deterministic) or the demonstrations (e.g., every trajectory is safe) to guarantee safety in practice. Still, using a guaranteed hull as a conservative safe set is a promising alternative to relying on point estimates of the feature expectations.

Crucially, CoCoRL can be applied in environments with continuous state and action spaces and nonlinear policy classes, as long as the feature expectations can be computed or estimated.

5.3.4 *Practical Implementation of CoCoRL*

We can implement CoCoRL, using existing CMDP solvers and algorithms for constructing convex hulls. First, we compute or estimate the demonstrations' feature expectations $\mathbf{f}(\pi_i^*)$. Then, we use the Quickhull algorithm [117] to construct a convex hull. Finally, we solve the inferred CMDP from Theorem 5.3.2 using a constrained RL algorithm, the choice depending on the environment. We make two further extensions to enhance the robustness of this algorithm.

HANDLING DEGENERATE SAFE SETS. The demonstrations might lie in a lower-dimensional subspace of \mathbb{R}^d (i.e., have a rank less than d). In that case, we project them onto this lower-dimensional subspace, construct the convex hull in that space, and then project back the resulting convex hull to \mathbb{R}^d . This is beneficial because Quickhull is not specifically designed to handle degenerate convex hulls.

INCREMENTALLY EXPANDING THE SAFE SET. If we construct S from all demonstrations, we will have many redundant points, which can incur numerical instabilities at the boundary of S . Instead, we incrementally add points from \mathcal{D} to S . In each iteration, we add the point furthest away from S , until the distance of the furthest point is less than a specified threshold ε . In addition to mitigating

numerical issues, this approach reduces the number of constraints in the inferred CMDP, making it easier to solve.

Appendix C.2 discusses both practical modifications in more detail. Importantly, if we choose a small enough ε , these modifications do not lose any theoretical guarantees.

5.4 EXPERIMENTS

We present experiments in two domains: (1) tabular environments, where we can solve MDPs and CMDPs exactly (Section 5.4.3); and (2) a driving simulation, where we investigate the scalability of CoCoRL in a more complex, continuous, environment (Section 5.4.4). Our experiments assess the safety and performance of CoCoRL and compare it to IRL-based baselines (Section 5.4.1). In Appendix C.3, we provide additional details about the experimental setup.

5.4.1 IRL-Based Constraint Learning Baselines

To the best of our knowledge, CoCoRL is the first algorithm that learns constraints from demonstrations with unknown rewards. Thus it is not immediately clear which baselines to compare it to. Nevertheless, we explore several IRL-based approaches that could serve as natural starting points for solving the constraint inference problem. Specifically, we consider the following three ways of applying IRL:

- *Average IRL* infers a separate reward function \hat{r}_i for each demonstration π_i^* and averages the inferred rewards before combining them with an evaluation reward $r_{\text{eval}} + \sum_i \hat{r}_i/k$.
- *Shared Reward IRL* parameterizes the inferred rewards as $\hat{r}_i + \hat{c}$, where \hat{c} is shared among all demonstrations. The parameters for both the inferred rewards and the shared constraint penalty are learned simultaneously.
- *Known Reward IRL* parameterizes the inferred reward as $r_i + \hat{c}$, where r_i is known, and only a shared constraint penalty is

learned. Note that *Known Reward IRL* has full access to the demonstrations’ rewards that the other methods do not need.

Each of these approaches can be implemented with any standard IRL algorithm. We primarily use Maximum Entropy IRL [47], but we also test Maximum Margin IRL [46] in tabular environments, which yields similar results.

To adapt Maximum Entropy IRL (Section 2.4.1) to our setting, we parameterizing the reward function as $\hat{r}(s, a) = \hat{\theta}_i^T \mathbf{f}(s, a) + \hat{\phi}^T \mathbf{f}(s, a)$. Here $\hat{\theta}_i$ parameterizes the “reward” component for each expert policy, and $\hat{\phi}$ parameterizes the shared “constraint” component.

Algorithm 4 shows the modified maximum entropy IRL algorithm. Instead of doing gradient updates on a single parameter, we do gradient updates for both $\hat{\theta}_i$ and $\hat{\phi}$ with two different learning rates α_θ and α_ϕ . Depending on initialization and learning rate, Algorithm 4 implements all of our IRL-based baselines:

- Average IRL: $\alpha_\theta > 0, \alpha_\phi = 0, \hat{\theta}_0 = 0, \hat{\phi}_0 = 0$
- Shared Reward IRL: $\alpha_\theta > 0, \alpha_\phi > 0, \hat{\theta}_0 = 0, \hat{\phi}_0 = 0$
- Known Reward IRL: $\alpha_\theta = 0, \alpha_\phi > 0, \hat{\theta}_i = \theta_i^{\text{known}}, \hat{\phi}_0 = 0$

For Known Reward IRL and Shared Reward IRL, we can apply the learned $\hat{\phi}$ to a new reward function by computing $r_{\text{eval}}(s, a) + \hat{\phi}^T \mathbf{f}(s, a)$. For Average IRL, we use $r_{\text{eval}}(s, a) + \frac{1}{k} \sum_i \hat{\theta}_i^T \mathbf{f}(s, a)$.

5.4.2 Single-State CMDPs

As a first simple test-bed, we consider a single state CMDP, where $\mathcal{S} = \{s\}$ and $\mathcal{A} = \mathbb{R}^d$. We remove the complexity of the transition dynamics by having $P(s|s, a) = 1$ for all actions. Also, we choose $\gamma = 0$, and the feature vectors are simply $\mathbf{f}(s, a) = a$. Given a reward function parameterized by $\theta \in \mathbb{R}^d$ and constraints parameterized by

Algorithm 4 Maximum Entropy IRL, adapted to constraint learning.

Require: Expert policies π_1^*, \dots, π_k^* , learning rates $\alpha_\theta, \alpha_\phi$

- 1: Initialize: $\hat{\theta}_i \leftarrow \hat{\theta}_0$ for $i = 1, \dots, k$, $\hat{\phi} \leftarrow \hat{\phi}_0$, $i \leftarrow 0$
 - 2: **while** not converged **do**
 - 3: $i \leftarrow \text{mod}(i + 1, k + 1)$
 - 4: Update policy $\hat{\pi}$ for $\hat{r}(s, a) = \hat{\theta}_i^T \mathbf{f}(s, a) + \hat{\phi}^T \mathbf{f}(s, a)$
 - 5: Compute the gradient: $\nabla = \mathbf{f}(\pi_i^*) - \mathbf{f}(\hat{\pi})$
 - 6: Update weights: $\hat{\theta}_i \leftarrow \hat{\theta}_i + \alpha_\theta \nabla$, $\hat{\phi} \leftarrow \hat{\phi} + \alpha_\phi \nabla$
 - 7: **return** $\hat{\theta}_1, \dots, \hat{\theta}_k, \hat{\phi}$
-

$\phi_1, \dots, \phi_n \in \mathbb{R}^d$ and thresholds ξ_1, \dots, ξ_n , we can find the best action by solving the linear program

$$a^* \in \underset{\phi_1^T a \leq \xi_1, \dots, \phi_n^T a \leq \xi_n}{\text{argmax}} \quad \theta^T a.$$

To generate demonstrations, we sample both the rewards $\theta_1, \dots, \theta_k$ and the constraints ϕ_1, \dots, ϕ_n from the unit sphere. We set all thresholds to $\xi_j = 1$. The constraints are shared between all demonstrations.

Our results confirm that CoCoRL guarantees safety. We never get an unsafe solution when optimizing over the inferred safe set. Figure 5.3 shows the reward we achieve as a function of the number of demonstrations for different dimensions d and different numbers of true constraints n . CoCoRL achieves good performance with a small number of samples and eventually approaches optimality. The number of samples depends on the dimensionality, as Theorem 5.3.3 suggests. Empirically, the sample complexity depends less on the number of constraints in the true environment, consistent with Theorem 5.3.3.

5.4.3 Tabular Environments

Next, we consider tabular CMDPs which we can still solve using linear programming (see Section 2.3.3). We construct a set of Gridworld environments that are structured as a 2D grid of cells. Each cell represents a state, and the agent can move left, right, up, down, or stay

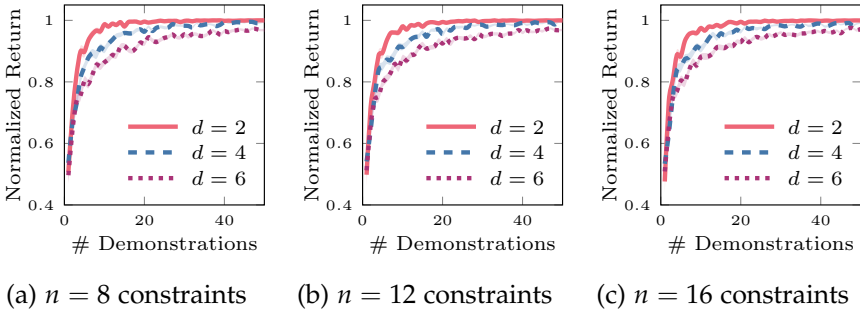


FIGURE 5.3: Return achieved by CoCoRL in single-state CMDPs for different dimensions d and numbers of constraints n . We plot the mean and standard errors over 100 random seeds, although the standard errors are nearly 0 everywhere. As expected, we need more demonstrations to approximate the true safe set well in higher dimensional feature spaces. There is no noticeable difference in sample complexity for different numbers of constraints. All solutions returned by CoCoRL are safe.

in the same cell. The environments have goal cells and limited cells. When the agent reaches a goal cell, it receives a reward; however, the constraints restrict how often the agent can visit the limited cells.

To introduce stochasticity, there is a fixed probability p that the agent executes a random action instead of the intended one. The positions of the goal and limited cells are sampled uniformly at random. The reward and cost values are sampled uniformly from the interval $[0, 1]$. The thresholds are also sampled uniformly, while we ensure feasibility via rejection sampling.

We perform three experiments to evaluate the transfer of learned constraints. First, we evaluate the learned constraints in the same environment and with the same reward distribution that they were learned from. Second, we change the reward distribution by sampling a new set of potential goals while keeping the constraints fixed. Third, we use a deterministic Gridworld ($p = 0$) during training and a stochastic one ($p = 0.2$) during evaluation.

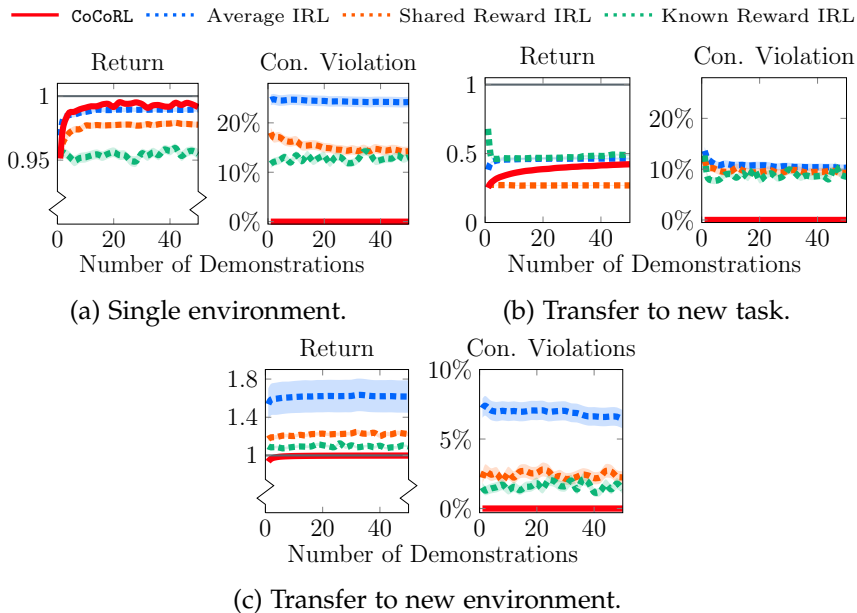


FIGURE 5.4: Experimental results in Gridworld environments. We consider three settings: (a) no constraint transfer, (b) transferring constraints to new goals in the same grid, and (c) transferring constraints to a new Gridworld with the same structure but different transition dynamics. For each setting, we measure the normalized policy return (**higher is better**), and the constraint violation (**lower is better**). CoCoRL consistently returns safe solutions, outperforming the IRL-based methods that generally perform worse and are unsafe. The plots show mean and standard errors over 100 random seeds. A return greater than 1 indicates a solution that surpasses the best safe policy, implying a violation of safety constraints. The IRL methods use maximum entropy IRL.

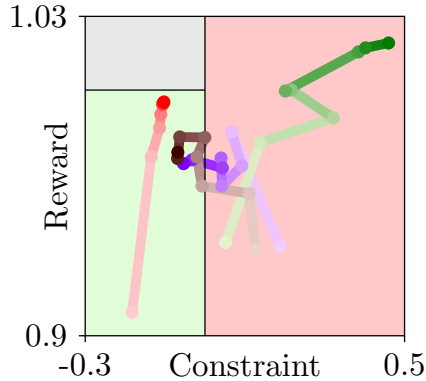


FIGURE 5.5: Illustration of safety and reward of **CoCoRL**, **Average IRL**, **Shared Reward IRL**, and **Known Reward IRL** in the Gridworld without constraint transfer. We show the 95th percentile of the constraint and the corresponding reward. The lines are colored from light to dark to indicate increasing number of demonstrations. The green region is safe, the red region is unsafe, and the grey region would be safe but is unattainable. **CoCoRL** quickly reaches a safe and near-optimal solution. **Known Reward IRL** achieves a high reward but is unsafe. **Shared Reward IRL** and **Average IRL** are first unsafe and then safe but suboptimal.

Figure 5.4 shows results of our experiments in 3×3 Gridworlds. CoCoRL consistently returns safe policies and converges to the best safe solution as more demonstrations are provided. In contrast, none of the IRL-based methods learns safe policies. In the on-distribution evaluation (Figure 5.4a), the IRL methods come closest to the desired safe policy but still cause constraint violations.

Figure 5.5 provides an illustration of how the different methods trade off reward and safety. While CoCoRL always remains safe, the IRL baselines only achieve higher rewards by violating constraints.

5.4.4 *Driving Environment*

To explore a more practical and interesting environment, we consider `highway-env`, a 2D driving simulator [118]. We focus on a four-way intersection, as depicted in Figure 5.1. The agent drives towards the intersection and can turn left, right, or go straight. The environment is continuous, but it has a discrete high-level action space: the agent can accelerate, decelerate, and make turns at the intersection. There are three possible goals for the agent: turning left, turning right, or going straight. Additionally, the reward functions contain different driving preferences related to velocity and heading angle. Rewards and constraints are defined as a linear function of a state feature vector including goal indicators, vehicle position, velocity, heading, and indicators for unsafe events including crashing, leaving the street, and violating the speed limit.

For constrained policy optimization, we use a constrained cross-entropy method (CEM) to optimize a parametric driving controller (see Appendix C.3.2 and [119]). We collect synthetic demonstrations that are optimized for different reward functions under fixed constraints.

Similar to the previous experiments, we evaluate the learned constraints in three settings: (1) evaluation on the same task distribution and environment; (2) transfer to a new task; and (3) transfer to a modified environment. To transfer to a new task, we learn the constraints from trajectories only involving left and right turns at the intersection, but evaluate them on a reward function that rewards going straight (the situation illustrated in Figure 5.1). For the transfer to a modified environment, we infer constraints in an environment with very defensive drivers and evaluate them with very aggressive drivers. Additional details on the parameterization of other drivers can be found in Appendix C.3.

Empirically, we observe that the CEM occasionally fails to find a feasible policy when the safe set S is small. This is likely due to the CEM relying heavily on random exploration to discover an initial feasible solution. In situations where we cannot find a solution

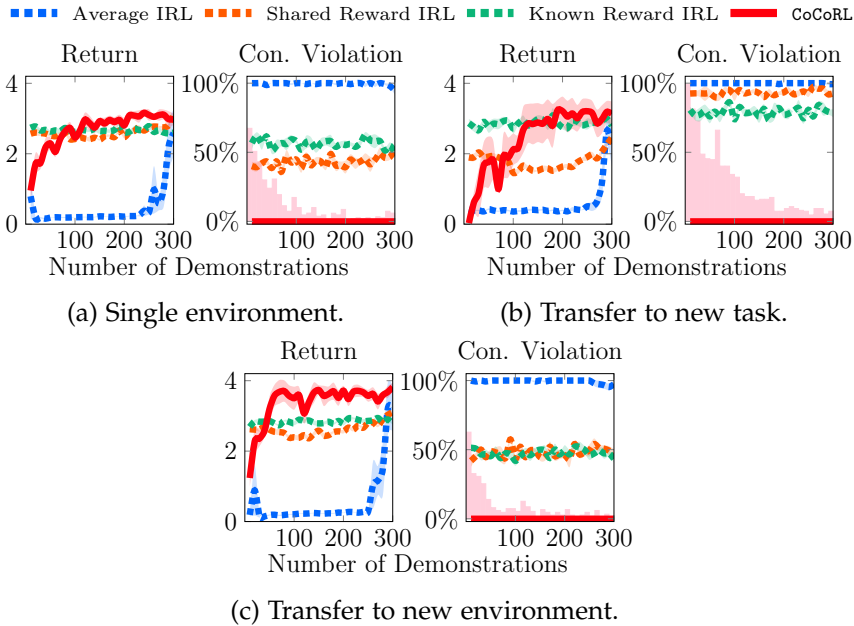


FIGURE 5.6: Experimental results in the *highway-env* intersection environment, depicted in the form of return and constraint violation plots, similar to Figure 5.4. All plots show mean and standard error over 5 random seeds with a fixed set of evaluation rewards for each setting. In all settings, CoCoRL consistently returns safe policies, as indicated by the low constraint violation values. However, there are instances where CoCoRL falls back to providing a default safe solution when the policy optimizer fails to find a feasible solution within the safe set S . The frequency of falling back to the default solution is shown by the bars in the constraint violation plots (◻). All baselines fail to return safe solutions and only outperform CoCoRL by violating constraints. Average IRL performs much worse than the other two methods, likely because it assumes that we can average the inferred rewards to remove the reward’s goal component, which is not true in this environment.

in S , we return a “default” safe controller that remains safe but achieves a return of 0 because it just stands still before the intersection. Importantly, we empirically confirm that whenever we find a solution in S , it is truly safe (i.e., in \mathcal{F}).

Figure 5.6 presents results from the driving experiments. CoCoRL consistently guarantees safety, and with a larger enough number of demonstrations (~ 200), CoCoRL returns high return policies. In contrast, Maximum Entropy IRL produces policies with decent performance in terms of return but they tend to be unsafe. Although the IRL solutions usually avoid crashes (which also have low rewards), they disregard constraints in conflict with the reward function, such as the speed limits and keeping distance to other vehicles. IRL results in substantially more frequent constraint violations when transferring the learned constraint penalty to a new reward. This is likely because the magnitude of the constraint penalty is no longer correct. CoCoRL does not suffer from this issue, and transferring the inferred constraints still results in safe and well-performing policies.

5.5 CONCLUSION

In this chapter, we introduced CoCoRL, a novel algorithm for inferring shared constraints from demonstrations with unknown rewards. Theoretical and empirical results show that CoCoRL guarantees safety and achieves strong performance, even when transferring constraints to new tasks or environments.

CoCoRL can use unlabeled demonstrations, which are often easier to obtain, particularly in domains such as robotics or autonomous driving. By learning safety constraints, CoCoRL offers a more sample-efficient and safer alternative to pure reward learning.

However, our setup has some limitations. We assume that all demonstrations are safe, which may not always be practical. Learning from potentially unsafe demonstrations requires improved models of human demonstrations or additional (safety) labels. Our second assumption is access to a feature representation in non-tabular en-

vironments. As we scale CoCoRL to more complex applications, the ability to *learn* features that capture crucial safety aspects will become increasingly important.

In contrast to Chapters 3 and 4, we did not focus on active learning in the present chapter. Rather, CoCoRL infers constraints from a fixed set of safe demonstrations. Clearly, active learning would also be useful for learning constraint models efficiently. Therefore, the next chapter develops an active learning algorithm for learning constraints from trajectories labelled as safe or unsafe.

INTERACTIVELY LEARNING CONSTRAINTS

In the previous chapter, we argued that constraints can be a natural representation of human preferences in many situations. This chapter explores this idea further and develops an algorithm for actively learning about unknown constraints.

Often, it is natural to decompose a task into a goal and a set of constraints, where the goal is easy to specify, and the constraints are difficult to specify. For example, a car manufacturer might have a set of possible controllers for a car to choose from that perform a specific task, such as reaching a target destination as quickly as possible. The ideal controller achieves this task well and drives safely and comfortably. Whereas the objective – travel time – is easy to specify as a reward function, the constraints – perceived safety and comfort – may require feedback from human drivers and passengers. The manufacturer aims to find the best, safe controller with as little human feedback as possible. If the controllers are evaluated in a simulation, it is acceptable to evaluate an unsafe controller during training; however, the constraints have to be satisfied during deployment.

We focus on similar situations where the decision-making problem is naturally characterized by an easy-to-evaluate part (the reward) and an expensive-to-evaluate part (the constraint). Hence, in this chapter, we study *learning about unknown, expensive-to-evaluate constraints*. Similar to the setup for reward learning in Chapter 3, we propose a two-phase approach to learning unknown constraints. In the first phase, we learn to estimate the expensive-to-evaluate constraint function well enough to solve the constraint optimization problem. In the second phase, we recommend a solution which might involve running a constrained RL algorithm. Constraint violations are allowed in the first phase, but the final recommendation has to satisfy the constraints.

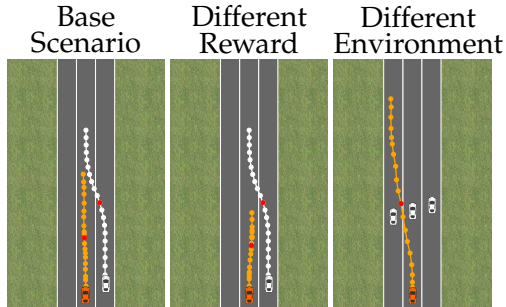
Concretely, in this chapter:

- We formalize learning about unknown constraints to find the best constrained solution as a novel linear bandit problem, we call *constrained linear best-arm identification* (CBAI; Section 6.2).
- We derive an instance-dependent sample complexity lower bound for the CBAI problem (Section 6.2.1).
- We propose *Adaptive Constraint Learning* (ACOL) to solve the CBAI problem and show that ACOL’s sample complexity almost matches the instance-dependent lower bound (Section 6.2.3).
- We test ACOL in synthetic CBAI instance and find that ACOL gets close to the performance of an oracle solution that has access to the true constraint function while outperforming a range of baselines (Section 6.3.1).
- As a concrete application, we again consider learning driving behavior in a simulation, where the constraints represent human preferences about driving behavior (Section 6.3.3). We demonstrate empirically that ACOL can learn these constraints and propose heuristic variants of the algorithm that empirically improve sample efficiency.

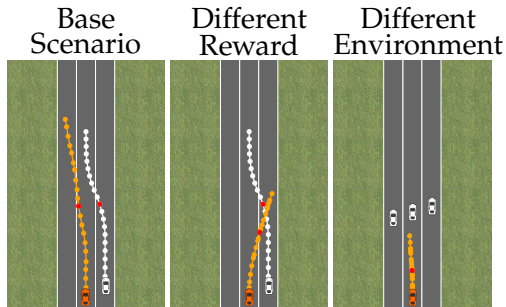
Beyond this, we further compare learning constraints to learning rewards. We observe that constraints are more robust to changes in the environment and can be transferred to selecting controllers for different goals, in contrast to encoding the constraints as a penalty in the reward function (see Figure 6.1 for an illustration and Section 6.3.3 for quantitative results).

The contents of this chapter are based on:

D. Lindner, S. Tschitschek, K. Hofmann, and A. Krause, “Interactively learning preference constraints in linear bandits”, in *International Conference on Machine Learning (ICML)*, 2022.



(a) Reward Penalty



(b) Constraint

FIGURE 6.1: Illustration of the added robustness when using constraints. In the “base” scenario, the orange car should drive at velocity v , which we encode as a reward. We model other rules, like “usually drive inside a lane” or “don’t get too close to other cars”, either as a reward penalty (a) or as a constraint (b). In the “different reward” scenario, the reward is to pull over to the right of the road instead of maintaining v . If we keep the same reward penalty, the agent fails, whereas the constrained agent completes the new task safely without any tuning. In the “different environment” scenario, the goal of driving at velocity v remains, but the road is now blocked. Here, the penalized agent trades off the penalty with achieving a high reward and tries to pass through the cars blocking the road. The constraint formulation does not allow violating a constraint such as “don’t get too close to other cars,” and the constraint agent stops before crashing into other cars.

6.1 RELATED WORK

Our problem formalization as a linear multi-armed bandit (MAB) *best-arm identification* problem [120] is similar to Soare *et al.* [121] in the unconstrained setting, but focused on learning constraints. Learning constraints is similar to actively classifying arms as “feasible” or “infeasible”; but, in contrast to typical active learning [21], we do not need to classify all arms. Instead, we only want to find the best feasible arm, which requires fewer samples than classifying all arms.

Much prior work on constraints in MABs considers other notions of constraints than we do. For example, constraints holding in expectation rather than with high probability [122], or constraints in the form of a lower bound (threshold) on the reward [123–126].

Amani *et al.* [127] and Moradipari *et al.* [128] consider a linear bandit with a separate (linear) constraint function. Both differ from our work in three important ways: (1) they assume an unknown reward function, whereas we assume the reward to be known; (2) they focus on cumulative regret minimization, whereas we focus on best-arm identification; and, (3) they require the constraints to be satisfied during exploration whereas we only require them to be satisfied for the final recommendation. These works adapt bandit algorithms based on upper confidence bounds [127] or Thompson sampling [128] to minimize regret in the constraint setting. To enable safe exploration, they need a convex and compact set of arms; we do not require this assumption.

Wang *et al.* [129] also study best-arm identification with linear constraints. In contrast to our work, they assume unknown rewards and focus on safety constraints that must be satisfied during exploration. To make this possible, they need to make more assumptions about the structure of the set of arms. In particular, they assume that the agent can only query in each dimension independently. Because of this, their algorithm cannot be applied to our setting without significant changes.

Our algorithm is conceptually similar to other bandit algorithms based on the principle of eliminating sub-optimal arms step-by-step.

Our theoretical analysis employs similar tools as those used for best-arm-identification in unconstrained linear bandits [66, 121].

Some works in Bayesian optimization (BO) also study the problem of exploring to find the best constrained solution to a problem with an expensive-to-evaluate constraint function. Common approaches heuristically extend BO methods to incorporate an unknown constraint function [130–132]. In contrast to this line of work, we obtain sample complexity guarantees by focusing on linear constraint functions. Similar to the bandit literature, most work on BO with constraints focuses on the setting where safety constraints must hold during exploration (e.g., [133]).

6.2 LINEAR CONSTRAINED BEST-ARM IDENTIFICATION

We seek to find the best constrained solution from a discrete set of arms represented by feature vectors $x \in \mathcal{X} \subset \mathbb{R}^d$. We assume that both the known reward function and the unknown constraint function are linear in x .

Definition 6.2.1. A *constrained linear best-arm identification (CBAI)* problem $\nu = (\mathcal{X}, \theta, \phi, \tau)$ consists of a finite set of arms $\mathcal{X} \subset \mathbb{R}^d$, a reward parameter $\theta \in \mathbb{R}^d$, a constraint parameter $\phi \in \mathbb{R}^d$, and threshold $\tau \in \mathbb{R}$. The agent knows \mathcal{X} and θ , but not ϕ and τ . In each iteration, the agent selects an arm $x \in \mathcal{X}$ and observes $\phi^T x + \eta_x$, where η_x is sub-Gaussian noise. The agent’s goal is to identify a constrained optimal arm

$$x^* \in \operatorname{argmax}_{x \in \mathcal{X}, \phi^T x \leq \tau} \theta^T x$$

within as few iterations as possible.

In our initial example, \mathcal{X} contains all potential driving controllers. $\theta^T x$ encodes the time controller x needs to the destination, which the agent knows and wants to minimize. $\phi^T x$ encodes the unknown driving preferences, which the agent must infer from as few experiments as possible. In the following, we assume *w.l.o.g.* $\tau = 0$ but

generalization to $\tau \neq 0$ is straightforward. If τ is unknown, we can model it as a constant shift in the constraint. To simplify notation, we omit τ and talk about a CBAI problem $\nu = (\mathcal{X}, \theta, \phi)$.

6.2.1 Lower Bounds

We first provide a lower bound on the sample complexity of solving a given CBAI problem. The following theorem states how many samples are necessary to distinguish a given CBAI instance from the closest instance with a different solution, which is necessary to solve an instance.

Theorem 6.2.1 (CBAI lower bound). Assume $\eta_x \sim \mathcal{N}(0, 1)$ for all $x \in \mathcal{X}$. For any CBAI problem $\nu = (\mathcal{X}, \theta, \phi)$, there exists another CBAI problem $\nu' = (\mathcal{X}, \theta, \phi')$ with the same set of actions \mathcal{X} and reward parameter θ but a different constraint parameter and optimal arm, such that the expected number of iterations τ needed by any allocation strategy that can distinguish between ν and ν' with probability at least $1 - \delta$ is lower bounded as

$$\mathbb{E}[\tau] \geq 2 \log \left(\frac{1}{2.4\delta} \right) \max_{x \in \mathcal{X}_\theta^\geq(x_\nu^*)} \frac{\|x\|_{A_\lambda^{-1}}^2}{(\phi^T x)^2},$$

where λ is a probability distribution over arms of the allocation strategy, i.e., $\lambda(x)$ is the probability that it pulls arm x , $A_\lambda = \sum_x \lambda(x) x x^T$ is the design matrix, and $\mathcal{X}_\theta^\geq(x_\nu^*) = \{x' \in \mathcal{X} \mid \theta^T x' \geq \theta^T x_\nu^*\}$ is the set of all arms with reward no less than x_ν^* , the optimal arm for ν .

The proof in Appendix D.1.1 uses a strategy similar to that for lower bounds for standard linear bandits [66, 121, 134]. We consider the log-likelihood ratio of making a series of observations in instance ν compared to ν' and consider how we can choose ν' to have a different solution but a small log-likelihood ratio, i.e., the agent makes similar observations as if it was in ν . In contrast to the standard linear bandit case, we need to carefully reason about the constraints when ensuring that ν' has a different solution than ν . We distinguish the

case that the solution of v is infeasible in v' and the case that an arm with larger reward is feasible in v' but not v . Reasoning about these two cases yields the result.

Our lower bound has a similar form as those for best-arm identification in linear bandits [121]. In particular, we have the same uncertainty term in the numerator. Instead of a suboptimality gap in the denominator, we get the distance to the constraint boundary: the problem is harder if arms are closer to the constraint boundary. However, our maximization is over individual arms instead of directions, i.e., pairs of arms, and the set $\mathcal{X}_\theta^{\geq}(x_\nu^*)$ does not appear in the standard linear bandit case.

We want to characterize the sample complexity of different algorithms for solving the CBAI problem. To this end, let us define the sample complexity of a given problem instance using the lower bound we just derived.

Definition 6.2.2 (CBAI sample complexity). We define the *sample complexity* of a CBAI problem v as

$$H_{\text{CLB}}(v) := \min_{\lambda} \max_{x \in \mathcal{X}_\theta^{\geq}(x_\nu^*)} \frac{\|x\|_{A_\lambda^{-1}}^2}{(\phi^T x)^2}$$

This describes the best sample complexity that any algorithm can achieve on CBAI problem v . It will also be helpful to have a worst-case upper bound on $H_{\text{CLB}}(v)$ as a point of comparison, which the next proposition provides.

Proposition 6.2.1. For any CBAI problem v , we have $H_{\text{CLB}}(v) \leq d/(C_{\min}^+)^2$, where $C_{\min}^+ = \min_{x \in \mathcal{X}} |\phi^T x|$. This bound is tight, i.e, there is an instance v , such that we have $H_{\text{CLB}}(v) = d/(C_{\min}^+)^2$.

This results indicates that a CBAI problem is harder if it has a larger dimension d , or if the distance of the arm that is closest to the constraint boundary (C_{\min}^+) is smaller. This worst case bound corresponds to situations where all arms are linearly independent and pulling one arm does not provide any information about any other arm.

ORACLE SOLUTION. We can make the definition of sample complexity more concrete by considering an oracle solution that has access to the true constraint value to select which arms to query. The oracle selects arms by explicitly minimizing H_{CLB} :

$$\lambda^* \in \operatorname{argmin}_{\lambda} \max_{x \in \mathcal{X}_{\theta}^{\geq}(x_{\nu}^*)} \frac{\|x\|_{A_{\lambda}^{-1}}^2}{(\phi^T x)^2}.$$

The oracle prefers arms with high uncertainty (large $\|x\|_{A_{\lambda}^{-1}}^2$) and arms close to the constraint boundary (small $(\phi^T x)^2$). Moreover, it focuses on reducing the uncertainty about arms that have higher reward than the true optimal arm (arms in $\mathcal{X}_{\theta}^{\geq}(x_{\nu}^*)$). As expected, this algorithm matches the sample complexity lower bound, i.e., it is instance-optimal apart from logarithmic factors.

Theorem 6.2.2 (Oracle sample complexity). The oracle algorithm finds the optimal solution to a constrained linear best-arm identification problem $\nu = (\mathcal{X}, \theta, \phi)$ within $N \propto H_{\text{CLB}}(\nu)$ with probability at least $1 - \delta$.

6.2.2 Confidence Intervals for Linear Regression

Our algorithms rely on high probability confidence intervals on the linear constraints constructed from observations. Hence, let us briefly review how to construct such confidence intervals from observations with sub-Gaussian noise.

Suppose, an algorithm queried a sequence of arms $\mathbf{x}_t = (x_1, \dots, x_t)$. For a given x_i , it observed $\tilde{y}_i = \phi^T x_i + \eta_{x_i}$, where ϕ is the true constraint parameter, and η_{x_i} is sub-Gaussian noise. We now aim to find confidence intervals such that $\phi^T x \in [l_{\phi}^t(x), u_{\phi}^t(x)]$ with probability at least $1 - \delta$, where $l_{\phi}^t(x) = \hat{\phi}^T x - \sqrt{\beta_t} \|x\|_{A_{x_t}^{-1}}$ and $u_{\phi}^t(x) = \hat{\phi}^T x + \sqrt{\beta_t} \|x\|_{A_{x_t}^{-1}}$. Based on these confidence intervals, we can decide whether a given arm is likely feasible or not.

If the queries follow a distribution that does not depend on the observations, it is straightforward to derive confidence intervals (e.g., see Chapter 20 in Lattimore and Szepesvári [24]).

Proposition 6.2.2. Let $\mathbf{x}_t = (x_1, \dots, x_t)$ be a sequence of arms from a fixed allocation for which we have observed $\phi^T x_i + \eta_{x_i}$ where η_{x_i} is independent sub-Gaussian noise. If we estimate $\hat{\phi}$ from the observations using least-squares regression and choose $\beta_t = \sqrt{2 \log(|\mathcal{X}|/\delta)}$ then we have $P(\exists x \in \mathcal{X} : \phi^T x \notin [l_\phi^t(x), u_\phi^t(x)]) \leq \delta$.

However, in sequential decision-making we usually want to adapt our strategy to previous observations. In this case, we need to be more careful in constructing confidence intervals, as observed by Abbasi-Yadkori *et al.* [135]. Unfortunately, the resulting confidence intervals are weaker than those for static allocations by a factor of \sqrt{d} .

Proposition 6.2.3 (Theorem 2 by Abbasi-Yadkori *et al.* [135]). Let $\mathbf{x}_t = (x_1, \dots, x_t)$ be a sequence of points selected with a possibly adaptive strategy for which we have observed $\phi^T x_i + \eta_{x_i}$ where η_{x_i} is independent sub-Gaussian noise. Assume $\|\phi\|_2 \leq S$ and $\|x\|_2 \leq L$ for all $x \in \mathcal{X}$. If we estimate $\hat{\phi}$ from the observations using least-squares regression, then for every $x \in \mathbb{R}^d$ and for all $t \geq 0$: $P(\exists x \in \mathcal{X} : \phi^T x \notin [l_\phi^t(x), u_\phi^t(x)]) \leq \delta$ with $\beta_t = \sqrt{d \log((1 + tL^2/\lambda)/\delta)} + \sqrt{\lambda S}$.

6.2.3 Algorithms With Static Confidence Intervals

To design an algorithm for solving CBAI problems, we need to decide (1) which arms to pull during exploration and (2) when we can stop the algorithm and return the correct arm with high probability. First, let us address the second question and then get back to the first one.

STOPPING CONDITION. Using the past observations, we can define confidence intervals for the constraint value of each arm. Let $l_\phi^t(x)$ and $u_\phi^t(x)$ be such that we know with high probability (w.h.p.) $\phi^T x \in [l_\phi^t(x), u_\phi^t(x)]$. Now we can also determine w.h.p. that all arms

with $l_\phi^t(x) > 0$ are infeasible, and all arms with $u_\phi^t(x) \leq 0$ are feasible. Moreover, we can identify suboptimal arms by considering $\bar{r} = \max_{u_\phi^t(x) \leq 0} \theta^T x$. The solution to this optimization problem are the highest-reward arms that are feasible w.h.p. Therefore, all arms with reward less than \bar{r} are clearly suboptimal. Combining these observations, we can define a set of arms that we are uncertain about, i.e., that could still be optimal:

$$\mathcal{U}_t = \{x \in \mathcal{X} \mid l_\phi^t(x) \leq 0 \text{ and } u_\phi^t(x) > 0 \text{ and } \theta^T x > \bar{r}\}$$

Note, that if \mathcal{U}_t is empty, we can stop and return an arm

$$x^* \in \operatorname{argmax}_{u_\phi^t(x) \leq 0} \theta^T x.$$

This arm will be optimal w.h.p.

ARM SELECTION CRITERION. In each iteration, we have to decide which arm to pull. We could, e.g., combine the above stopping condition with querying uniformly random arms. This algorithm would return the correct optimal arm with high probability. However, random querying will usually not be the most sample efficient approach. Another natural approach is to select the arms that we are most uncertain about, which is sometimes called *uncertainty sampling*. We could, e.g., choose a fixed allocation

$$\lambda^G \in \operatorname{argmin}_\lambda \max_{x \in \mathcal{X}} \|x\|_{A_\lambda^{-1}}.$$

This approach is also called *G-Allocation* in the experimental design literature. The following theorem shows that G-Allocation achieves sample complexity on order $d/C_{\min}^+{}^2$, so it matches the worst-case lower bound in Proposition 6.2.1.

Theorem 6.2.3 (G-Allocation sample complexity). G-Allocation finds the optimal arm within $N \propto d/C_{\min}^+{}^2$ iterations with probability at least $1 - \delta$.

However, we can do better by focusing on arms that we *cannot yet exclude as being certainly feasible, infeasible, or suboptimal*. Concretely, we modify G-Allocation to reduce uncertainty only about arms in \mathcal{U}_t :

$$\lambda^{\text{ACOL}} \in \operatorname{argmin}_{\lambda} \max_{x \in \mathcal{U}_t} \|x\|_{A_{\lambda}^{-1}}$$

ROUNDING. All algorithms implementing a static allocation require a rounding procedure to translate an allocation λ into a finite sequence of arms x_1, \dots, x_n . The experimental design literature provides various efficient, ε -approximate rounding procedures. We use a standard procedure described in Chapter 12 of Friedrich [136].

ADAPTIVE CONSTRAINT LEARNING (ACOL). Algorithm 5 shows the full algorithm we call *Adaptive Constraint Learning* (ACOL). In each round t , the algorithm selects arms to reduce the uncertainty about arms in \mathcal{U}_t , then updates \mathcal{U}_t , and decides if it can stop and return a recommendation. The round length N_t is chosen carefully to allow us to provide a tight sample complexity result. The following theorem – the main theoretical result of this chapter – establishes that ACOL returns the correct solution to any CBAI problem and provides an upper bound on the number of samples necessary.

Theorem 6.2.4 (ACOL sample complexity). Assume Algorithm 5 is implemented with an ε -approximate rounding strategy. Then, after N iterations the algorithm returns an optimal arm with probability at least $1 - \delta$, and we have:

$$\begin{aligned} N &\leq 8 \log \left(\frac{|\mathcal{X}| \bar{t}^2}{\delta^2} \right) (1 + \varepsilon) \sum_{t=1}^{\bar{t}} \min_{\lambda} \max_{x \in \mathcal{U}_t} \frac{\|x\|_{A_{\lambda}^{-1}}^2}{(\phi^T x)^2} + \bar{t} \\ &\leq 8 \log \left(\frac{|\mathcal{X}| \bar{t}^2}{\delta^2} \right) (1 + \varepsilon) \bar{t} \bar{H}_{\text{CLB}}(\nu) + \bar{t} \end{aligned}$$

where

$$\begin{aligned} \bar{H}_{\text{CLB}}(\nu) &= \min_{\lambda} \max_{x \in \mathcal{X}} \|x\|_{A_{\lambda}^{-1}}^2 / (\phi^T x)^2, \\ \bar{t} &= \lceil -\log_2 C_{\min}^+ \rceil. \end{aligned}$$

Algorithm 5 Adaptive Constraint Learning (ACOL).

Require: significance δ

- 1: $\mathcal{U}_1 \leftarrow \mathcal{X}$ (uncertain arms)
 - 2: $S_1 \leftarrow \{\}$ (feasible arms)
 - 3: $t \leftarrow 1$ (round)
 - 4: **while** $\mathcal{U}_t \neq \{\}$ **do**
 - 5: $\delta_t \leftarrow \delta^2/t^2$
 - 6: $\lambda_t^* \leftarrow \operatorname{argmin}_\lambda \max_{x \in \mathcal{U}_t} \|x\|_{A_\lambda^{-1}}^2$
 - 7: $\rho_t^* \leftarrow \min_\lambda \max_{x \in \mathcal{U}_t} \|x\|_{A_\lambda^{-1}}^2$
 - 8: $N_t \leftarrow \max \left\{ \left\lceil 2^{2t+3} \log \left(\frac{|\mathcal{X}|}{\delta_t} \right) (1 + \varepsilon) \rho_t^* \right\rceil, r(\varepsilon) \right\}$
 - 9: $\mathbf{x}_{N_t} \leftarrow \operatorname{Round}(\lambda_t^*, N_t)$
 - 10: Pull arms x_1, \dots, x_{N_t} and observe constraint values
 - 11: $t \leftarrow t + 1$
 - 12: Update $\hat{\phi}_t$ and A based on new data
 - 13: $l_\phi^t(x) \leftarrow \hat{\phi}_t^T x - \sqrt{\beta_t} \|x\|_{A^{-1}}$ for all arms x
 - 14: $u_\phi^t(x) \leftarrow \hat{\phi}_t^T x + \sqrt{\beta_t} \|x\|_{A^{-1}}$ for all arms x
 - 15: $S_t \leftarrow S_{t-1} \cup \{x | u_\phi^t(x) \leq 0\}$
 - 16: $\bar{r} \leftarrow \max_{x \in S_t} \theta^T x$
 - 17: $\mathcal{U}_t \leftarrow \mathcal{U}_{t-1} \setminus \{x | l_\phi^t(x) > 0\} \setminus \{x | u_\phi^t(x) \leq 0\} \setminus \{x | \theta^T x < \bar{r}\}$
 - 18: **return** $x^* \in \operatorname{argmax}_{x \in S_t} \theta^T x$
-

Moreover, $\bar{H}_{\text{CLB}}(v) \leq d/(C_{\min}^+)^2$.

We prove the theorem in Appendix D.1. The key step uses Proposition 6.2.2 to show that the confidence intervals shrink exponentially. This implies that in a logarithmic number of rounds, the largest confidence interval will be less than C_{\min}^+ ; and once this is the case, \mathcal{U}_t is empty and the algorithm returns the correct solution. Combining this with the round lengths of N_t allows us to prove the result.

The sample complexity of ACOL is of order $\bar{H}_{\text{CLB}}(v)$, except for logarithmic factors. Also, we show that $\bar{H}_{\text{CLB}} \leq d/(C_{\min}^+)^2$, so the

bound matches the lower bound of Proposition 6.2.1 for worst-case instances, but it is much tighter for benign instances. In particular, the bound in Theorem 6.2.4 contains the same min-max problem as the instance dependent sample complexity $H_{\text{CLB}}(\nu)$, only with the maximization being over different sets, namely \mathcal{U}_t instead of $\mathcal{X}_\theta^\geq(x_v^*)$. Note, that we cannot expect a practical algorithm to only explore arms in $\mathcal{X}_\theta^\geq(x_v^*)$ because we do not know x_v^* a priori. Instead, ACOL explores in \mathcal{U}_t , a conservative estimate of $\mathcal{X}_\theta^\geq(x_v^*)$ that shrinks over time given the knowledge so far. Theorem 6.2.4 does not exactly match the instance dependent lower bound, but the difference only depends on how well \mathcal{U}_t approximates the set of relevant arms.

6.2.4 Algorithms With Adaptive Confidence Intervals

While the algorithm we just introduced comes with a strong sample complexity guarantee, it is impractical in various ways, primarily because of the round-based structure. In particular, the algorithm requires a rounding procedure to determine a sequence of arms; it then follows this sequence for a predefined round length and can not stop before finishing a round. Also, in between rounds, the algorithm discards all previously made observations, which is necessary to apply Proposition 6.2.2.

Next, we present an alternative version of this algorithm that uses the adaptive confidence intervals of Proposition 6.2.3. This allows us to remove the round-based structure in favor of a greedy algorithm that does not have the same limitation. This algorithm, which we call *Greedy Adaptive Constraint Learning* (G-ACOL), is shown in Algorithm 6. Unfortunately, for G-ACOL, we can only provide significantly weaker sample complexity guarantees; but we find it performs well empirically.

Since the adaptive confidence intervals hold for all $t > 0$ simultaneously, we can now check the stopping condition after each sample. Instead of determining a static allocation that reduces uncertainty about the uncertain arms, we now greedily select the arm to pull that

Algorithm 6 Greedy Adaptive Constraint Learning (G-ACOL).

Require: β_t, λ

- 1: initialize $\hat{S}_1(\hat{\phi}), \mathcal{U}_1 \leftarrow \mathcal{X}, S_1 \leftarrow \{\}, A \leftarrow \lambda I, t \leftarrow 1$
 - 2: **while** $\mathcal{U}_t \neq \{\}$ **do**
 - 3: $x^* \leftarrow \max_{x \in \mathcal{U}_t} \|x\|_{A^{-1}}^2$
 - 4: Pull arm x^* and observe constraint value
 - 5: $t \leftarrow t + 1, A \leftarrow A + xx^T$
 - 6: $l_\phi^t(x) \leftarrow \hat{\phi}_t^T x - \sqrt{\beta_t} \|x\|_{A^{-1}}$ for all arms x
 - 7: $u_\phi^t(x) \leftarrow \hat{\phi}_t^T x + \sqrt{\beta_t} \|x\|_{A^{-1}}$ for all arms x
 - 8: $S_t \leftarrow S_{t-1} \cup \{x | u_\phi^t(x) \leq 0\}$
 - 9: $\bar{r} \leftarrow \max_{x \in S_t} \theta^T x$
 - 10: $\mathcal{U}_t \leftarrow \mathcal{U}_{t-1} \setminus \{x | l_\phi^t(x) > 0\} \setminus \{x | u_\phi^t(x) \leq 0\}$
 - 11: $\setminus \{x | \theta^T x < \bar{r}\}$
 - 12: **return** $x^* \in \operatorname{argmax}_{x \in S_t} \theta^T x$
-

reduces uncertainty within \mathcal{U}_t the most. Thanks to Proposition 6.2.3, this algorithm still stops and returns the correct solution. However, it achieves worse sample complexity due to the additional factor of \sqrt{d} in Proposition 6.2.3.

HEURISTIC MODIFICATIONS. There is a variety of heuristic modifications that we can make to G-ACOL to improve its practical performance at the cost of losing some theoretical guarantees. First, we could use a different query rule within the set of uncertain arms, such as uniformly random querying, which reduces computational cost. Second, the β_t resulting from Proposition 6.2.3 tends to be very large. In practice, we can tune β_t to get good confidence intervals that are much smaller than the ones suggested by the theory. Third, we can turn the algorithm into an “anytime” algorithm by defining a recommendation rule, such as recommending the best arm that is certainly feasible. Then, we can stop the algorithm after an a priori unknown budget of queries and receive a best guess for the optimal arm.

6.3 EXPERIMENTS

We perform three experiments. First, in Section 6.3.1, we consider synthetic CBAI instances to evaluate ACOL and compare it to natural baselines. Additionally, we investigate the effect of various heuristic modifications to the algorithm. Second, in Section 6.3.2, we compare ACOL to algorithms that safely minimize regret. And, third, in Section 6.3.3, we consider learning constraints that represent human preferences in a simulated driving scenario. This experiment illustrates how to model preference learning problems as CBAI problems. In the driving simulation, we also demonstrate the benefits of learning constraints in terms of robustness and transferability.

We provide more details on the experiments in Appendix D.2. For all experiments we use a significance of $\delta = 0.05$ and, if not stated differently, observations have Gaussian noise with $\sigma = 0.05$.

6.3.1 Synthetic Experiments

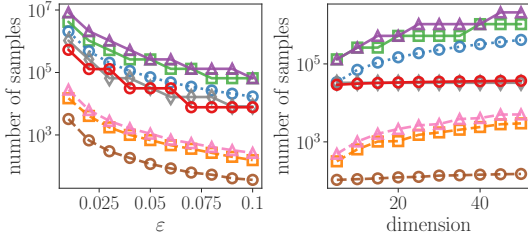
We consider two synthetic CBAI instances and a range of baselines and multiple variants of ACOL/ G-ACOL.

INSTANCE 1 – IRRELEVANT DIMENSIONS. First, we consider CBAI instances which contain a number of dimensions that are irrelevant for learning the correct constraint boundary. The problems have dimension d , and $d + 1$ arms: x_1, \dots, x_{d+1} . For each $i = 1, \dots, d - 1$, we have $x_i = \mathbf{e}_i$, whereas $x_d = (1 - \varepsilon)\mathbf{e}_d$, and $x_{d+1} = (1 + \varepsilon)\mathbf{e}_d$, for some $\varepsilon > 0$. \mathbf{e}_i denotes the i -th unit vector. The reward and constraint parameter are both $\theta = \phi = \mathbf{e}_d$. We define a threshold $\tau = 1$; hence, x_1, \dots, x_{d-1} are feasible but suboptimal, x_d is optimal and x_{d+1} is infeasible. Importantly, the arms x_1, \dots, x_{d-1} are “irrelevant” to finding the correct constraint boundary between x_d and x_{d+1} . An ideal algorithm would focus its queries primarily on x_d and x_{d+1} . We can vary the problem difficulty by changing ε (more difficult for small values), and d (more difficult for large values).

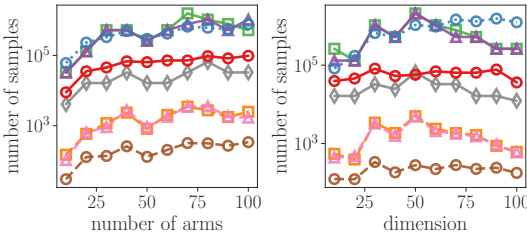
INSTANCE 2 – UNIT SPHERE. To create CBAI instances with a range of different reward and constraint functions, we sample arms x_1, \dots, x_n uniformly from a d -dimensional unit sphere. We also sample the reward parameter θ from the unit sphere. As constraint parameter, we choose $\phi = x_i - x_j$ where x_i and x_j are the two closest arms in ℓ_2 -distance. We can increase the problem difficulty by increasing the dimension d and the number of arms n .

BASELINES. We compare ACOL and G-ACOL to various baselines. The *Oracle* solution uses knowledge of the true constraint parameter to choose the best possible static allocation. In practice, we cannot implement the oracle because we do not know the constraint parameter; but, it yields a performance upper bound to which we can compare other algorithms. *G-Allocation* uses a static allocation that uniformly reduces uncertainty, whereas *Uniform* pulls all arms with equal probability. We also consider variants of these algorithms that use the adaptive confidence interval in Proposition 6.2.3. We call the adaptive version of G-Allocation *Greedy MaxVar* because it greedily selects arms with the highest uncertainty estimate from \mathcal{U}_t . We call uniform sampling with the adaptive confidence intervals *Adaptive Uniform* respectively. For all algorithms that use adaptive confidence intervals, in addition to the version using Proposition 6.2.3, we test a “tuned” version that considers β_t as a numeric hyperparameter instead (indicated by the name of the algorithms followed by *(tuned)*). We chose $\beta_t = \frac{1}{4}$, for all experiments, which we determined from minimal tuning on the “irrelevant dimensions” instance for the Greedy MaxVar algorithm. For clarity, we omit a few of the baselines that perform poorly in our plots. Appendix D.2.2 provides the full results.

RESULTS. Figure 6.2 shows our results in the synthetic CBAI instances. All algorithms find the correct solution, but their sample efficiency varies widely. From all algorithms with theoretical guarantees, the (unrealistic) oracle solution needs the fewest number of iterations, as expected. But ACOL can get close to the oracle performance and outperforms G-Allocation and uniform sampling in all



(a) Irrelevant dimensions



(b) Unit sphere

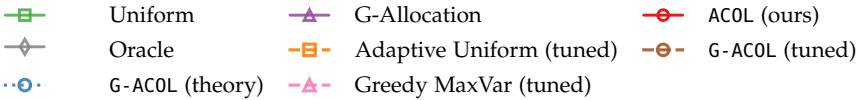
Legend:

FIGURE 6.2: We plot the median number of iterations for finding the constrained optimal solution as a function of different parameters of the problem instance. All methods return the correct constrained optimal solution. For “irrelevant dimensions”, we vary ϵ for fixed $d = 10$, and d for fixed $\epsilon = 0.05$. For “unit sphere”, we vary n for fixed $d = 30$, and d for fixed $n = 30$. Note that for “unit sphere”, the instances are randomly sampled for each random seed, whereas the “irrelevant dimensions” instance stays the same. Overall, ACOL is the most sample efficient approach of all algorithms that provide theoretical guarantees. By “tuning” β_i , we can gain several orders of magnitude in sample efficiency at the cost of theoretical guarantees. G-ACOL remains the most sample efficient among these tuned approaches.

cases. For example, if we increase the number of irrelevant dimensions in the first experiment, G-Allocation and uniform sampling need more samples to determine which dimension is relevant. In contrast, both ACOL quickly focuses on the relevant dimension. Therefore, the number of iterations it needs does not increase when adding irrelevant dimensions to the problem, similar to the oracle solution.

Methods that use adaptive confidence intervals with β_t suggested by Proposition 6.2.3 turn out to be less sample efficient than their round-based counterparts using static confidence intervals, including G-ACOL performing worse than ACOL. The reason for this is that the confidence interval in Proposition 6.2.3 is quite loose. We can heuristically choose smaller confidence intervals and consider β_t as a tunable hyperparameter. We find that we can achieve orders of magnitude better sample complexity without much tuning and still always find the correct solution. Even though this approach loses the theoretical guarantees, it could be very valuable in practical applications.

6.3.2 Comparing ACOL to Regret Minimization

To highlight the difference of our constrained linear best-arm identification setting to regret minimization with constraints, we perform an experiment to compare G-ACOL to the approaches by Amani *et al.* [127] and Moradipari *et al.* [128]. The algorithm by Amani *et al.* [127] performs UCB and the algorithm by Moradipari *et al.* [128] performs Thompson sampling, both within the set of certainly feasible arms.

We can translate both approaches to our setting with known rewards by greedily selecting arms from S_t w.r.t. their reward. Because we do not start with a known safe arm, we add an additional phase in which we select arms randomly until S_t is not empty. Let us call this approach *MaxRew-S*. As a hybrid of this approach and ACOL, we can design an algorithm that greedily select arms from \mathcal{U}_t w.r.t. their reward. Let us call this algorithm *MaxRew-U*.

Unfortunately, *MaxRew-S* gets stuck in our synthetic instances because we do not make any assumptions on the safe set such

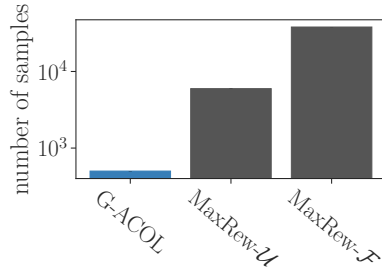


FIGURE 6.3: We compare G-ACOL to MaxRew- \mathcal{S} and MaxRew- \mathcal{U} , that adapt regret minimization approaches to the CBAI setting. We focus on a simple 1-dimensional problem, where we ensure the set of feasible arms is connected. We find that MaxRew- \mathcal{S} is particularly sample inefficient because it only selects arms that are certainly feasible. MaxRew- \mathcal{U} is less sample efficient than G-ACOL because it selects arms with high reward over more informative arms.

as convexity and compactness. To evaluate these algorithms, we, therefore, consider a third synthetic instance in which the safe set is connected. We consider 10 arms in $d = 1$ that are equally spaced between 0 and 1. The reward and constraint vectors are $\theta = \phi = 1$, and the threshold is $\tau = 0.25$. Here the safe set is connected, but we can learn the constraint boundary more efficiently if we are allowed to violate the constraint during exploration.

We compare G-ACOL to MaxRew- \mathcal{S} and MaxRew- \mathcal{U} in Figure 6.3. We find that G-ACOL explores much more efficiently than both of the other approaches. MaxRew- \mathcal{S} is particularly sample inefficient, because it ensures feasibility during exploration, which is not necessary in our case. In Appendix D.2.2, we provide results for MaxRew- \mathcal{U} in all of our environments. We cannot provide these results for MaxRew- \mathcal{S} because it gets stuck in all other environments.

6.3.3 Preference Learning Experiments

We now consider the application that initially motivated us to define the CBAI problem. As discussed in Chapter 4, we are interested in situations where the reward parameter θ describes an easy-to-specify goal or metric, and the constraint parameter ϕ describes expensive-to-evaluate human preferences.

As an example of this, we consider a driving simulator, which Sadigh *et al.* [22] originally introduced to study learning reward functions to represent human preferences about driving behavior. Instead, we change the setting to have the reward θ represent an easy-to-specify goal such as “drive at velocity v ”, and the constraint ϕ represent other driving rules such as “usually drive in a lane” or “don’t get too close to other cars”, as shown in Figure 6.1. Appendix D.2 provides more details on the environment.

The agent has to select a controller to drive the car from a set of precomputed controllers \mathcal{X} , i.e., the set of “arms”. The optimal controller x^* maximizes $\theta^T x^*$ and satisfies $\phi^T x^* \leq \tau$. The agent can try out individual controllers to get feedback on whether they are feasible. In contrast to our previous experiments, the feedback is binary. However, we can still model it via a sub-Gaussian noise model by ensuring the constraint values are in $[0, 1]$ and interpreting them as probabilities. Therefore, this is a CBAI problem, and we can apply the same algorithms we applied to our synthetic problems.

ROBUSTNESS OF LEARNING CONSTRAINTS. First, we want to quantify the observation of Figure 6.1 that constraints can be a particularly robust representation of human preferences. Specifically, using constraints to represent human preferences can increase robustness to changes in the environment and allow to transfer the constraints to different reward functions. Constraints are more robust than modeling the same preferences as a penalty on the reward function. Figure 6.4 quantifies this by directly comparing the two options in terms of the reward and constraint values they achieve. In particular, we find that the magnitude of the reward penalty often

has to be updated if the environment changes, whereas the constraint formulation is robust to such changes.

RESULTS OF LEARNING CONSTRAINTS. We consider the driving scenario as a CBAI problem and study learning the constraint function. Here, we only report results for the base scenario in Figure 6.1. Appendix D.2.2 contains similar results for the other two scenarios which are qualitatively similar. In Figure 6.5, we compare the performance of ACOL and other algorithms with theoretical correctness guarantees to versions of these algorithms with heuristic confidence intervals. In both cases ACOL or G-ACOL is the most sample efficient algorithm. By choosing the heuristic confidence intervals, we can reduce the number of samples necessary by two orders of magnitude from $\sim 10^5$ to $\sim 10^3$, at the cost of theoretical guarantees. In all cases, using ACOL is preferable over alternatives because it finds the correct solution with fewer queries about the constraint function.

6.4 CONCLUSION

It is natural to formalize sequential decision-making problems in many practical situations as optimizing a known reward function subject to unknown, expensive-to-evaluate constraints. In this chapter, we studied constrained linear best-arm identification (CBAI), a linear bandit setting to learn about constraints efficiently, and proposed Adaptive Constraint Learning (ACOL) to solve this problem efficiently.

There are some limitations of the setup we studied in this chapter. Specifically, our theoretical analysis is limited to a single constraint function, which might not be appropriate for applications where the constraints are non-additive. It should be possible to extend the same theoretical ideas to multiple linear constraints that all have to be satisfied, which would allow us to apply ACOL to such situations. From the empirical perspective, we found that modeling human preferences as constraints rather than rewards can be more

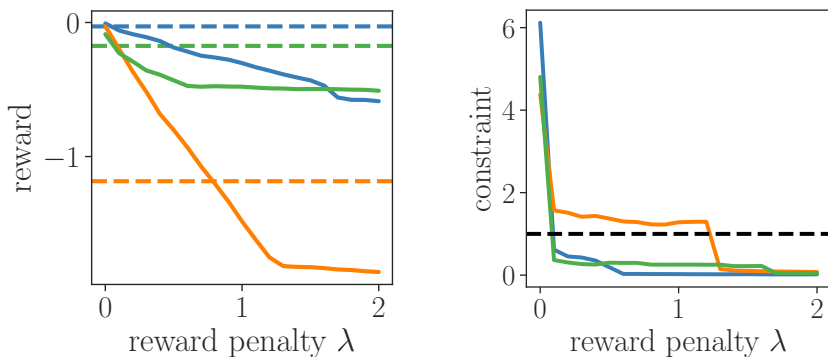


FIGURE 6.4: These plots quantify our finding that learning constraints is more robust to changes in the environment than learning a penalized reward function. We consider the three scenarios from Figure 6.1: the base scenario (—), a scenario with a different goal (—), and a scenario with a change in the environment (—). We find a policy that optimizes the reward function $\theta^T x - \lambda \phi^T x$ and plot the reward and the constraint of the solution for different values of λ . In particular, we need to choose a different value of λ for each environment to find the best solution with a constraint value below 1. The dashed horizontal lines in the reward plot show the reward a constrained solution obtains on the corresponding instance, which does not require any tuning. For each scenario, the smallest λ we find to yield a feasible solution still gives a worse solution in terms of reward than the constrained solution.

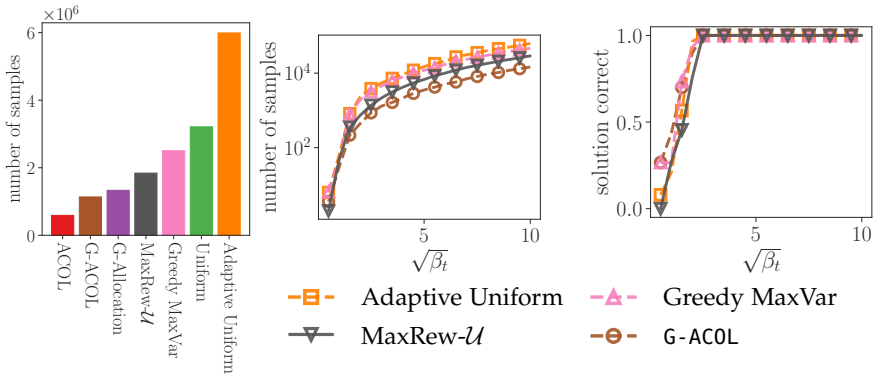


FIGURE 6.5: The left chart shows the number of iterations that all algorithms with theoretical guarantees need to find the correct solution in the driving scenario. ACOL is the fastest, but it still needs $\sim 10^5$ samples. Instead, we can use heuristic confidence intervals where we consider β_t as a hyperparameter instead of choosing the values suggested by theory. The two right plots shows the number of iterations and the percentage of the times the methods return a correct solution as a function of β_t . None of these algorithms is guaranteed to return the correct solution. But, empirically, we find that for $\sqrt{\beta_t}$ beyond the vertical line, the algorithms always return the correct solution. This again shows that tuning β_t can drastically improve the sample efficiency while still returning the correct solution empirically.

robust. Future work should study using constraints to model human preferences in more practical applications.

ACOL extends our analysis of active learning in the first part of this dissertation to the setting of learning constraint models for CMDPs. While we focused on the bandit setting in this chapter, studying exploration, similar to Chapter 4, would be a valuable extension.

When learning reward models, it is common to learn an initial model from demonstrations using IRL and then improve it using other forms of feedback and potentially active learning (e.g., [57, 58]). A similar approach could be useful for learning constraint models. Future work could, for example, combine CoCoRL (Chapter 5) with ACOL to learn a constraint model from demonstrations and then improve it using active learning.

CONCLUSION

The primary focus of this dissertation was to develop algorithmic foundations for Reinforcement Learning from Human Feedback (RLHF), focusing on sample efficiency and learning safe policies. In particular, in Chapter 1 we asked two questions:

- How can we make RLHF more sample efficient?
- How can we learn constraints from human feedback?

To address the first question, we developed algorithms to improve the sample efficiency of learning reward models in Chapters 3 and 4. To address the second question, we developed novel methods for learning constraints from human feedback in Chapters 5 and 6. We studied the proposed algorithms from a theoretical perspective, proving sample complexity guarantees, and an empirical perspective, testing them in simulations of practical RL applications.

Our research is relevant to many application areas where defining a reward function is challenging, such as robotics and autonomous driving. Importantly, improving the efficiency and robustness of RLHF promises to make RL more widely applicable. We did not focus on large language models (LLMs), where RLHF has unlocked many capabilities. Nonetheless, applying our ideas and algorithms to that setting and using them to improve the sample complexity of RLHF in LLMs is an exciting direction for future work.

The central gap to bridge between our work and real-world applications is the need for extended studies with humans. RLHF is inherently human-centered and critically depends on interaction design [18]. Practitioners find that collecting high-quality human feedback is a crucial factor for RLHF to work well (e.g., [13]), and by focusing on the algorithmic foundations of RLHF, we risk neglecting the human-centered aspects. While studying these aspects was not

the focus of this dissertation, they will be critical to ensure that RLHF systems are effective and safe.

On a technical level, our research raises many open questions, two important ones being *uncertainty quantification* and *trading off sample efficiency and computational cost*. First, all algorithms we presented require well-calibrated uncertainty estimates on the reward or constraint model, which can be difficult to provide when using complex model architectures. For example, in LLMs, the reward model is usually a large transformer model, and training an ensemble of reward models can be difficult [137]. Second, we primarily focused on improving sample efficiency in terms of human data. Sometimes, this can come at a higher computational cost or require more interactions with the environment. In practice, we must carefully balance the trade-off between sample efficiency for human data, environment interactions, and computational cost. Doing this well will likely require a better understanding of the trade-off in the quality of human feedback and the cognitive demand on the human.

In summary, we focused on building solid algorithmic foundations for RLHF and shedding light on potential applications. Making RLHF more efficient and robust paves the way for applying RL to a wider array of applications. The road ahead is long and exciting – as we continue to develop more capable and general systems, systematic study of RLHF built on solid foundations will be critical in steering AI systems to be effective and aligned with human needs.

APPENDIX

INFORMATION DIRECTED REWARD LEARNING

This appendix provides additional information about Chapter 3. In particular, we provide proofs for the propositions (Appendix A.1), details about the implementation of IDRL (Appendix A.2), and details about the experiments (Appendix A.3).

A.1 PROOFS

In this section, we provide proofs of all results mentioned in the main chapter. The results generally follow from well-known facts about Gaussian distributions and information theory.

Proposition 3.5.2. Assume we estimate $\hat{\theta}$ with Bayesian linear regression with noise variance σ^2 , and prior $\hat{\theta} \sim \mathcal{N}(0, \alpha^{-1}I)$ after collecting data

$$\mathcal{D} = ((\boldsymbol{\phi}(q_{i_1}), y_{i_1}), \dots, (\boldsymbol{\phi}(q_{i_{t-1}}), y_{i_{t-1}})).$$

Also, assume an infinitely wide prior $\alpha^{-1} \rightarrow \infty$.

We can then write the maximization in the first step of IDRL as

$$\operatorname{argmax}_{\pi, \pi' \in \Pi_c} H(\widehat{G}(\pi) - \widehat{G}(\pi') | \mathcal{D}) = \operatorname{argmax}_{\pi, \pi' \in \Pi_c} \|\mathbf{f}(\pi)_\phi - \mathbf{f}(\pi')_\phi\|_{A_{\mathcal{D}}^{-1}}^2$$

where $A_{\mathcal{D}} = \sum_{q_i \in \mathcal{Q}_c} N_i \boldsymbol{\phi}(q_i) \boldsymbol{\phi}(q_i)^T$.

Furthermore, for a given pair of policies, π_1 and π_2 , we can write the maximization in the second step of IDRL as

$$\begin{aligned} & \operatorname{argmax}_{q \in \mathcal{Q}_c} I(\widehat{G}(\pi_1) - \widehat{G}(\pi_2); (q, \hat{y}) | \mathcal{D}) \\ &= \operatorname{argmin}_{q \in \mathcal{Q}_c} \|\mathbf{f}(\pi_1)_\phi - \mathbf{f}(\pi_2)_\phi\|_{A_{\mathcal{D}, q}^{-1}}^2 \end{aligned}$$

where $A_{\mathcal{D}, q} = \boldsymbol{\phi}(q) \boldsymbol{\phi}(q)^T + \sum_{q_i \in \mathcal{Q}_c} N_i \boldsymbol{\phi}(q_i) \boldsymbol{\phi}(q_i)^T$ and N_i is the number of times q_i occurs in \mathcal{D} .

Proof. In the Bayesian linear regression setting (cf. Chapter 3 in [138]) with prior weight distribution $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \alpha^{-1})$, the posterior weight distribution is a Gaussian with covariance matrix

$$\Sigma_\theta = \left(\alpha I + \sigma^{-2} \sum_{q \in \mathcal{D}} \boldsymbol{\phi}(q) \boldsymbol{\phi}(q)^T \right)^{-1} = (\alpha I + \sigma^{-2} A_{\mathcal{D}})^{-1}$$

$$A_{\mathcal{D}} = \sum_{q \in \mathcal{D}} \boldsymbol{\phi}(q) \boldsymbol{\phi}(q)^T = \sum_{q_i \in \mathcal{Q}_c} N_i \boldsymbol{\phi}(q_i) \boldsymbol{\phi}(q_i)^T$$

For an infinitely wide prior ($\alpha^{-1} \rightarrow \infty$): $\Sigma_\theta \rightarrow \sigma^2 A_{\mathcal{D}}^{-1}$.

Using the linear mapping from $\hat{\boldsymbol{\theta}}$ to the expected return of a policy $\widehat{G}(\pi)$, the posterior variance of the difference in return between two policies is

$$\text{Var}[\widehat{G}(\pi_1) - \widehat{G}(\pi_2) | \mathcal{D}] = \sigma^2 (\mathbf{v}(\pi_1, \pi_2)^\phi)^T A_{\mathcal{D}}^{-1} \mathbf{v}(\pi_1, \pi_2)^\phi \quad (\text{A.1})$$

where $\mathbf{v}(\pi_1, \pi_2)^\phi = \mathbf{f}(\pi_1)_\phi - \mathbf{f}(\pi_2)_\phi$

The first part of the statement follows using Proposition 3.4.1:

$$\begin{aligned} & \underset{\pi, \pi' \in \Pi_c}{\text{argmax}} H(\widehat{G}(\pi) - \widehat{G}(\pi') | \mathcal{D}) \\ &= \underset{\pi, \pi' \in \Pi_c}{\text{argmax}} \text{Var}[\widehat{G}(\pi) - \widehat{G}(\pi') | \mathcal{D}] \quad (\text{Proposition 3.4.1}) \\ &= \underset{\pi, \pi' \in \Pi_c}{\text{argmax}} \sigma^2 (\mathbf{v}(\pi, \pi')^\phi)^T A_{\mathcal{D}}^{-1} \mathbf{v}(\pi, \pi')^\phi \quad (\text{Equation (A.1)}) \\ &= \underset{\pi, \pi' \in \Pi_c}{\text{argmax}} (\mathbf{v}(\pi, \pi')^\phi)^T A_{\mathcal{D}}^{-1} \mathbf{v}(\pi, \pi')^\phi \\ &= \underset{\pi, \pi' \in \Pi_c}{\text{argmax}} \|\mathbf{v}(\pi, \pi')^\phi\|_{A_{\mathcal{D}}^{-1}} \\ &= \underset{\pi, \pi' \in \Pi_c}{\text{argmax}} \|\mathbf{f}(\pi)_\phi - \mathbf{f}(\pi')_\phi\|_{A_{\mathcal{D}}^{-1}} \end{aligned}$$

After defining

$$\begin{aligned} A_{\mathcal{D}, q} &= A_{\mathcal{D} \cup \{(q, y)\}} \\ &= \boldsymbol{\phi}(q) \boldsymbol{\phi}(q)^T + \sum_{q_i \in \mathcal{Q}_c} N_i \boldsymbol{\phi}(q_i) \boldsymbol{\phi}(q_i)^T, \end{aligned}$$

the second part of the statement follows analogously to the first one after applying Proposition 3.4.1:

$$\begin{aligned}
& \operatorname{argmax}_{q \in \mathcal{Q}_c} I(\widehat{G}(\pi_1) - \widehat{G}(\pi_2); (q, \hat{y}) | \mathcal{D}) \\
&= \operatorname{argmin}_{q \in \mathcal{Q}_c} \operatorname{Var}[\widehat{G}(\pi_1) - \widehat{G}(\pi_2) | \mathcal{D} \cup \{q, y\}] \quad (\text{Prop. 3.4.1}) \\
&= \operatorname{argmin}_{q \in \mathcal{Q}_c} \sigma^2(\mathbf{v}(\pi_1, \pi_2)^\phi)^T A_{\mathcal{D}, q}^{-1} \mathbf{v}(\pi_1, \pi_2)^\phi \quad (\text{Equation (A.1)}) \\
&= \operatorname{argmin}_{q \in \mathcal{Q}_c} (\mathbf{v}(\pi_1, \pi_2)^\phi)^T A_{\mathcal{D}, q}^{-1} \mathbf{v}(\pi_1, \pi_2)^\phi \\
&= \operatorname{argmin}_{q \in \mathcal{Q}_c} \|\mathbf{v}(\pi_1, \pi_2)^\phi\|_{A_{\mathcal{D}, q}^{-1}} \\
&= \operatorname{argmin}_{q \in \mathcal{Q}_c} \|\mathbf{f}(\pi_1)_\phi - \mathbf{f}(\pi_2)_\phi\|_{A_{\mathcal{D}, q}^{-1}}
\end{aligned}$$

□

Proposition A.1.1. If $\hat{r}(s)$ is a GP, the difference in expected return between two fixed policies π, π' follows a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ with

$$\begin{aligned}
\mu &= \mathbb{E}[\widehat{G}(\pi) - \widehat{G}(\pi')] = \mathbf{v}(\pi, \pi')^T \mu_{\hat{\mathbf{r}}} \\
\sigma^2 &= \operatorname{Var}[\widehat{G}(\pi) - \widehat{G}(\pi')] = \mathbf{v}(\pi, \pi')^T \cdot \Sigma_{\hat{\mathbf{r}}} \cdot \mathbf{v}(\pi, \pi')
\end{aligned}$$

where $\mathbf{v}(\pi, \pi') = \mathbf{f}(\pi) - \mathbf{f}(\pi')$ is the difference between expected state-visitation frequencies of π and π' respectively, and $\mu_{\hat{\mathbf{r}}}$ and $\Sigma_{\hat{\mathbf{r}}}$ are the mean and covariance of the joint Gaussian distribution of the reward of all states π or π' visit.

Proof. If a random variable X is Gaussian distributed $X \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, $\boldsymbol{\mu} \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$, then for $\mathbf{a} \in \mathbb{R}^n$, $\mathbf{a}^T X$ is also Gaussian distributed $\mathbf{a}^T X \sim \mathcal{N}(\mathbf{a}^T \boldsymbol{\mu}, \mathbf{a}^T \Sigma \mathbf{a})$ (Theorem 14.2 in [139]).

We can directly apply this fact to $\hat{\mathbf{r}} \sim \mathcal{N}(\mu_{\hat{\mathbf{r}}}, \Sigma_{\hat{\mathbf{r}}})$ and $\widehat{G}(\pi) - \widehat{G}(\pi') = \mathbf{v}(\pi, \pi')^T \hat{\mathbf{r}}$, resulting in

$$\widehat{G}(\pi) - \widehat{G}(\pi') \sim \mathcal{N}(\mathbf{v}(\pi, \pi')^T \mu_{\hat{\mathbf{r}}}, \mathbf{v}(\pi, \pi')^T \Sigma_{\hat{\mathbf{r}}} \mathbf{v}(\pi, \pi')).$$

□

Proposition 3.4.1. If $\hat{r}(s)|\mathcal{D}$ is a GP, then $P(\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D})$ is Gaussian and:

$$\begin{aligned} \operatorname{argmax}_{\pi, \pi' \in \Pi_c} H(\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D}) &= \operatorname{argmax}_{\pi, \pi' \in \Pi_c} \operatorname{Var}[\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D}] \\ \operatorname{argmax}_{q \in \mathcal{Q}_c} I(\widehat{G}(\pi_1) - \widehat{G}(\pi_2); (q, \hat{y})|\mathcal{D}) &= \operatorname{argmin}_{q \in \mathcal{Q}_c} \operatorname{Var}[\widehat{G}(\pi_1) - \widehat{G}(\pi_2)|\mathcal{D} \cup \{(q, \hat{y})\}] \end{aligned}$$

Proof. If a random variable X is Gaussian distributed $X \sim \mathcal{N}(\mu, \sigma^2)$, then the entropy $H(X)$ is given by (Theorem 8.4.1 in [140])

$$H(X) = \frac{1}{2} \log(2\pi e\sigma^2). \quad (\text{A.2})$$

Proposition A.1.1 shows that the conditional distribution of $\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D}$ is Gaussian, which implies both statements.

For the first statement, observe that the entropy of $\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D}$ is

$$H(\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D}) = \frac{1}{2} \log(2\pi e \operatorname{Var}[\widehat{G}(\pi) - \widehat{G}(\pi')|\mathcal{D}]), \quad (\text{A.3})$$

and that two policies that maximize the variance on the r.h.s. also maximize the entropy, because the logarithm is a monotonic function.

To see the second statement, let $\hat{\Delta}_{\pi_1, \pi_2} = \widehat{G}(\pi_1) - \widehat{G}(\pi_2)$. Then

$$\begin{aligned} &\operatorname{argmax}_{q \in \mathcal{Q}_c} I(\hat{\Delta}_{\pi_1, \pi_2}; (q, \hat{y})|\mathcal{D}) \\ &= \operatorname{argmax}_{q \in \mathcal{Q}_c} (H(\hat{\Delta}_{\pi_1, \pi_2}|\mathcal{D}) - H(\hat{\Delta}_{\pi_1, \pi_2}|\mathcal{D} \cup \{(q, \hat{y})\})) \\ &= \operatorname{argmin}_{q \in \mathcal{Q}_c} H(\hat{\Delta}_{\pi_1, \pi_2}|\mathcal{D} \cup \{(q, \hat{y})\}) \\ &= \operatorname{argmin}_{q \in \mathcal{Q}_c} \frac{1}{2} \log(2\pi e \operatorname{Var}[\hat{\Delta}_{\pi_1, \pi_2}|\mathcal{D} \cup \{(q, \hat{y})\}]) \\ &= \operatorname{argmin}_{q \in \mathcal{Q}_c} \operatorname{Var}[\hat{\Delta}_{\pi_1, \pi_2}|\mathcal{D} \cup \{(q, \hat{y})\}]. \end{aligned}$$

Here we wrote the information gain in terms of conditional entropies (Theorem 2.4.1 in [140]), and used that only one of the terms depends

on q . This turns the maximization of information gain into a minimization of a conditional entropy. As before, we can further simplify this to minimizing conditional variance by using the entropy of a Gaussian and the fact that the logarithm is a monotonic function. \square

Proposition 3.4.2. Let q be a linear reward query. If the prior belief about the reward $\hat{r}(s)$ is a GP, then the posterior belief about the reward $\hat{r}(s)|(q, y)$ is also a GP.

Proof. Let $q = (X, C)$ be a linear reward query, i.e., $X = \{s_1, \dots, s_N\} \subseteq \mathcal{S}$ is a set of states and $C = \{c_1, \dots, c_N\}$ a set of linear weights, and $y = \sum_{j=1}^N c_j r(s_j)$ the corresponding observation.

Let $S^* = \{s_1^*, \dots, s_n^*\} \subseteq \mathcal{S}$ be a set of states for which we want to compute the posterior belief. We show that

$$P(\hat{r}(s_1^*), \dots, \hat{r}(s_n^*)|(q, y)) \sim \mathcal{N}(\boldsymbol{\mu}_q^*, \Sigma_q^*)$$

for some $\boldsymbol{\mu}_q^*$ and Σ_q^* . Because this holds for any set of states S^* , it shows that the posterior reward model is a GP.

We define the following vector notation:

$$\begin{aligned} \mathbf{c} &= (c_1, \dots, c_N)^T \in \mathbb{R}^N \\ \hat{\mathbf{r}} &= (\hat{r}(s_1), \dots, \hat{r}(s_N))^T \in \mathbb{R}^N \\ \hat{\mathbf{r}}^* &= (\hat{r}(s_1^*), \dots, \hat{r}(s_n^*))^T \in \mathbb{R}^n \end{aligned}$$

such that $y = \mathbf{c}^T \hat{\mathbf{r}}$.

The prior distribution of $\hat{\mathbf{r}}$ is Gaussian, i.e.,

$$P(\hat{\mathbf{r}}|X) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma),$$

with mean $\boldsymbol{\mu}$ and covariance Σ .

Because \hat{y} is a linear function of $\hat{\mathbf{r}}$ plus Gaussian noise, the prior distribution of \hat{y} is also Gaussian (Theorem 14.2 in [139]):

$$P(\hat{y}|X, C) \sim \mathcal{N}(\mathbf{c}^T \boldsymbol{\mu}, \mathbf{c}^T \Sigma \mathbf{c} + \sigma_n^2 I).$$

Further, $\hat{\mathbf{r}}$ and $\hat{\mathbf{r}}^*$ are jointly Gaussian distributed:

$$P\left(\begin{bmatrix} \hat{\mathbf{r}} \\ \hat{\mathbf{r}}^* \end{bmatrix} | X, S^*\right) \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}^* \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma^* \\ (\Sigma^*)^T & \Sigma^{**} \end{bmatrix}\right)$$

where $\boldsymbol{\mu}^*$ is the mean of $\hat{\mathbf{r}}^*$, and $\Sigma^* = \text{Cov}[\hat{\mathbf{r}}, \hat{\mathbf{r}}^*]$ and $\Sigma^{**} = \text{Cov}[\hat{\mathbf{r}}^*, \hat{\mathbf{r}}^*]$ denote the components of the joint covariance matrix.

Hence, $\hat{\mathbf{r}}^*$ and \hat{y} are also jointly Gaussian distributed:

$$P\left(\begin{bmatrix} \hat{\mathbf{r}}^* \\ \hat{y} \end{bmatrix} \mid X, C, S^*\right) \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}^* \\ \mathbf{c}^T \boldsymbol{\mu} \end{bmatrix}, \begin{bmatrix} \Sigma^{**} & (\Sigma^*)^T \mathbf{c} \\ \mathbf{c}^T \Sigma^* & \mathbf{c}^T \Sigma \mathbf{c} + \sigma_n^2 I \end{bmatrix}\right)$$

where we used the linearity of the covariance function to find the covariance matrix:

$$\begin{aligned} \text{Cov}[\hat{y}, \hat{\mathbf{r}}^*] &= \text{Cov}[\mathbf{c}^T \hat{\mathbf{r}}, \hat{\mathbf{r}}^*] \\ &= \mathbf{c}^T \text{Cov}[\hat{\mathbf{r}}, \hat{\mathbf{r}}^*] = \mathbf{c}^T \Sigma^* \\ \text{Cov}[\hat{\mathbf{r}}^*, \hat{y}] &= (\Sigma^*)^T \mathbf{c} \end{aligned}$$

Finally, we can use standard results on conditioning Gaussian distributions (cf. Chapter A.2 in [30]) to find that the conditional distribution is still Gaussian:

$$P(\hat{\mathbf{r}}^* \mid (q, y)) = P(\hat{\mathbf{r}}^* \mid y, X, C, S^*) \sim \mathcal{N}(\boldsymbol{\mu}_q^*, \Sigma_q^*)$$

with

$$\begin{aligned} \boldsymbol{\mu}_q^* &= \boldsymbol{\mu}^* + ((\Sigma^*)^T \mathbf{c})(\mathbf{c}^T \Sigma \mathbf{c} + \sigma_n^2 I)^{-1}(y - \mathbf{c}^T \boldsymbol{\mu}) \\ \Sigma_q^* &= \Sigma^{**} - ((\Sigma^*)^T \mathbf{c})(\mathbf{c}^T \Sigma \mathbf{c} + \sigma_n^2 I)^{-1}(\mathbf{c}^T \Sigma^*). \end{aligned}$$

When conditioning the distribution, we replaced our belief about the observation \hat{y} with its actual realization y . \square

A.2 IMPLEMENTATION DETAILS

A.2.1 Baselines

Algorithm 7 shows pseudocode for the general reward learning algorithm that all of our baselines implement. They only differ in the choice of acquisition function in line 5.

UNIFORM SAMPLING. The uniform sampling baseline runs Algorithm 7 with q^* sampled uniformly from \mathcal{Q}_c instead of line 5.

Algorithm 7 Generic reward learning algorithm using an acquisition function $u(q, \mathcal{D})$. Our baselines use information gain and expected improvement for u . Uniform sampling samples q^* uniformly from \mathcal{Q}_c instead of line 5.

```

1:  $\mathcal{D} \leftarrow \{\}$ 
2: Initialize reward model with prior distribution  $P(\hat{r})$ 
3: while not converged do
4:   Select a query:
5:      $q^* \in \operatorname{argmax}_{q \in \mathcal{Q}_c} u(q, \mathcal{D})$ 
6:   Query  $q^*$  and update reward model:
7:      $y^* \leftarrow \text{Response to query } q^*$ 
8:      $P(\hat{r} | \mathcal{D} \cup \{(q^*, y^*)\}) \propto P(y^* | \hat{r}, \mathcal{D}, q^*) P(\hat{r} | \mathcal{D})$ 
9:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(q^*, y^*)\}$ 
10:  $\bar{r} \leftarrow$  mean estimate of the reward model
11:  $\bar{\pi}^* \leftarrow \text{RL}(\bar{r})$ 
12: return  $\bar{\pi}^*$ 

```

INFORMATION GAIN ON THE REWARD. We can write the information gain on the reward in terms of conditional entropies (Theorem 2.4.1 in [140]):

$$u(s, \mathcal{D}) = I((q, \hat{y}); \hat{r} | \mathcal{D}) = H(\hat{y} | \mathcal{D}, q) - H(\hat{y} | \hat{r}, \mathcal{D}, q). \quad (\text{A.4})$$

For linear reward query, the second term of equation (A.4) is constant and $P(\hat{y} | \mathcal{D}, q)$ is Gaussian. Hence, its entropy is (Theorem 8.4.1 in [140]):

$$H(\hat{y} | \mathcal{D}, q) = \frac{1}{2} \log(2\pi e \operatorname{Var}[\hat{y} | \mathcal{D}, q]).$$

And because the logarithm is a monotonic function, we simply have

$$\operatorname{argmax}_{q \in \mathcal{Q}_c} I((q, \hat{y}); \hat{r} | \mathcal{D}) = \operatorname{argmax}_{q \in \mathcal{Q}_c} \operatorname{Var}[\hat{y} | \mathcal{D}, q].$$

Hence, for a GP reward model with linear reward queries, we use $u(s, \mathcal{D}) = \operatorname{Var}[\hat{y} | \mathcal{D}, q]$ in practice.

EXPECTED IMPROVEMENT. We define EI as:

$$u(s, \mathcal{D}) = L \cdot \Phi(M) + \text{Var}[\hat{y}|\mathcal{D}, q] \cdot \rho(M),$$

$$M = \frac{L}{\text{Var}[\hat{y}|\mathcal{D}, q]}, \quad L = \mathbb{E}[\hat{y}|\mathcal{D}, q] - y_{\max} - \zeta,$$

where ρ is the p.d.f and Φ is the c.d.f of the standard normal distribution. y_{\max} is the best observation made so far. ζ is a hyperparameter, that we set to 0.001. EI quantifies how much larger a new observation is expected to be than the largest observation made so far. In our setting, EI is only applicable if observations are numerical. In particular, we cannot use EI if the queries are comparisons of states or trajectories.

EXPECTED POLICY DIVERGENCE. Expected policy divergence (EPD; [52]) compares two policies: $\tilde{\pi}$ trained from a reward model conditioned on the current dataset \mathcal{D} , and π^* estimates a policy trained from a reward model conditioned on $\mathcal{D} \cup \{(q, y)\}$. EPD aims to quantify the effect of making a query q and observing response y on the currently optimal policy. For each potential query it assumes an observation at an upper confidence bound conditioned on the current model, and then selects observations that maximize some distance measure between policies $d(\tilde{\pi}, \pi^*)$. Daniel *et al.* [52] introduce EPD using the KL divergence $D_{\text{KL}}(\tilde{\pi}||\pi^*)$ as distance measure $d(\tilde{\pi}, \pi^*)$. However, in most of our experiments, the policies are deterministic, in which case the KL divergence is not well-defined. For tabular environments, we define d to count the number of states in which the policies differ. For the *Driver* environment we use an ℓ_2 -distance between the policy representations.

MAXIMUM REGRET. *Maximum Regret* (MR; [61]) assumes a set of candidate reward functions $\mathcal{R}_c = \{r_1, \dots, r_n\}$ to be given and then considers a set of candidate policies $\Pi_c = \{\pi_1, \dots, \pi_n\}$, where each policy π_i is optimal for one of the reward functions r_i . MR can queries comparison queries of the form $q = (\pi_i, \pi_j)$. In practice, we use MR for queries that compare trajectories sampled from π_i and π_j .

MR aims to compare policies π_i and π_j that perform poorly when evaluated under each other’s reward function r_j and r_i respectively. If $G_{r_i}(\pi_j)$ is the return of policy π_j evaluated using reward function r_i , MR is defined as

$$u((\pi_i, \pi_j), \mathcal{D}) = P(r_i|\mathcal{D}) \cdot P(r_j|\mathcal{D}) \cdot (R(r_i, r_j) + R(r_j, r_i))$$

where $R(r_i, r_j)$ is a measure of regret of a policy optimized for reward function r_i when evaluated under reward function r_j . Wilde *et al.* [61] use a regret measure based on a ratio of returns: $R(r_i, r_j) = 1 - G_{r_j}(\pi_i)/G(\pi_j)$. However, this measure is only meaningful if all rewards are positive, which is not the case in our experiments. Therefore, we instead use a regret measure based on differences of returns: $R(r_i, r_j) = G(\pi_j) - G_{r_j}(\pi_i)$.¹

For computing the probabilities $P(r_i|\mathcal{D})$, Wilde *et al.* [61] use a simple Bayesian model that assumes a uniform prior and a likelihood of making an observation of the form

$$P(y = +1|q = (\pi_i, \pi_j)) = \begin{cases} p & \text{if } G(\pi_i) > G(\pi_j) \\ 1 - p & \text{else} \end{cases} \quad (\text{A.5})$$

with $0.5 < p \leq 1$. In the main chapter we report results for MR using a GP reward model. We tested the simple Bayesian model in equation (A.5) in preliminary experiments, and found it to result in comparable results to the GP model with observations simulated using a linear observation model.

A.2.2 IDRL for Deep RL

This section provides some additional implementation details for our Deep RL implementation of IDRL, and compares it to the implementation of Christiano *et al.* [12].

¹ This change was suggested by the authors of Wilde *et al.* [61] in personal communication to deal with negative rewards.

A.2.2.1 Hyperparameter Choices

NEURAL NETWORK MODEL. We use the same network architecture as Christiano *et al.* [12]: a two-layer neural network with 64 hidden units each and leaky-ReLU activation functions ($\alpha = 0.01$). For training we use ℓ_2 -regularization with $\lambda = 0.5$.

POLICY TRAINING. We use the `stable-baselines3` implementation of SAC [141], with default hyperparameters. For training the policy, we append a feature to the observations that measure the remaining time within an episode, $f_t = (t_{\max} - t)/t_{\max}$, where t is the current time step in an episode and t_{\max} is the episode length. Adding this feature tends to speed up training significantly in the MuJoCo environments. We do not add this feature for learning the reward function. Policies are trained for 10^7 timesteps in total.

SAMPLING RATE. We provide 25% of samples to the reward model before starting to train the policy, and during training provide samples at a sampling rate proportional to $1/T$. Concretely, if N_s samples are provided in N_b batches over the course of training, the i -th batch will contain $\frac{N_s}{H_{N_b}} \cdot \frac{1}{T}$ samples, where H_n is the n -th harmonic number.

CANDIDATE POLICIES. We maintain a set of 3 candidate policies, that are each updated 10^7 timesteps, as the main policy. The candidate policies are updated in regular intervals, which are controlled by a hyperparameter N_p . Over the course of training, the candidate policies will be updated N_p times using $10^7/N_p$ timesteps each time.

HYPERPARAMETER TUNING. We only tuned two hyperparameters explicitly: N_b , the number of batches of training samples the model gets during training, and N_p the number of times the candidate policies are updated during training. We selected all other hyperparameters after preliminary experiments and to be as similar as possible to Christiano *et al.* [12]. We first tuned N_b using a random acquisition function and values in $\{10, 100, 1000, 10000\}$, and

chose $N_b = 1000$ which gave the best performance evaluated over 5 random seeds. We choose the same N_b for all acquisition functions. Then, we tuned N_p for the IDRL acquisition function and values in $\{10, 100, 200, 400, 600, 800, 1000\}$. We chose $N_p = 100$ which lead to best performance evaluated over 5 random seeds. All hyperparameters were only tuned on the HalfCheetah environment.

A.2.2.2 Comparison to Christiano *et al.*

In this section we point out differences in our Deep RL setup compared to Christiano *et al.* [12]. Some of the modifications are necessary for applying IDRL. Other differences result from us not being able to reproduce the exact environments and hyperparameters because Christiano *et al.* [12] do not provide code of their experiments.

REWARD MODEL. Christiano *et al.* [12] model the reward function with an ensemble of DNNs. We learn a feature representation using a single DNN, and combine this with a Bayesian linear model which makes computing the IDRL objective more straightforward.

POLICY LEARNING. We use SAC while Christiano *et al.* [12] use TRPO for learning the policy. We chose SAC because it is significantly more sample efficient in MuJoCo environments.

SAMPLING RATE. Christiano *et al.* [12] provide 25% of total samples to the model initially, and provide the rest of the samples at an adaptive sampling rate which they choose to be “roughly proportional to $2 \cdot 10^6 / (T + 2 \cdot 10^6)$ ” (App. A.1 in [12]), where T is the number of environment interactions so far. Unfortunately, they do not provide enough information to exactly reproduce their sampling schedule. Instead, we simplify the schedule to be proportional to $1/T$.

CLIP LENGTH. Christiano *et al.* [12] query comparisons between clips that “last 1.5 seconds, which varies from 15 to 60 timesteps

depending on the task” (App. A.1 in [12]). Unfortunately, they do not specify the framerate used for each task, so we can not reproduce the exact clip lengths. Instead, we simply choose a length of 40 timesteps for each environment which is roughly in the middle of the range they provide.

OBSERVATIONS. From their paper it is unclear whether Christiano *et al.* [12] include the agents position in the observation in locomotion environments such as the HalfCheetah. Note, that including the observation makes the reward learning task much easier because the reward function is linear in the change of the agent’s x-position. Therefore, we do not include the position in the observation which we use to predict the reward function.

PENALTIES FOR TERMINATION. Most of the standard MuJoCo environments have termination conditions, that, e.g., terminate an episode when the robot falls over. Such termination can leak information about the reward, i.e., longer episodes are better. Therefore, Christiano *et al.* [12] replace “these termination conditions by a penalty which encourages the parameters to remain in the range” (App. A in [12]). Unfortunately, they do not specify the exact penalties they use. We also remove the termination condition, but replace it with a bonus for “being alive” which is implemented in the version 3 environments of OpenAI Gym.

A.3 EXPERIMENT DETAILS

A.3.1 *Environments*

This section provides more details on the environments used in the experiments in Section 6.3.

A.3.1.1 Chain

The *Chain* environment has a discrete state space with N states, and a discrete action space with 2 actions a_r and a_l . In the first M states of the chain both actions moves the agent right, whereas in the last $N - M$ states a_l moves the agent left and a_r moves the agent right. The dynamics are deterministic. The initial state distribution is uniform over the state space.

For the GP model of the reward, we choose a squared-exponential (SE) kernel

$$k(s, s') = \sigma^2 \exp\left(-\frac{d(s, s')^2}{2l^2}\right)$$

with variance $\sigma = 2$ and lengthscale $l = 3$. The distance d counts the number of states between s and s' on the chain.

A.3.1.2 Junction

The *Junction* environment has a discrete state space with $N + 2M$ states, and a discrete action space with 2 actions a_1 and a_2 . In the first N states either action moves the agent right. From state s_N action a_1 moves the agent to s_{A_1} and action a_2 moves the agent to s_{B_1} . In either of the two paths the agent moves to one of the adjacent states with probability 0.5, independent of the action it took. The reward of states s_1, \dots, s_N is 0, the reward of states s_{B_1}, \dots, s_{B_M} is 0.8, and the reward of state s_{A_i} is

$$r(s_{A_i}) = 1 - \left(0.7 \cdot \frac{i}{M} - 1\right)^2$$

This reward function ensures that the average reward in the upper chain is smaller than 0.8 but the maximum reward is bigger than 0.8. The initial state distribution is uniform over the state space.

For the GP model of the reward, we choose a SE kernel with variance $\sigma = 2$ and lengthscale $l = 3$. The distance d measures the shortest path between s and s' on graph that defines the *Junction* (disregarding the transition function).

A.3.1.3 *Gridworld*

The *Gridworld* environment consists of a 10×10 grid in which 20 objects of each of 10 different types are placed, so 20 objects in total. Each object type gives a reward uniformly sampled from $[-1, 1]$ when standing on it, while floor tiles give 0 reward. Between each two cells, with probability 0.3 there is a wall. The environment has a discrete states space with 100 states and a discrete action space with 5 actions: *north*, *east*, *south*, *west*, and *stay*. The dynamics are deterministic. The initial position of the agent is randomly selected but fixed for one instance of the environment.

For the GP model of the reward we choose a kernel

$$k(s, s') = \begin{cases} 1 & \text{if in } s \text{ and } s' \text{ the agent is standing} \\ & \text{on the same object type} \\ 0 & \text{else} \end{cases}$$

so that the model learns a reward for each of the object types independently.

A.3.1.4 *Driver*

We implement the *Driver* environment based on code provided by Sadigh *et al.* [22] and Bıyık *et al.* [59]. The environment uses point-mass dynamics with a continuous state and action space. The state $s = (x, y, \theta, v)$ consists of the agent's position (x, y) , its heading θ , and its velocity v . The actions $a = (a_1, a_2)$ consist of a steering input and an acceleration. The environment dynamics are defined as

$$\begin{aligned} s_{t+1} &= (x_{t+1}, y_{t+1}, \theta_{t+1}, v_{t+1}) \\ &= (x_t + \Delta x, y_t + \Delta y, \theta_t + \Delta\theta, \text{clip}(v_t + \Delta v, -1, 1)) \\ (\Delta x, \Delta y, \Delta\theta, \Delta v) &= (v \cos \theta, v \sin \theta, v a_1, a_2 - \alpha v) \end{aligned}$$

where $\alpha = 1$ is a friction parameter, and the velocity is clipped to $[-1, 1]$ at each timestep.

The environment contains a highway with three lanes. In addition to the agent, the environment contains a second car that changes from the right to the middle lane, moving on a predefined trajectory. The reward function is linear in a set of features

$$f(s) = (f_1(s), f_2(s), f_3(s), f_4(s), 1)$$

where $f_1(s) \propto \exp(d_1^2)$, d_1 is the distance to the closest lane center, $f_2(s) \propto (v - 1)^2$, $f_3(s) \propto \sin(\theta)$, and $f_4(s) \propto \exp(-d_2^2 - cd_3^2)$ where d_2 and d_3 are the distance between the agent’s car and the other car along the x and y directions respectively, and c is a constant. Reward functions for the environment are sampled from a Gaussian with zero mean and unit covariance. The first 4 features are normalized before the constant is appended.

We use a fixed time horizon $T = 50$, and policies are parameterized by 5 actions that are each applied for 10 time steps. For solving the environment we optimize over these policies using an L-BFGS-B solver as proposed by Sadigh *et al.* [22]. We additionally use the set of candidate policies Π_c as a lookup table to reduce the variance of this solver. In particular, if Π_c contains a policy that is better for a given reward function than the one returned by the solver, we choose this one instead. This was first proposed by Wilde *et al.* [61].

A.3.1.5 MuJoCo Corridor

Our MuJoCo *Corridor* environments are based on code of the maze environments by Duan *et al.* [142]. Our “maze” is a corridor of 13 cells. The robot starts in the leftmost cell and one of the cells is a fixed goal cell. The true reward function, that is not directly available to the agent, rewards the agent proportional to its velocity in positive x -direction if the agent is before the goal, and rewards the agent proportionally to its velocity in negative x -direction if the agent is past the goal. This provides a reward function that is harder to learn than just moving in one direction. We encode this reward function as a linear function of a set of features

$$f(s) = (v_x I_1, v_y I_1, I_1, \dots, v_x I_{13}, v_y I_{13}, I_{13})^T \in \mathbb{R}^{39}$$

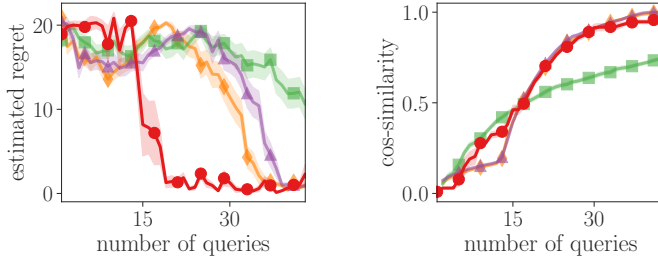


FIGURE A.1: Regret and cosine similarity in the *Ant-Corridor* environment, comparing IDRL (—●—) to IGR (—▲—), EI (—◆—), and uniform sampling (—■—). The experimental setup is exactly the same to the results shown in Figure 3.4b in the main chapter. The *Ant-Corridor* environment is the same as the *Swimmer-Corridor*, only with a different robot.

where I_k are indicator features that are 1 if the agent is in cell k and 0 otherwise, and v_x and v_y are the x- and y-velocity of the center of mass of the *Swimmer*.

We use *augmented random search* [70] with linear policies to solve the environment for a given reward function. The policy is linear in a separate set of features than the reward function. The features for the policy are based on the standard features provided by the *Swimmer* environment, extended by an indicator feature for the *Swimmer* being in each of the cells.

A.3.2 Additional Results

A.3.2.1 Ant Corridor

Figure A.1 shows results in the *Ant-Corridor*, using the same experimental setup as the results in Figure 3.4b in the main chapter, which shows results in the *Swimmer-Corridor*.

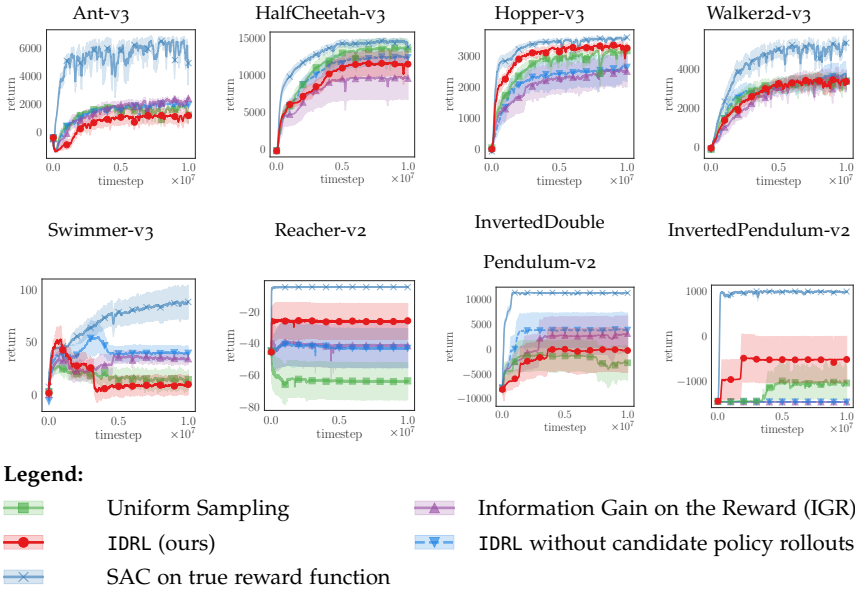


FIGURE A.2: Return of policies trained using a model trained from 1400 comparison for each of the MuJoCo environments, corresponding to the average shown in Figure 3.5.

A.3.2.2 Individual Deep RL Experiments

In Figure A.2 we provide learning curves for the individual MuJoCo environments that were aggregated to create Figure 3.5 in the main chapter. To aggregate the results we normalized the return in each environment:

$$G_{\text{norm}}(\pi) = 100 \cdot \frac{G(\pi) - G(\pi_{\text{rand}})}{G(\pi^*) - G(\pi_{\text{rand}})},$$

where π_{rand} is a policy that samples action uniformly at random, and π^* is an expert policy trained using SAC on the true reward. This results in a score that is 0 for a policy that performs as well as a random policy, and 100 for a policy that matches an expert performance. In Figure 3.5 this score is averaged over all environments.

B

ACTIVE EXPLORATION FOR IRL

This appendix provides additional details about Chapter 4. In particular, we provide proofs of the theoretical results (Appendix B.1), and more details about the experiments (Appendix B.2).

B.1 PROOFS

B.1.1 *Simulation Lemmas*

In this section, we establish several simulation lemmas that we will use throughout our analysis. Some of the results were already derived in prior work for the infinite-horizon setting, e.g., by Zanette *et al.* [87] and Metelli *et al.* [72]. For completeness, we provide proofs for all results in the finite-horizon setting.

Definition B.1.1 (Occupancy measures). We define $\mu_{\mathcal{M},\pi}^{h,h'}(s|s_0)$ as the probability of being in state s at timestep $h' \geq h$ following a policy π in $\text{MDP} \setminus \mathbb{R} \mathcal{M}$ starting in state s_0 at timestep h . We can compute it recursively as:

$$\begin{aligned}\mu_{\mathcal{M},\pi}^{h,h}(s'|s) &:= \mathbb{1}_{\{s'=s\}} \\ \mu_{\mathcal{M},\pi}^{h,h'+1}(s'|s) &:= \sum_{s'',\tilde{a}} P(s'|s'',\tilde{a})\pi_{h'}(\tilde{a}|s'')\mu_{\mathcal{M},\pi}^{h,h'}(s''|s)\end{aligned}$$

We define the same probability for state-action pairs analogously:

$$\begin{aligned}\mu_{\mathcal{M},\pi}^{h,h'}(s',a'|s,a) &:= \mathbb{1}_{\{s'=s,a'=a\}} \\ \mu_{\mathcal{M},\pi}^{h,h'+1}(s',a'|s,a) &:= \sum_{\tilde{s},\tilde{a}} \pi_{h'}(a'|s')P(s'|\tilde{s},\tilde{a})\mu_{\mathcal{M},\pi}^{h,h'}(\tilde{s},\tilde{a}|s,a)\end{aligned}$$

as well as

$$\begin{aligned}\mu_{\mathcal{M},\pi}^{h,h}(s',a'|s) &:= \pi_h(a'|s') \mathbb{1}_{\{s'=s\}} \\ \mu_{\mathcal{M},\pi}^{h,h'+1}(s',a'|s) &:= \sum_{\tilde{s},\tilde{a}} \pi_{h'}(a'|s') P(s'|\tilde{s},\tilde{a}) \mu_{\mathcal{M},\pi}^{h,h'}(\tilde{s},\tilde{a}|s)\end{aligned}$$

Because the environment is Markovian, it also holds for $h' > h$ that

$$\mu_{\mathcal{M},\pi}^{h,h'}(s'|s) = \sum_{\tilde{s},\tilde{a}} \mu_{\mathcal{M},\pi}^{h+1,h'}(s'|\tilde{s}) P(\tilde{s}|s,a) \pi_h(a|s)$$

and equivalently for state-action pairs.

Lemma B.1.1. The value function and Q-function of a policy π in an MDP $\mathcal{M} \cup r$ at timestep h can be expressed as:

$$\begin{aligned}V_{\mathcal{M} \cup r}^{\pi,h}(s) &= \sum_{h'=h}^H \sum_{s',a'} \mu_{\mathcal{M},\pi}^{h,h'}(s',a'|s) r_{h'}(s',a') \\ Q_{\mathcal{M} \cup r}^{\pi,h}(s,a) &= \sum_{h'=h}^H \sum_{s',a'} \mu_{\mathcal{M},\pi}^{h,h'}(s',a'|s,a) r_{h'}(s',a')\end{aligned}$$

Proof. We show the result for the value function; the derivation for the Q-function is analogous.

Note that for $h = H$ the statement holds because $V_{\mathcal{M} \cup r}^{\pi,H}(s) = 0$. The general result follows by induction. Assume that for $h + 1$ the statement holds. Then:

$$\begin{aligned}V_{\mathcal{M} \cup r}^{\pi,h}(s) &\stackrel{(a)}{=} \sum_a \pi_h(a|s) \left(r_h(s,a) + \sum_{s'} P(s'|s,a) V_{\mathcal{M} \cup r}^{\pi,h+1}(s') \right) \\ &\stackrel{(b)}{=} \sum_a \pi_h(a|s) \left(r_h(s,a) + \sum_{s'} P(s'|s,a) \left(\sum_{h'=h+1}^H \sum_{s'',a''} \mu_{\mathcal{M},\pi}^{h+1,h'}(s'',a''|s') r_{h'}(s'',a'') \right) \right) \\ &\stackrel{(c)}{=} \sum_a \pi_h(a|s) r_h(s,a) + \sum_{h'=h+1}^H \sum_{s',a'} \mu_{\mathcal{M},\pi}^{h,h'}(s'|s) \pi_{h'}(a'|s') r_{h'}(s',a') \\ &\stackrel{(d)}{=} \sum_{h'=h}^H \sum_{s',a'} \mu_{\mathcal{M},\pi}^{h,h'}(s'|s) \pi_{h'}(a'|s') r_{h'}(s',a')\end{aligned}$$

where (a) uses the Bellman equation, (b) the induction step, (c) uses Definition B.1.1 and relabels $s'' \rightarrow s'$, $a'' \rightarrow a'$, and (d) uses Definition B.1.1 again and relabels $a \rightarrow a'$. \square

Lemma B.1.2 (Simulation lemma 1 by Metelli *et al.* [72]). Let \mathcal{M} be an MDP $\setminus \mathbb{R}$, and r, \hat{r} two reward functions with corresponding optimal policies $\pi^*, \hat{\pi}^*$. Then,

$$Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\mathcal{M} \cup \hat{r}}^{\hat{\pi}^*, h}(s, a) \leq \sum_{h'=h}^H \sum_{s', a'} \mu_{\mathcal{M}, \pi^*}^{h, h'}(s', a' | s, a) (r_{h'}(s', a') - \hat{r}_{h'}(s', a'))$$

$$V_{\mathcal{M} \cup r}^{\pi^*, h}(s) - V_{\mathcal{M} \cup \hat{r}}^{\hat{\pi}^*, h}(s) \leq \sum_{h'=h}^H \sum_{s', a'} \mu_{\mathcal{M}, \pi^*}^{h, h'}(s', a' | s) (r_{h'}(s', a') - \hat{r}_{h'}(s', a'))$$

Proof. Note that $Q_{\mathcal{M} \cup \hat{r}}^{\hat{\pi}^*, h}(s, a) \geq Q_{\mathcal{M} \cup \hat{r}}^{\pi^*, h}(s, a)$ for all s, a because $\hat{\pi}^*$ is optimal for \hat{r} . Hence

$$Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\mathcal{M} \cup \hat{r}}^{\hat{\pi}^*, h}(s, a) \leq Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\mathcal{M} \cup \hat{r}}^{\pi^*, h}(s, a)$$

$$\stackrel{(a)}{=} \sum_{h'=h}^H \sum_{s', a'} \mu_{\mathcal{M}, \pi^*}^{h, h'}(s', a' | s, a) (r_{h'}(s', a') - \hat{r}_{h'}(s', a'))$$

where (a) uses Lemma B.1.1. After observing $V_{\mathcal{M} \cup \hat{r}}^{\hat{\pi}^*, h}(s) \geq V_{\mathcal{M} \cup \hat{r}}^{\pi^*, h}(s)$, the second result follows analogously. \square

Lemma 4.2.3. Let \mathcal{M} be an MDP $\setminus \mathbb{R}$, r, \hat{r} two reward functions with optimal policies $\pi^*, \hat{\pi}^*$. Then,

$$Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, h}(s, a)$$

$$\leq \sum_{h'=h}^H \sum_{s', a'} \left(\mu_{\mathcal{M}, \pi^*}^{h, h'}(s', a' | s, a) - \mu_{\mathcal{M}, \hat{\pi}^*}^{h, h'}(s', a' | s, a) \right) (r_{h'}(s', a') - \hat{r}_{h'}(s', a'))$$

Proof.

$$\begin{aligned}
& Q_{\mathcal{M}_{\cup r}}^{\pi^*,h}(s,a) - Q_{\mathcal{M}_{\cup r}}^{\hat{\pi}^*,h}(s,a) \\
&= (Q_{\mathcal{M}_{\cup r}}^{\pi^*,h}(s,a) - Q_{\mathcal{M}_{\cup \hat{r}}}^{\hat{\pi}^*,h}(s,a)) + (Q_{\mathcal{M}_{\cup \hat{r}}}^{\hat{\pi}^*,h}(s,a) - Q_{\mathcal{M}_{\cup r}}^{\hat{\pi}^*,h}(s,a)) \\
&\stackrel{(a)}{\leq} \sum_{h'=h}^H \sum_{s',a'} \mu_{\mathcal{M},\pi^*}^{h,h'}(s',a'|s,a) (r_{h'}(s',a') - \hat{r}_{h'}(s',a')) + (Q_{\mathcal{M}_{\cup \hat{r}}}^{\hat{\pi}^*,h}(s,a) - Q_{\mathcal{M}_{\cup r}}^{\hat{\pi}^*,h}(s,a)) \\
&\stackrel{(b)}{=} \sum_{h'=h}^H \sum_{s',a'} \mu_{\mathcal{M},\pi^*}^{h,h'}(s',a'|s,a) (r_{h'}(s',a') - \hat{r}_{h'}(s',a')) \\
&+ \sum_{h'=h}^H \sum_{s',a'} \mu_{\mathcal{M},\hat{\pi}^*}^{h,h'}(s',a'|s,a) (\hat{r}_{h'}(s',a') - r_{h'}(s',a')) \\
&= \sum_{h'=h}^H \sum_{s',a'} \left(\mu_{\mathcal{M},\pi^*}^{h,h'}(s',a'|s,a) - \mu_{\mathcal{M},\hat{\pi}^*}^{h,h'}(s',a'|s,a) \right) (r_{h'}(s',a') - \hat{r}_{h'}(s',a')) \\
&\quad \text{where (a) uses Lemma B.1.2 and (b) uses Lemma B.1.1.} \quad \square
\end{aligned}$$

Lemma B.1.3. Let $\mathcal{M}_1, \mathcal{M}_2$ be two $\text{MDP} \setminus \mathbb{R}$ with transition dynamics P_1, P_2 respectively, r a reward function and π a policy. Then, for any state s and timestep h :

$$\begin{aligned}
& V_{\mathcal{M}_2 \cup r}^{\pi,h}(s) - V_{\mathcal{M}_1 \cup r}^{\pi,h}(s) \\
&= \sum_{h'=h}^H \sum_{s',a',s''} \mu_{\mathcal{M}_2,\pi}^{h,h'}(s';s) \pi_{h'}(a'|s') (P_2(s''|s',a') - P_1(s''|s',a')) V_{\mathcal{M}_1 \cup r}^{\pi,h'+1}(s'') \\
&\text{and}
\end{aligned}$$

$$\begin{aligned}
& V_{\mathcal{M}_1 \cup r}^{\pi,h}(s) - V_{\mathcal{M}_2 \cup r}^{\pi,h}(s) \\
&= \sum_{h'=h}^H \sum_{s',a',s''} \mu_{\mathcal{M}_2,\pi}^{h,h'}(s';s) \pi_{h'}(a'|s') (P_1(s''|s',a') - P_2(s''|s',a')) V_{\mathcal{M}_1 \cup r}^{\pi,h'+1}(s'') \\
&\text{Moreover,}
\end{aligned}$$

$$\begin{aligned}
& \left| V_{\mathcal{M}_2 \cup r}^{\pi,h}(s) - V_{\mathcal{M}_1 \cup r}^{\pi,h}(s) \right| \\
&\leq \sum_{h'=h}^H \sum_{s',a',s''} \mu_{\mathcal{M}_2,\pi}^{h,h'}(s';s) \pi_{h'}(a'|s') \left| P_2(s''|s',a') - P_1(s''|s',a') \right| V_{\mathcal{M}_1 \cup r}^{\pi,h'+1}(s'')
\end{aligned}$$

Proof. We start by writing explicitly the value-functions:

$$\begin{aligned}
& V_{\mathcal{M}_2 \cup r}^{\pi, h}(s) - V_{\mathcal{M}_1 \cup r}^{\pi, h}(s) \\
&= \sum_{a, s'} \pi_h(a|s) \left(P_2(s'|s, a) V_{\mathcal{M}_2 \cup r}^{\pi, h+1}(s') - P_1(s'|s, a) V_{\mathcal{M}_1 \cup r}^{\pi, h+1}(s') \pm P_2(s'|s, a) V_{\mathcal{M}_1 \cup r}^{\pi, h+1}(s') \right) \\
&= \sum_{a, s'} \pi_h(a|s) \left((P_2(s'|s, a) - P_1(s'|s, a)) V_{\mathcal{M}_1 \cup r}^{\pi, h+1}(s') + P_2(s'|s, a) (V_{\mathcal{M}_2 \cup r}^{\pi, h+1}(s') - V_{\mathcal{M}_1 \cup r}^{\pi, h+1}(s')) \right)
\end{aligned}$$

Unrolling the recursion gives the first result; the second result follows similarly:

$$\begin{aligned}
& V_{\mathcal{M}_1 \cup r}^{\pi, h}(s) - V_{\mathcal{M}_2 \cup r}^{\pi, h}(s) \\
&= \sum_{a, s'} \pi_h(a|s) \left(P_1(s'|s, a) V_{\mathcal{M}_1 \cup r}^{\pi, h+1}(s') - P_2(s'|s, a) V_{\mathcal{M}_2 \cup r}^{\pi, h+1}(s') \pm P_2(s'|s, a) V_{\mathcal{M}_1 \cup r}^{\pi, h+1}(s') \right) \\
&= \sum_{a, s'} \pi_h(a|s) \left((P_1(s'|s, a) - P_2(s'|s, a)) V_{\mathcal{M}_1 \cup r}^{\pi, h+1}(s') + P_2(s'|s, a) (V_{\mathcal{M}_1 \cup r}^{\pi, h+1}(s') - V_{\mathcal{M}_2 \cup r}^{\pi, h+1}(s')) \right)
\end{aligned}$$

Together, the first two results imply the third one because all terms in the sums are non-negative. \square

Lemma B.1.4. Let $\mathcal{M}_1, \mathcal{M}_2$ be two $\text{MDP} \setminus \mathbb{R}$ with transition dynamics P_1, P_2 respectively, r a reward function, and π_1^*, π_2^* optimal policy in $\mathcal{M}_1 \cup r$ and $\mathcal{M}_2 \cup r$, respectively. Then, for any state s and time h :

$$\begin{aligned}
& V_{\mathcal{M}_1 \cup r}^{*, h}(s) - V_{\mathcal{M}_2 \cup r}^{*, h}(s) \\
&\leq \sum_{h'=h} \sum_{s', a', s''} \mu_{\mathcal{M}_2, \pi_1^*}^{h, h'}(s'; s) \pi_{1, h}^*(a'|s') (P_1(s''|s', a') - P_2(s''|s', a')) V_{\mathcal{M}_1 \cup r}^{*, h}(s'')
\end{aligned}$$

and

$$\begin{aligned}
& V_{\mathcal{M}_2 \cup r}^{*, h}(s) - V_{\mathcal{M}_1 \cup r}^{*, h}(s) \\
&\leq \sum_{h'=h} \sum_{s', a', s''} \mu_{\mathcal{M}_2, \pi_2^*}^{h, h'}(s'; s) \pi_{2, h}^*(a'|s') (P_2(s''|s', a') - P_1(s''|s', a')) V_{\mathcal{M}_2 \cup r}^{*, h}(s'')
\end{aligned}$$

Proof.

$$\begin{aligned}
& V_{\mathcal{M}_1 \cup r}^{*,h}(s) - V_{\mathcal{M}_2 \cup r}^{*,h}(s) \\
&= \sum_{a,s'} \left(\pi_{1,h}^*(a|s) P_1(s'|s,a) V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') - \pi_{2,h}^*(a|s) P_2(s'|s,a) V_{\mathcal{M}_2 \cup r}^{\pi_{2,h}^*,h+1}(s') \right. \\
&\quad \left. \pm \pi_{1,h}^*(a|s) P_2(s'|s,a) V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') \pm \pi_{1,h}^*(a|s) P_2(s'|s,a) V_{\mathcal{M}_2 \cup r}^{\pi_{2,h}^*,h+1}(s') \right) \\
&= \sum_{a,s'} \left(\pi_{1,h}^*(a|s) P_2(s'|s,a) (V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') - V_{\mathcal{M}_2 \cup r}^{\pi_{2,h}^*,h+1}(s')) \right. \\
&\quad + \pi_{1,h}^*(a|s) (P_1(s'|s,a) - P_2(s'|s,a)) V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') \\
&\quad \left. + (\pi_{1,h}^*(a|s) - \pi_{2,h}^*(a|s)) P_2(s'|s,a) V_{\mathcal{M}_2 \cup r}^{\pi_{2,h}^*,h+1}(s') \right) \\
&\leq \sum_{a,s'} \left(\pi_{1,h}^*(a|s) P_2(s'|s,a) (V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') - V_{\mathcal{M}_2 \cup r}^{\pi_{2,h}^*,h+1}(s')) \right. \\
&\quad \left. + \pi_{1,h}^*(a|s) (P_1(s'|s,a) - P_2(s'|s,a)) V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') \right)
\end{aligned}$$

where the last inequality uses that π^* is optimal for $\mathcal{M}_2 \cup r$. Unrolling the recursion gives the first result. A similar argument yields the second results:

$$\begin{aligned}
& V_{\mathcal{M}_2 \cup r}^{*,h}(s) - V_{\mathcal{M}_1 \cup r}^{*,h}(s) \\
&= \sum_{a,s'} \left(\pi_{2,h}^*(a|s) P_2(s'|s,a) V_{\mathcal{M}_2 \cup r}^{\pi_{2,h}^*,h+1}(s') - \pi_{1,h}^*(a|s) P_1(s'|s,a) V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') \right. \\
&\quad \left. \pm \pi_{2,h}^*(a|s) P_2(s'|s,a) V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') \right) \\
&= \sum_{a,s'} \left(\pi_{2,h}^*(a|s) P_2(s'|s,a) (V_{\mathcal{M}_2 \cup r}^{\pi_{2,h}^*,h+1}(s') - V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s')) \right. \\
&\quad \left. + \pi_{2,h}^*(a|s) P_2(s'|s,a) V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') - \pi_{1,h}^*(a|s) P_1(s'|s,a) V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') \right) \\
&\leq \sum_{a,s'} \left(\pi_{2,h}^*(a|s) P_2(s'|s,a) (V_{\mathcal{M}_2 \cup r}^{\pi_{2,h}^*,h+1}(s') - V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s')) \right. \\
&\quad \left. + \pi_{2,h}^*(a|s) (P_2(s'|s,a) - P_1(s'|s,a)) V_{\mathcal{M}_1 \cup r}^{\pi_{1,h}^*,h+1}(s') \right)
\end{aligned}$$

□

B.1.2 Feasible Reward Set

In this section, we characterize the feasible reward set first implicitly, then explicitly, and prove a result about error propagation. [72] provide a similar analysis in the infinite-horizon setting.

Lemma 4.2.1 (Feasible Reward Set Implicit). A reward function r is feasible if and only if for all s, a, h it holds that: $A_{\mathcal{M} \cup r}^{\pi, h}(s, a) = 0$ if $\pi_h^E(a|s) \geq 0$ and $A_{\mathcal{M} \cup r}^{\pi, h}(s, a) \leq 0$ if $\pi_h^E(a|s) = 0$. Moreover, if the second inequality is strict, π^E is uniquely optimal, i.e., $\Pi_{\mathcal{M} \cup r}^* = \{\pi^E\}$.

Proof. The result follows directly from Definition 4.2.1. \square

Lemma B.1.5. A Q-function satisfies the conditions of Lemma 4.2.1 if and only if there exists an $\{A_h \in \mathbb{R}_{\geq 0}^{|\mathcal{S}| \times |\mathcal{A}|}\}_{h \in H}$ and $\{V_h \in \mathbb{R}^{|\mathcal{S}|}\}$ such that for every $h, s, a \in [H] \times \mathcal{S} \times \mathcal{A}$:

$$Q_{\mathcal{M} \cup r}^{\pi^E, h}(s, a) = -A_h(s, a) \mathbb{1}_{\{\pi_h^E(a|s)=0\}} + V_h(s)$$

Proof. We first show that if $Q_{\mathcal{M} \cup r}^{\pi^E, h}(s, a)$ has this form, the conditions of Lemma 4.2.1 are satisfied, and then the converse. Assume $Q_{\mathcal{M} \cup r}^{\pi^E, h}(s, a) = -A_h(s, a) \mathbb{1}_{\{\pi_h^E(a|s)=0\}} + V_h(s)$. Then,

$$V_{\mathcal{M} \cup r}^{\pi^E, h}(s) = \sum_a \pi_h^E(a|s) Q_{\mathcal{M} \cup r}^{\pi^E, h}(s, a) = V_h(s).$$

If $\pi_h^E(a|s) > 0$, then $Q_{\mathcal{M} \cup r}^{\pi^E, h}(s, a) = V_{\mathcal{M} \cup r}^{\pi^E, h}(s)$, which is the first condition of Lemma 4.2.1. If $\pi_h^E(a|s) = 0$, $Q_{\mathcal{M} \cup r}^{\pi^E, h}(s, a) = V_{\mathcal{M} \cup r}^{\pi^E, h}(s) - A_h(s, a) \leq V_{\mathcal{M} \cup r}^{\pi^E, h}(s)$, which is the second condition of Lemma 4.2.1.

For the converse, assume that the conditions of Lemma 4.2.1 hold, and let $V_h(s) = V_{\mathcal{M} \cup r}^{\pi^E, h}(s)$ and $A_h(s, a) = V_{\mathcal{M} \cup r}^{\pi^E, h}(s) - Q_{\mathcal{M} \cup r}^{\pi^E, h}(s, a)$. \square

Lemma 4.2.2 (Feasible Reward Set Explicit). A reward function r is feasible if and only if there exists an $\{A_h \in \mathbb{R}_{\geq 0}^{|\mathcal{S}| \times |\mathcal{A}|}\}_{h \in [H]}$ and $\{V_h \in \mathbb{R}^{|\mathcal{S}|}\}_{h \in [H]}$ such that for all s, a, h it holds that:

$$r_h(s, a) = -A_h(s, a) \mathbb{1}_{\{\pi_h^E(a|s)=0\}} + V_h(s) + \sum_{s'} P(s'|s, a) V_{h+1}(s')$$

Proof.

Since $Q_{\mathcal{M} \cup r}^{\pi^E, h}(s, a) = r_h(s, a) + \sum_{s'} P(s'|s, a) V_{h+1}(s')$, we can use Lemma B.1.5 to see:

$$\begin{aligned} r_h(s, a) &= Q_{\mathcal{M} \cup r}^{\pi^E, h}(s, a) - \sum_{s'} P(s'|s, a) V_{h+1}(s') \\ &= -A_h(s, a) \mathbb{1}_{\{\pi_h^E(a|s)=0\}} + V_h(s) + \sum_{s'} P(s'|s, a) V_{h+1}(s') \end{aligned}$$

□

Theorem 4.2.1 (Error Propagation). Let (\mathcal{M}, π^E) and $(\widehat{\mathcal{M}}, \widehat{\pi}^E)$ be two IRL problems. Then, for any $r \in \mathcal{R}_{(\mathcal{M}, \pi^E)}$ there exists $\widehat{r} \in \widehat{\mathcal{R}}_{(\widehat{\mathcal{M}}, \widehat{\pi}^E)}$ such that:

$$\begin{aligned} |r_h(s, a) - \widehat{r}_h(s, a)| &\leq A_h(s, a) |\pi_h^E(a|s) - \widehat{\pi}_h^E(a|s)| \\ &\quad + \sum_{s'} V_{h+1}(s') |P(s'|s, a) - \widehat{P}(s'|s, a)| \end{aligned}$$

and we can bound $V_h \leq (H - h)r_{\max}$ and $A_h \leq (H - h)r_{\max}$.

Proof. We start by rewriting r and \widehat{r} using Lemma 4.2.2:

$$\begin{aligned} r_h(s, a) &= -A_h(s, a) \mathbb{1}_{\{\pi_h^E(a|s)=0\}} + V_h(s) + \sum_{s'} P(s'|s, a) V_{h+1}(s') \\ \widehat{r}_h(s, a) &= -\widehat{A}_h(s, a) \mathbb{1}_{\{\widehat{\pi}_h^E(a|s)=0\}} + \widehat{V}_h(s) + \sum_{s'} \widehat{P}(s'|s, a) \widehat{V}_{h+1}(s') \end{aligned}$$

We can choose (*w.l.o.g.*) $V_h = \widehat{V}_h$ and $\widehat{A}_h = A_h$:

$$\begin{aligned} r_h(s, a) - \widehat{r}_h(s, a) &= -A_h(s, a) \mathbb{1}_{\{\pi_h^E(a|s)=0\}} + V_h(s) + \sum_{s'} P(s'|s, a) V_{h+1}(s') \\ &\quad + A_h(s, a) \mathbb{1}_{\{\widehat{\pi}_h^E(a|s)=0\}} - V_h(s) - \sum_{s'} \widehat{P}(s'|s, a) V_{h+1}(s') \\ &= -A_h(s, a) (\mathbb{1}_{\{\pi_h^E(a|s)=0\}} - \mathbb{1}_{\{\widehat{\pi}_h^E(a|s)=0\}}) + \sum_{s'} V_{h+1}(s') (P(s'|s, a) - \widehat{P}(s'|s, a)) \end{aligned}$$

Note that $\mathbb{1}_{\{\pi_h^E(a|s)=0\}} \leq 1 - \pi_h^E(a|s)$ and $\mathbb{1}_{\{\widehat{\pi}_h^E(a|s)=0\}} \leq 1 - \widehat{\pi}_h^E(a|s)$, which can be easily checked for both cases of the indicator. Using this, we get

Algorithm 8 Uniform sampling IRL with a generative model.

Require: significance δ , target accuracy ϵ , samples per round n_{\max}

- 1: Initialize $k \leftarrow 0$, $\epsilon_0 \leftarrow H$
 - 2: **while** $\epsilon_k > \epsilon/2$ **do**
 - 3: Uniformly sample $\lceil \frac{n_{\max}}{|\mathcal{S}||\mathcal{A}|H} \rceil$ samples from all s, a, h
 - 4: For each sample, observe transition and expert action
 - 5: $k \leftarrow k + 1$
 - 6: Update \hat{P}_k , $\hat{\tau}_k$, and C_k^h
 - 7: Update accuracy $\epsilon_k \leftarrow H \max_{s,a,h} C_k^h(s, a)$
-

$$r_h(s, a) - \hat{r}_h(s, a) \leq A_h(s, a)(\pi_h^E(a|s) - \hat{\tau}_h^E(a|s)) + \sum_{s'} V_{h+1}(s')(P(s'|s, a) - \hat{P}(s'|s, a))$$

The result follows by taking the absolute value and applying the triangle inequality. \square

B.1.3 Uniform Sampling IRL With Generative Model

In this section, we derive sample complexity results for uniform sampling with a generative model (Algorithm 8). Metelli *et al.* [72] proved an analogous result for the infinite-horizon setting focusing on transferable rewards. In contrast, our focus is on the finite-horizon setting. Moreover, [72] considers to learn a reward that is transferable to a known target environment. In our setting, instead, we suppose to use the recovered reward function in the unknown source environment.

Definition 4.2.2 (Optimality Criterion). Let $\mathcal{R}_{\mathfrak{y}}$ be the exact feasible set and $\mathcal{R}_{\hat{\mathfrak{y}}}$ be the feasible set recovered after observing $n \geq 0$ samples collected from \mathcal{M} and π^E . We say that an algorithm for

Active IRL is (ϵ, δ, n) -correct if after n iterations with probability at least $1 - \delta$ it holds that:

$$\inf_{\hat{r} \in \mathcal{R}_{\mathfrak{B}}} \sup_{\hat{\pi}^* \in \Pi_{\widehat{\mathcal{M}} \cup \hat{r}}^*} \max_a \left| Q_{\mathcal{M} \cup r}^{\pi^*, 0}(s_0, a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, 0}(s_0, a) \right| \leq \epsilon \quad \text{for each } r \in \mathcal{R}_{\mathfrak{B}},$$

$$\inf_{r \in \mathcal{R}_{\mathfrak{B}}} \sup_{\pi^* \in \Pi_{\mathcal{M} \cup r}^*} \max_a \left| Q_{\mathcal{M} \cup r}^{\pi^*, 0}(s_0, a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, 0}(s_0, a) \right| \leq \epsilon \quad \text{for each } \hat{r} \in \mathcal{R}_{\mathfrak{B}},$$

where π^* is an optimal policy in $\mathcal{M} \cup r$ and $\hat{\pi}^*$ is an optimal policy in $\widehat{\mathcal{M}} \cup \hat{r}$.

Lemma B.1.6 (Good Event). Let π^E be a (possibly stochastic) expert policy. We estimate the expert policy with $\hat{\pi}^E$ and the transition model P with an estimate \hat{P}_k from k episodic interactions. Let $n_k^h(s, a)$ and $n_k^h(s)$ be the number of times state action pairs and states have been observed at time h within the first k episodes, and $n_k^{h+}(s, a) = \max\{1, n_k^h(s, a)\}$. Then,

$$\begin{aligned} |\pi_h^E(a|s) - \hat{\pi}_h^E(a|s)A_h(s, a)| &\leq (H - h)r_{\max} \sqrt{\frac{\ell_k^h(s, a)}{n_k^{h+}(s, a)}} \\ |\pi_h^E(a|s) - \hat{\pi}_h^E(a|s)\hat{A}_h(s, a)| &\leq (H - h)r_{\max} \sqrt{\frac{\ell_k^h(s, a)}{n_k^{h+}(s, a)}} \\ \sum_{s'} |(P(s'|s, a) - \hat{P}_k(s'|s, a))V_r^{\pi, h}(s')| &\leq (H - h)r_{\max} \sqrt{\frac{2\ell_k^h(s, a)}{n_k^{h+}(s, a)}} \\ \sum_{s'} |(P(s'|s, a) - \hat{P}_k(s'|s, a))\hat{V}_r^{\pi, h}(s')| &\leq (H - h)r_{\max} \sqrt{\frac{2\ell_k^h(s, a)}{n_k^{h+}(s, a)}} \end{aligned}$$

where $\ell_k^h(s, a) = \log\left(24|\mathcal{S}||\mathcal{A}|H(n_k^{h+}(s, a))^2/\delta\right)$, holds simultaneously for all $(s, a, h) \in \mathcal{S} \times \mathcal{A} \times [H]$ and $k \geq 1$ with probability at least $1 - \delta$. We call the event that these equations hold the *good event* \mathcal{E} and write $P(\mathcal{E}) \geq 1 - \delta$.

Proof. We show that each statement individually does not hold with probability less than $\delta/4$, which implies the result via a union bound.

Let us denote $\beta_1(s, a, h) := (H - h)r_{\max} \sqrt{\frac{2\ell_k^h(s, a)}{n_k^{h+}(s, a)}}$. First, consider the last two inequalities. The probability that either of them does not hold is:

$$\begin{aligned}
& \Pr \left(\exists k \geq 1, (s, a, h) \in \mathcal{S} \times \mathcal{A} \times [H] : \sum_{s'} |(P(s'|s, a) - \widehat{P}_k(s'|s, a))V_r^{\pi, h}(s')| > \beta_1(s, a, h) \right) \\
& \stackrel{(a)}{\leq} \Pr \left(\exists m \geq 0, (s, a, h) \in \mathcal{S} \times \mathcal{A} \times [H] : \sum_{s'} |(P(s'|s, a) - \widehat{P}_k(s'|s, a))V_r^{\pi, h}(s')| > \beta_1(s, a, h) \right) \\
& \stackrel{(b)}{\leq} \sum_{m \geq 0} \sum_{s, a} \sum_{h=0}^H \Pr \left(\sum_{s'} |(P(s'|s, a) - \widehat{P}_k(s'|s, a))V_r^{\pi, h}(s')| > \beta_1(s, a, h) \right) \\
& \stackrel{(c)}{\leq} \sum_{m \geq 0} \sum_{s, a} \sum_{h=0}^H 2 \exp \left(-\frac{2\beta_1(s, a, h)^2 m^2}{4m(H-h)^2 r_{\max}^2} \right) \leq \sum_{m \geq 0} \sum_{s, a} \sum_{h=0}^H 2 \exp(-\ell_k(s, a)) \\
& = \sum_{m \geq 0} \sum_{s, a} \sum_{h=0}^H \frac{2\delta}{24|\mathcal{S}||\mathcal{A}|H(m^+)^2} = \frac{\delta}{12} \left(1 + \sum_{m \geq 0} \frac{1}{m^2} \right) = \frac{\delta}{12} \left(1 + \frac{\pi^2}{6} \right) \leq \frac{\delta}{4}
\end{aligned}$$

Step (a) assumes that we visit a state action pair m times, and focuses on these m times the transition model for the given state-action pair is updated. Step (b) uses a union bound over m and (s, a) . Step (c) applies Hoeffding's inequality using that we estimate P with an average of samples, and $V_r^{\pi, h} \leq (H - h)r_{\max}$. The factor m^2 in the numerator results from dividing by $1/m$ to average over samples, and the factor $4m$ in the denominator results from the sum over m in the denominator of Hoeffding's bound.

We show the first two inequalities similarly, with

$$\beta_2(s, a, h) := (H - h)r_{\max} \sqrt{\frac{\ell_k^h(s, a)}{n_k^{h+}(s, a)}}$$

$$\begin{aligned}
& \Pr \left(\exists k \geq 1, (s, a, h) \in \mathcal{S} \times \mathcal{A} \times [H] : |(\pi_k^E(a|s) - \hat{\pi}_k^E(a|s))A_h(s, a)| > \beta_2(s, a, h) \right) \\
& \stackrel{(a)}{\leq} \Pr \left(\exists m \geq 0, (s, a, h) \in \mathcal{S} \times \mathcal{A} \times [H] : |(\pi_k^E(a|s) - \hat{\pi}_k^E(a|s))A_h(s, a)| > \beta_2(s, a, h) \right) \\
& \stackrel{(b)}{\leq} \sum_{m \geq 0} \sum_{s, a} \sum_{h=0}^H \Pr \left(|(\pi_k^E(a|s) - \hat{\pi}_k^E(a|s))A_h(s, a)| > \beta_2(s, a, h) \right) \\
& \stackrel{(c)}{\leq} \sum_{m \geq 0} \sum_{s, a} \sum_{h=0}^H 2 \exp \left(-\frac{2\beta_2(s, a, h)^2 m^2}{4m(H-h)^2 r_{\max}^2} \right) \leq \sum_{m \geq 0} \sum_{s, a} \sum_{h=0}^H 2 \exp(-\ell_k(s, a)) \\
& = \sum_{m \geq 0} \sum_{s, a} \sum_{h=0}^H \frac{2\delta}{24|\mathcal{S}||\mathcal{A}|H(m^+)^2} = \frac{\delta}{12} \left(1 + \sum_{m \geq 0} \frac{1}{m^2} \right) = \frac{\delta}{12} \left(1 + \frac{\pi^2}{6} \right) \leq \frac{\delta}{4} \\
& \text{A union bound over all equations results in } P(\mathcal{E}) \geq 1 - \delta. \quad \square
\end{aligned}$$

Definition B.1.2. We define the reward uncertainty as

$$C_k^h(s, a) = (H - h)r_{\max} \min \left(1, 2\sqrt{\frac{2\ell_k^h(s, a)}{n_k^h(s, a)}} \right)$$

Corollary B.1.1. Under the good event \mathcal{E} , in each iteration k it holds for all $(s, a, h) \in \mathcal{S} \times \mathcal{A} \times [H]$ that:

$$|r_h(s, a) - \hat{r}_h^k(s, a)| \leq C_k^h(s, a)$$

Proof.

$$\begin{aligned}
|r_h(s, a) - \hat{r}_h^k(s, a)| & \stackrel{(a)}{\leq} A_h(s, a) |\pi_h^E(a|s) - \hat{\pi}_h^E(a|s)| + \sum_{s'} V_{h+1}(s') |P(s'|s, a) - \hat{P}(s'|s, a)| \\
& \stackrel{(b)}{\leq} (H - h)r_{\max} \left(2\sqrt{\frac{2\ell_k^h(s, a)}{n_k^h(s, a)}} \right) = C_k^h(s, a)
\end{aligned}$$

where (a) uses Theorem 4.2.1 and (b) uses Lemma B.1.6. \square

Corollary B.1.2. Let \mathcal{S} be a sampling strategy. Let $\mathcal{R}_{\mathfrak{B}}$ be the exact feasible set and $\mathcal{R}_{\mathfrak{B}_k}$ be the feasible set recovered after k iterations. If

$$H \max_{s, a, h} C_k^h(s, a) \leq \frac{\epsilon}{2},$$

then the conditions of Definition 4.2.2 are satisfied.

Proof. For the first condition of Definition 4.2.2, observe:

$$\begin{aligned}
& \inf_{\hat{r} \in \mathcal{R}_{\mathfrak{B}_k}} \sup_{\hat{\pi}^* \in \Pi_{\mathcal{M} \cup r}^*} \max_{s,a,h} (Q_{\mathcal{M} \cup r}^{\pi^*,h}(s,a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*,h}(s,a)) \\
\stackrel{(a)}{\leq} & \inf_{\hat{r} \in \mathcal{R}_{\mathfrak{B}_k}} \sup_{\hat{\pi}^* \in \Pi_{\mathcal{M} \cup r}^*} \max_{s,a,h} \sum_{h'=h}^H \sum_{s',a'} \left(\mu_{\mathcal{M},\pi^*}^{h,h'}(s',a'|s,a) - \mu_{\mathcal{M},\hat{\pi}^*}^{h,h'}(s',a'|s,a) \right) (r_{h'}(s',a') - \hat{r}_{h'}(s',a')) \\
\stackrel{(b)}{\leq} & \inf_{\hat{r} \in \mathcal{R}_{\mathfrak{B}_k}} \sup_{\hat{\pi}^* \in \Pi_{\mathcal{M} \cup r}^*} \max_{s,a,h} \left| \sum_{h'=h}^H \sum_{s',a'} \left(\mu_{\mathcal{M},\pi^*}^{h,h'}(s',a'|s,a) - \mu_{\mathcal{M},\hat{\pi}^*}^{h,h'}(s',a'|s,a) \right) C_k^{h'}(s',a') \right| \\
\leq & 2H \max_{s,a,h} C_k^h(s,a)
\end{aligned}$$

where (a) uses Lemma 4.2.3 and (b) uses Corollary B.1.1.

For the second condition of Definition 4.2.2, it follows similarly that:

$$\inf_{r \in \mathcal{R}_{\mathfrak{B}}} \sup_{\pi^* \in \Pi_{\mathcal{M} \cup r}^*} \max_{s,a,h} (Q_{\mathcal{M} \cup r}^{\pi^*,h}(s,a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*,h}(s,a)) \leq 2H \max_{s,a,h} C_k^h(s,a)$$

Hence, if $H \max_{s,a,h} C_k^h(s,a) \leq \epsilon/2$, both conditions of Definition 4.2.2 are satisfied. \square

Theorem B.1.1 (Sample Complexity of Uniform Sampling IRL). With probability at least $1 - \delta$, Algorithm 8 stops at iteration τ fulfilling Definition 4.2.2 with a number of samples upper bounded by:

$$n \leq \tilde{\mathcal{O}} \left(\frac{H^5 r_{\max}^2 |\mathcal{S}| |\mathcal{A}|}{\epsilon^2} \right)$$

Proof. First, note

$$H \max_{s,a,h} C_k^h(s,a) = H^2 r_{\max} \max_{s,a,h} \left(2 \sqrt{\frac{2\ell_k^h(s,a)}{n_k^{h+}(s,a)}} \right)$$

After τ iterations, we have collected $\tau \cdot n_{\max}$ samples and for each s, a, h , we have: $n_{\tau}^{h+}(s,a) \geq \frac{\tau n_{\max}}{|\mathcal{S}| |\mathcal{A}| H} \geq 1$

To terminate at iteration τ , we need to have for all s, a, h :

$$2H^2 r_{\max} \sqrt{\frac{2\ell_{\tau}^h(s,a)}{n_{\tau}^h(s,a)}} \leq \frac{\epsilon}{2}$$

which implies

$$n_{\tau}^h(s, a) \geq \frac{32H^4 r_{\max}^2 \ell_{\tau}^h(s, a)}{\epsilon^2}$$

By using Lemma B.8 by [72], we can conclude that the number of samples necessary to ensure accuracy ϵ is:

$$n \leq \tilde{O} \left(\frac{H^5 r_{\max}^2 |\mathcal{S}| |\mathcal{A}|}{\epsilon^2} \right)$$

□

Corollary B.1.3. If the true reward function does not depend on the timestep h , i.e., $r_h(s, a) = r(s, a)$, then we can modify Algorithm 8 to only need $n \leq \tilde{O} \left(\frac{H^4 r_{\max}^2 |\mathcal{S}| |\mathcal{A}|}{\epsilon^2} \right)$ samples.

Proof. If we know that the reward function does not depend on h we can choose $C_k(s, a) = \min_h C_k^h(s, a)$ as a confidence interval of the reward. Consequently, we can sample all states for a fixed h .

We still need for all s, a :

$$2H^2 r_{\max} \sqrt{\frac{2\ell_{\tau}^h(s, a)}{n_{\tau}^h(s, a)}} \leq \frac{\epsilon}{2} \Rightarrow n_{\tau}^h(s, a) \geq \frac{32H^4 r_{\max}^2 \ell_{\tau}^h(s, a)}{\epsilon^2}$$

Again, we use Lemma B.8 by [72], but we can eliminate one sum over H , ending up with:

$$n \leq \tilde{O} \left(\frac{H^4 r_{\max}^2 |\mathcal{S}| |\mathcal{A}|}{\epsilon^2} \right)$$

□

B.1.4 AceIRL: Problem Independent Analysis

We are now ready to analyze the sample complexity of AceIRL (Algorithm 3). We first consider the simple version of the algorithm: AceIRL Greedy. Then, we consider the full version of the algorithm

after introducing a few additional lemma about the policy confidence set. We start by defining the error upper bound and deriving two lemmas that will help us to show that it is indeed an upper bound on the error we want to reduce.

Definition B.1.3. We define recursively:

$$E_k^H(s, a) = 0$$

$$E_k^h(s, a) = \min\left((H - h)r_{\max}, C_k^h(s, a) + \sum_{s'} \widehat{P}(s'|s, a) \max_{a' \in \mathcal{A}} E_k^{h+1}(s', a')\right)$$

where \widehat{P} is the estimated transition model of the environment.

The first lemma shows that the error upper bound can upper bound the error due to estimating the transition model.

Lemma B.1.7. Under the good event \mathcal{E} , for all policies π and reward functions r and all s, a, h :

$$|Q_{\widehat{\mathcal{M}} \cup r}^{\pi, h}(s, a) - Q_{\mathcal{M} \cup r}^{\pi, h}(s, a)| \leq E_k^h(s, a)$$

Proof.

$$\begin{aligned} & |Q_{\widehat{\mathcal{M}} \cup r}^{\pi, h}(s, a) - Q_{\mathcal{M} \cup r}^{\pi, h}(s, a)| = \left| \sum_{s'} \widehat{P}(s'|s, a) \sum_{a'} \pi(a'|s') Q_{\widehat{\mathcal{M}} \cup r}^{\pi, h+1}(s', a') \right. \\ & \left. - \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') Q_{\mathcal{M} \cup r}^{\pi, h+1}(s', a') \pm \sum_{s'} \widehat{P}(s'|s, a) \sum_{a'} \pi(a'|s') Q_{\mathcal{M} \cup r}^{\pi, h+1}(s', a') \right| \\ & \leq \left| \sum_{s'} (\widehat{P}(s'|s, a) - P(s'|s, a)) \sum_{a'} \pi(a'|s') Q_{\mathcal{M} \cup r}^{\pi, h+1}(s', a') \right| \\ & + \sum_{s'} \widehat{P}(s'|s, a) \sum_{a'} \pi(a'|s') \left| Q_{\widehat{\mathcal{M}} \cup r}^{\pi, h+1}(s, a) - Q_{\mathcal{M} \cup r}^{\pi, h+1}(s, a) \right| \\ & \leq C_k^h(s, a) + \sum_{s'} \widehat{P}(s'|s, a) \sum_{a'} \pi(a'|s') \left| Q_{\widehat{\mathcal{M}} \cup r}^{\pi, h+1}(s, a) - Q_{\mathcal{M} \cup r}^{\pi, h+1}(s, a) \right| \end{aligned}$$

For $h = H$ the result holds trivially. Now assuming it holds for $h + 1$, we consider step h :

$$\begin{aligned}
& |Q_{\widehat{\mathcal{M}}_{Ur}}^{\pi, h}(s, a) - Q_{\mathcal{M}_{Ur}}^{\pi, h}(s, a)| \\
& \leq C_k^h(s, a) + \sum_{s'} \widehat{P}(s'|s, a) \sum_{a'} \pi(a'|s') \left| Q_{\widehat{\mathcal{M}}_{Ur}}^{\pi, h+1}(s, a) - Q_{\mathcal{M}_{Ur}}^{\pi, h+1}(s, a) \right| \\
& \leq C_k^h(s, a) + \sum_{s'} \widehat{P}(s'|s, a) \max_{a'} \left| Q_{\widehat{\mathcal{M}}_{Ur}}^{\pi, h+1}(s, a) - Q_{\mathcal{M}_{Ur}}^{\pi, h+1}(s, a) \right| \\
& \leq C_k^h(s, a) + \sum_{s'} \widehat{P}(s'|s, a) \max_{a'} E_k^{h+1}(s', a') = E_k^h(s, a)
\end{aligned}$$

□

The next lemma shows that the error upper bound can also upper bound the error in estimating the reward function, which is due to estimating the transition model and the expert policy.

Lemma B.1.8. Under the good event \mathcal{E} , for all reward function r , all policies π , and all $s, a \in \mathcal{S} \times \mathcal{A}$:

$$|Q_{\widehat{\mathcal{M}}_{U\hat{r}}}^{\pi, h}(s, a) - Q_{\mathcal{M}_{Ur}}^{\pi, h}(s, a)| \leq E_k^h(s, a)$$

Proof. For $h = H$ the result holds trivially. Now assuming it holds for $h + 1$, we consider step h :

$$\begin{aligned}
& |Q_{\widehat{\mathcal{M}}_{U\hat{r}}}^{\pi, h}(s, a) - Q_{\mathcal{M}_{Ur}}^{\pi, h}(s, a)| \\
& \leq |\hat{r}(s, a) - r(s, a)| + \sum_{s'} \widehat{P}(s'|s, a) \sum_{a'} \pi(a'|s') |Q_{\widehat{\mathcal{M}}_{U\hat{r}}}^{\pi, h+1}(s', a') - Q_{\mathcal{M}_{Ur}}^{\pi, h+1}(s', a')| \\
& \leq |\hat{r}(s, a) - r(s, a)| + \sum_{s'} \widehat{P}(s'|s, a) \max_{a'} |Q_{\widehat{\mathcal{M}}_{U\hat{r}}}^{\pi, h+1}(s', a') - Q_{\mathcal{M}_{Ur}}^{\pi, h+1}(s', a')| \\
& \leq |\hat{r}(s, a) - r(s, a)| + \sum_{s'} \widehat{P}(s'|s, a) \max_{a'} E_k^{h+1}(s', a') = E_k^h(s, a)
\end{aligned}$$

□

We can now combine the previous two lemmas to show that E is indeed an upper bound on the error we want to reduce. This implies correctness of AceIRL Greedy, which the following lemma formalizes.

Lemma B.1.9 (Correctness of AceIRL Greedy). If AceIRL Greedy stops in episode k , after sampling n samples, i.e., $E_k^0(s_0, \pi_{k+1}(s_0)) \leq \frac{\epsilon}{4}$, then it fulfills Definition 4.2.2.

Proof. Let us define the error

$$e_k^h(s, a) := |Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a)|$$

where π^* is the true optimal policy in $\mathcal{M} \cup r$, and $\hat{\pi}^*$ is the optimal policy in $\widehat{\mathcal{M}} \cup \widehat{r}$, i.e., in the estimated MDP using the inferred reward function. Then,

$$\begin{aligned} e_k^h(s, a) &= |Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a) \pm Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\pi^*, h}(s, a) \pm Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a)| \\ &\leq \underbrace{|Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\pi^*, h}(s, a)|}_{\leq E_k^h(s, a)} + |Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a)| + \underbrace{|Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, h}(s, a)|}_{\leq E_k^h(s, a)} \\ &\leq 2E_k^h(s, a) + |Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a)| \\ &\quad \text{where, we used Lemma B.1.7.} \end{aligned}$$

Let us consider the remaining term $|Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a)|$ in two steps. First, we have:

$$\begin{aligned} Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a) &\leq \underbrace{Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\pi^*, h}(s, a)}_{\leq E_k^h(s, a)} + \underbrace{Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a)}_{\leq 0} \\ &\quad + \underbrace{Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, h}(s, a)}_{\leq E_k^h(s, a)} \leq 2E_k^h(s, a), \end{aligned}$$

where we used Lemma B.1.8 and the fact that $\hat{\pi}^*$ is optimal in the MDP $\widehat{\mathcal{M}} \cup \widehat{r}$. Second, we have:

$$\begin{aligned} Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a) - Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) &\leq \underbrace{Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, h}(s, a)}_{\leq E_k^h(s, a)} + \underbrace{Q_{\mathcal{M} \cup r}^{\hat{\pi}^*, h}(s, a) - Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a)}_{\leq 0} \\ &\quad + \underbrace{Q_{\mathcal{M} \cup r}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\pi^*, h}(s, a)}_{\leq E_k^h(s, a)} \leq 2E_k^h(s, a), \end{aligned}$$

where we used Lemma B.1.7 and the fact that π^* is optimal in the MDP $\mathcal{M} \cup r$. Overall, we find that

$$|Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\pi^*, h}(s, a) - Q_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\hat{\pi}^*, h}(s, a)| \leq 2E_k^h(s, a),$$

and consequently,

$$e_k^h(s, a) \leq 4E_k^h(s, a).$$

Hence, if $E_k^0(s_0, \pi_{k+1}(s_0)) \leq \frac{\epsilon}{4}$, we have for all $a \in \mathcal{A}$:

$$e_k^0(s_0, a) \leq \epsilon$$

which implies correctness according to Definition 4.2.2. \square

Next, we will analyze the sample complexity of AceIRL Greedy. Let us first define pseudo-counts that will be crucial to deal with the uncertainty of the transition dynamics in our analysis. This is similar to the analysis of UCRL for reward-free exploration by Kaufmann *et al.* [85].

Definition B.1.4. We define the *pseudo-counts* of visiting a specific state action pair at timestep h within the first k iterations as

$$\bar{n}_k^h(s, a) := \sum_{i=1}^k \mu_{\mathcal{M}, \pi_i}^{0, h}(s, a | s_0),$$

where π_i is the exploration policy in episode i .

The following lemma allows us to introduce the pseudo-counts when considering the contraction of the reward confidence intervals.

Lemma B.1.10. With probability at least $1 - \frac{\delta}{2}$ for all $s, a, h, k \in \mathcal{S} \times \mathcal{A} \times [H] \times \mathbb{N}^+$, we have:

$$\min \left(\frac{2\ell_k^h(s, a)}{n_k^h(s, a)}, 1 \right) \leq \frac{8\bar{\ell}_k^h(s, a)}{\max(\bar{n}_k^h(s, a), 1)}$$

where $\bar{\ell}_k^h(s, a) = \log(24|\mathcal{S}||\mathcal{A}|H(\bar{n}_k^h(s, a))^2/\delta)$.

Proof. This result adapts Lemma 7 by Kaufmann *et al.* [85] to our setting.

By Lemma 10 in Kaufmann *et al.* [85], we have with probability at least $1 - \frac{\delta}{2}$:

$$n_k^h(s, a) \geq \frac{1}{2}\bar{n}_k^h(s, a) - \beta_{\text{cnt}}(\delta),$$

where $\beta_{\text{cnt}}(\delta) = \log(2|\mathcal{S}||\mathcal{A}|H/\delta)$.

We distinguish two cases. First let $\beta_{\text{cnt}}(\delta) \leq \frac{1}{4}\bar{n}_k^h(s, a)$. Then $n_k^h(s, a) \geq \frac{1}{4}\bar{n}_k^h(s, a)$, and

$$\begin{aligned} \min \left(\frac{2\ell_k^h(s, a)}{n_k^h(s, a)}, 1 \right) &\leq \frac{2\ell_k^h(s, a)}{\max(n_k^h(s, a), 1)} = \frac{2\log(24|\mathcal{S}||\mathcal{A}|H(n_k^h(s, a))^2/\delta)}{\max(n_k^h(s, a), 1)} \\ &\leq \frac{2\log(24|\mathcal{S}||\mathcal{A}|H(\bar{n}_k^h(s, a)/4)^2/\delta)}{(\bar{n}_k^h(s, a)/4)} \leq \frac{8\bar{\ell}_k^h(s, a)}{\max(\bar{n}_k^h(s, a), 1)} \end{aligned}$$

where we use that $\log(24|\mathcal{S}||\mathcal{A}|Hx^2/\delta)/x$ is non-increasing for $x > 1$, and $\log(24|\mathcal{S}||\mathcal{A}|Hx^2/\delta)$ is non-decreasing and $\beta_{\text{cnt}}(\delta) \geq 1$.

Now consider let $\beta_{\text{cnt}}(\delta) > \frac{1}{4}\bar{n}_k^h(s, a)$. Then,

$$\min \left(\frac{2\ell_k^h(s, a)}{n_k^h(s, a)}, 1 \right) \leq 1 < 4 \frac{\beta_{\text{cnt}}(\delta)}{\max(\bar{n}_k^h(s, a), 1)} \leq \frac{4\bar{\ell}_k^h(s, a)}{\max(\bar{n}_k^h(s, a), 1)}$$

where we used that $\ell_k^h(s, a) = \log(24|\mathcal{S}||\mathcal{A}|H(n_k^h(s, a))^2/\delta) = \beta_{\text{cnt}}(\delta) + \log(6n_k^h(s, a))^2 \geq \beta_{\text{cnt}}(\delta)$. \square

The final lemma we need shows relates the error upper bound which is defined using our estimated transition model to a similar quantity defined using the (unknown) real transitions.

Lemma B.1.11. Under the good event \mathcal{E} , we have for any s, a, h :

$$E_k^h(s, a) \leq 2C_k^h(s, a) + \sum_{s'} P(s'|s, a) \max_{a'} E_k^{h+1}(s', a')$$

where P is the true transition model that we do not know.

Proof. First note that $E_k^h(s, a) \leq H$ by definition. Now, consider:

$$\begin{aligned} E_k^h(s, a) &\leq C_k^h(s, a) + \sum_{s'} \widehat{P}(s'|s, a) \max_{a'} E_k^{h+1}(s', a') \\ &= C_k^h(s, a) + \sum_{s'} (\widehat{P}(s'|s, a) - P(s'|s, a) + P(s'|s, a)) \max_{a'} E_k^{h+1}(s', a') \\ &= C_k^h(s, a) + \underbrace{\sum_{s'} (\widehat{P}(s'|s, a) - P(s'|s, a)) \max_{a'} E_k^{h+1}(s', a')}_{\leq C_k^h(s, a)} + \sum_{s'} P(s'|s, a) \max_{a'} E_k^{h+1}(s', a') \\ &\leq 2C_k^h(s, a) + \sum_{s'} P(s'|s, a) \max_{a'} E_k^{h+1}(s', a') \end{aligned}$$

where we used the good event and the fact that C_k^h can only shrink over episodes. \square

Finally, we can analyze the sample complexity of AceIRL Greedy.

Theorem B.1.2 (AceIRL Greedy Sample Complexity (problem independent)). AceIRL Greedy terminates with an (ϵ, δ, n) -correct solution, with

$$n \leq \tilde{\mathcal{O}} \left(\frac{H^5 r^2 \max_{\mathcal{S}} |\mathcal{S}| |\mathcal{A}|}{\epsilon^2} \right).$$

Proof. Lemma B.1.9 shows that if AceIRL Greedy terminates, then it returns a (ϵ, δ, n) -correct solution. So, we need to show that it terminates within τ iterations and bound τ .

Let us consider the average error, defined by

$$\begin{aligned} q_k^h &:= \sum_{s,a} \mu_{\mathcal{M}, \pi_{k+1}}^{0,h}(s, a | s_0) E_k^h(s, a) \\ &\stackrel{(a)}{\leq} \sum_{s,a} \mu_{\mathcal{M}, \pi_{k+1}}^{0,h}(s, a | s_0) (2C_k^h(s, a) + \sum_{s'} P(s' | s, a) \max_{a'} E_k^{h+1}(s', a')) \\ &= \sum_{s,a} \mu_{\mathcal{M}, \pi_{k+1}}^{0,h}(s, a | s_0) (2C_k^h(s, a) + \sum_{s'} P(s' | s, a) \sum_{a'} \pi_{k+1}(a' | s') E_k^{h+1}(s', a')) \\ &= 2 \sum_{s,a} \mu_{\mathcal{M}, \pi_{k+1}}^{0,h}(s, a | s_0) C_k^h(s, a) + q_k^{h+1} \end{aligned}$$

where we used Lemma B.1.11 in step (a). Unrolling the recursion, results in:

$$q_k^h \leq 2 \sum_{h'=h}^H \sum_{s,a} \mu_{\mathcal{M}, \pi_{k+1}}^{0,h'}(s, a | s_0) C_k^{h'}(s, a)$$

If the algorithm terminates at τ , we have for each $k < \tau$, and $s, a, h \in \mathcal{S} \times \mathcal{A} \times [H]$: $\epsilon < 4E_k^0(s_0, \pi_{k+1}(s_0))$.

We have $q_k^0 = E_k^0(s_0, \pi_{k+1}(s_0))$; therefore, as long we haven't stopped, we have $\epsilon \leq 4q_k^0$. Writing out this inequality, yields:

$$\epsilon \leq 4q_k^0 \leq 8 \sum_{h=0}^H \sum_{s,a} \mu_{\mathcal{M}, \pi_{k+1}}^{0,h}(s, a | s_0) C_k^h(s, a) \leq 4Hr_{\max} \sum_{h=0}^H \sum_{s,a} \mu_{\mathcal{M}, \pi_{k+1}}^{0,h}(s, a | s_0) \sqrt{\frac{8 \log(12|\mathcal{S}||\mathcal{A}|H(n_k^h(s, a))^2/\delta)}{\max(n_k^h(s, a), 1)}}$$

Using Lemma B.1.10, we can relate this to the pseudo-counts

$$\begin{aligned} \epsilon &< 4Hr_{\max} \sum_{h=0}^H \sum_{s,a} \mu_{\mathcal{M}, \pi_{k+1}}^{0,h}(s, a | s_0) \sqrt{\frac{8 \log(12|\mathcal{S}||\mathcal{A}|H(\bar{n}_k^h(s, a))^2/\delta)}{\max(\bar{n}_k^h(s, a), 1)}} \\ &\leq 4Hr_{\max} \sum_{h=0}^H \sum_{s,a} \mu_{\mathcal{M}, \pi_{k+1}}^{0,h}(s, a | s_0) \sqrt{\frac{8 \log(12|\mathcal{S}||\mathcal{A}|Hk^2/\delta)}{\max(\bar{n}_k^h(s, a), 1)}} \end{aligned}$$

Summing the inequality over $k = 0, \dots, T$ with $T < \tau$, we obtain

$$\begin{aligned} \epsilon(T+1) &\leq 4Hr_{\max} \sqrt{8 \log(12|\mathcal{S}||\mathcal{A}|HT^2/\delta)} \sum_{h=0}^H \sum_{s,a} \sum_{k=1}^T \mu_{\mathcal{M}, \tau_{k+1}}^{0,h}(s, a|s_0) \frac{1}{\sqrt{\max(\bar{n}_k^h(s, a), 1)}} \\ &= 4Hr_{\max} \sqrt{8 \log(12|\mathcal{S}||\mathcal{A}|HT^2/\delta)} \sum_{h=0}^H \sum_{s,a} \sum_{k=1}^T \frac{\bar{n}_h^{k+1}(s, a) - \bar{n}_h^k(s, a)}{\sqrt{\max(\bar{n}_k^h(s, a), 1)}} \end{aligned}$$

where we used the definition of the pseudo-counts in the last equality. Using Lemma 19 by [143], we can further bound the sum in k :

$$\begin{aligned} \epsilon(T+1) &= 4Hr_{\max} \sqrt{8 \log(12|\mathcal{S}||\mathcal{A}|HT^2/\delta)} \sum_{h=0}^H \sum_{s,a} \sqrt{\bar{n}_h^{T+1}(s, a)} \\ &\leq 4Hr_{\max} \sqrt{8 \log(12|\mathcal{S}||\mathcal{A}|HT^2/\delta)} \sqrt{|\mathcal{S}||\mathcal{A}|} \sum_{h=0}^H \sqrt{\sum_{s,a} \bar{n}_h^{T+1}(s, a)} \\ &= 4H^2r_{\max} \sqrt{8 \log(12|\mathcal{S}||\mathcal{A}|HT^2/\delta)} \sqrt{|\mathcal{S}||\mathcal{A}|} \sqrt{T+1} \end{aligned}$$

It follows that

$$\begin{aligned} \epsilon\sqrt{T+1} &\leq 4H^2r_{\max} \sqrt{8|\mathcal{S}||\mathcal{A}| \log(12|\mathcal{S}||\mathcal{A}|HT^2/\delta)} \\ \epsilon^2\tau &\leq 128H^4r_{\max}^2 |\mathcal{S}||\mathcal{A}| \log(12|\mathcal{S}||\mathcal{A}|H(\tau-1)^2/\delta) \end{aligned}$$

setting $\tau = T + 1$.

For large enough τ , this inequality cannot hold because $\sqrt{T+1}$ on the l.h.s grows faster than $\log(\tau)$ on the r.h.s. Hence, the stopping time τ is finite. Further, we can apply Lemma 15 by [85], and follow that

$$\tau \leq \tilde{\mathcal{O}} \left(\frac{H^4 r_{\max}^2 |\mathcal{S}||\mathcal{A}|}{\epsilon^2} \right)$$

If we observe H samples in each iteration, i.e., $N_E = 1$, we get a sample complexity of

$$n \leq \tilde{\mathcal{O}} \left(\frac{H^5 r_{\max}^2 |\mathcal{S}||\mathcal{A}|}{\epsilon^2} \right)$$

□

B.1.5 *AceIRL: Problem Dependent Analysis*

For the problem dependent analysis, we will need this additional lemma also used by Kakade and Langford [144].

Lemma B.1.12 (Lemma 6.1 by Kakade and Langford [144]). For any policy π :

$$V_{\mathcal{M} \cup r}^{\pi^*, h}(s) - V_{\mathcal{M} \cup r}^{\pi, h}(s) = - \sum_{s', a'} \sum_{h'=h}^H \mu_{\mathcal{M}, \pi}^{h, h'}(s', a'; s) A_{\mathcal{M} \cup r}^{*, h'}(s', a')$$

Proof.

$$\begin{aligned} V_{\mathcal{M} \cup r}^{*, h}(s) - V_{\mathcal{M} \cup r}^{\pi, h}(s) &= \sum_a \pi_h^*(a|s) \left(r_h(s, a) + \sum_{s'} P(s'|s, a) V_{\mathcal{M} \cup r}^{*, h+1}(s') \right) \\ &\quad - \sum_a \pi_h(a|s) \left(r_h(s, a) + \sum_{s'} P(s'|s, a) V_{\mathcal{M} \cup r}^{\pi, h+1}(s') \right) \\ &\quad \pm \sum_{a, s'} \pi_h(a|s) P(s'|s, a) V_{\mathcal{M} \cup r}^{*, h+1}(s') \\ &= \sum_a (\pi_h^*(a|s) - \pi_h(a|s)) r(s, a) \\ &\quad + \sum_{a, s'} (\pi_h^*(a|s) - \pi_h(a|s)) P(s'|s, a) V_{\mathcal{M} \cup r}^{*, h+1}(s') \\ &\quad + \sum_{a, s'} \pi_h(a|s) P(s'|s, a) (V_{\mathcal{M} \cup r}^{*, h+1}(s) - V_{\mathcal{M} \cup r}^{\pi, h+1}(s)) \\ &= - \sum_a \pi(a|s) A_{\mathcal{M} \cup r}^{*, h}(s, a) + \sum_{a, s'} \pi_h(a|s) P(s'|s, a) (V_{\mathcal{M} \cup r}^{*, h+1}(s) - V_{\mathcal{M} \cup r}^{\pi, h+1}(s)) \end{aligned}$$

Unrolling the recursion yields the result. \square

We can now start with the analysis. First, we define the policy confidence set, and show that it indeed contains the relevant policies under the good event.

Definition B.1.5. We define the *policy confidence set* as

$$\widehat{\Pi}_k = \{ \pi | V_{\widehat{\mathcal{M}} \cup \widehat{r}}^*(s_0) - V_{\widehat{\mathcal{M}} \cup \widehat{r}}^{\pi}(s_0) \leq 10\epsilon_k \}$$

where $\hat{r} = \mathcal{A}(\mathcal{R}_{\hat{g}})$ is the reward estimated using an IRL algorithm \mathcal{A} . We choose ϵ_k recursively by solving the optimization problem

$$\epsilon_k = \max_{\pi \in \hat{\Pi}_{k-1}} \sum_{h=0}^H \sum_{s', a'} \mu_{\widehat{\mathcal{M}}, \pi}^{0, h}(s', a'; s_0) C_k^h(s', a')$$

starting with $\epsilon_0 = \frac{1}{10}H$.

The following lemma will help us to deal with uncertainty about the transition dynamics.

Lemma B.1.13. Under the good event \mathcal{E} , if $\pi \in \hat{\Pi}_k$, then:

$$\begin{aligned} |V_{\widehat{\mathcal{M}} \cup \hat{r}}^{\pi, h}(s) - V_{\mathcal{M} \cup \hat{r}}^{\pi, h}(s)| &\leq \epsilon_k \\ |V_{\widehat{\mathcal{M}} \cup \hat{r}}^{*, h}(s) - V_{\mathcal{M} \cup \hat{r}}^{*, h}(s)| &\leq \epsilon_k \end{aligned}$$

Proof. First by Lemma B.1.3:

$$\begin{aligned} &|V_{\widehat{\mathcal{M}} \cup r}^{\pi, h}(s) - V_{\mathcal{M} \cup r}^{\pi, h}(s)| \\ &\leq \sum_{h'=h}^H \sum_{s', a', s''} \mu_{\widehat{\mathcal{M}}, \pi}^{h, h'}(s'; s) \pi_{h'}(a' | s') |\widehat{P}(s'' | s', a') - P(s'' | s', a')| V_{\mathcal{M} \cup r}^{\pi, h'+1}(s'') \\ &\leq \sum_{h'=h}^H \sum_{s', a'} \mu_{\widehat{\mathcal{M}}, \pi}^{h, h'}(s'; s) \pi_{h'}(a' | s') C_k(s', a') \leq \epsilon_k \end{aligned}$$

Then, by Lemma B.1.4:

$$\begin{aligned} &V_{\mathcal{M} \cup r}^{*, h}(s) - V_{\widehat{\mathcal{M}} \cup r}^{*, h}(s) \\ &\leq \sum_{h'=h}^H \sum_{s', a', s''} \mu_{\widehat{\mathcal{M}}, \pi^*}^{h, h'}(s'; s) \pi_{h'}^*(a' | s') (P(s'' | s', a') - \widehat{P}(s'' | s', a')) V_{\mathcal{M} \cup r}^{*, h'}(s'') \\ &\leq \sum_{h'=h}^H \sum_{s', a'} \mu_{\widehat{\mathcal{M}}, \pi^*}^{h, h'}(s'; s) \pi_{h'}^*(a' | s') C_k(s', a') \leq \epsilon_k \end{aligned}$$

And, similarly

$$\begin{aligned}
& V_{\widehat{\mathcal{M}}_{\mathcal{U}r}}^{*,h}(s) - V_{\mathcal{M}_{\mathcal{U}r}}^{*,h}(s) \\
& \leq \sum_{h'=h} \sum_{s',a',s''} \mu_{\widehat{\mathcal{M}},\widehat{\pi}^*}^{h,h'}(s';s) \widehat{\pi}_{h'}^*(a'|s') (\widehat{P}(s''|s',a') - P(s''|s',a')) V_{\widehat{\mathcal{M}}_{\mathcal{U}r}}^{*,h}(s'') \\
& \leq \sum_{h'=h} \sum_{s',a'} \mu_{\widehat{\mathcal{M}},\widehat{\pi}^*}^{h,h'}(s';s) \widehat{\pi}_{h'}^*(a'|s') C_k(s',a') \leq \epsilon_k
\end{aligned}$$

□

Now we show that the relevant policies are always in the policy confidence set, conditioned on the good event.

Lemma B.1.14. Conditioned the good event \mathcal{E} , if $\pi^*, \widehat{\pi}^* \in \widehat{\Pi}_{k-1}$, then $\pi^* \in \widehat{\Pi}_k$.

Proof. Let $r \in \mathcal{R}_{\mathfrak{B}}$. Then

$$\begin{aligned}
& V_{\widehat{\mathcal{M}} \cup \widehat{r}_k}^{*,h}(s) - V_{\widehat{\mathcal{M}} \cup r_k}^{\pi^*,h}(s) = V_{\widehat{\mathcal{M}} \cup \widehat{r}_k}^{*,h}(s) - V_{\widehat{\mathcal{M}} \cup r}^{*,h}(s) + V_{\widehat{\mathcal{M}} \cup r}^{*,h}(s) - V_{\widehat{\mathcal{M}} \cup \widehat{r}_k}^{\pi^*,h}(s) \\
& \stackrel{(a)}{\leq} \sum_{h'=h}^H \sum_{s',a'} \mu_{\widehat{\mathcal{M}},\pi^*}^{h,h'}(s',a'|s) C_k^{h'}(s',a') + \sum_{h'=h}^H \sum_{s',a'} \mu_{\widehat{\mathcal{M}},\pi^*}^{h,h'}(s',a'|s) C_k^{h'}(s',a') \stackrel{(b)}{\leq} 2\epsilon_k
\end{aligned}$$

where (a) uses Lemma B.1.1, Lemma B.1.2 and Corollary B.1.1, (b) uses that $\pi^* \in \widehat{\Pi}_{k-1}$ and the definition of ϵ_k . Hence,

$$\max_s \left(V_{\widehat{\mathcal{M}} \cup \widehat{r}_k}^{*,h}(s) - V_{\widehat{\mathcal{M}} \cup r_k}^{\pi^*,h}(s) \right) \leq 2\epsilon_k \leq 10\epsilon_k$$

and therefore $\pi^* \in \widehat{\Pi}_k$. □

Lemma B.1.15. Conditioned on the good event \mathcal{E} , for every policy π and episodes $k' > k$, there exists $\widehat{r}_{k'} \in \mathcal{R}_{\mathfrak{B}^{k'}}$, such that:

$$\max_s \left(V_{\widehat{\mathcal{M}} \cup \widehat{r}_{k'}}^{\pi,h}(s) - V_{\mathcal{M} \cup r_k}^{\pi,h}(s) \right) \leq 4\epsilon_k$$

Proof. Similarly to the proof of the previous lemma, we have

$$\begin{aligned}
& V_{\widehat{\mathcal{M}} \cup \widehat{r}_{k'}}^{\pi,h}(s) - V_{\widehat{\mathcal{M}} \cup r_k}^{\pi,h}(s) = V_{\widehat{\mathcal{M}} \cup \widehat{r}_{k'}}^{\pi,h}(s) - V_{\widehat{\mathcal{M}} \cup r}^{\pi,h}(s) + V_{\widehat{\mathcal{M}} \cup r}^{\pi,h}(s) - V_{\widehat{\mathcal{M}} \cup r_k}^{\pi,h}(s) \\
& \leq \sum_{h'=h}^H \sum_{s',a'} \mu_{\widehat{\mathcal{M}},\pi}^{h,h'}(s',a'|s) C_{k'}^{h'}(s',a') + \sum_{h'=h}^H \sum_{s',a'} \mu_{\widehat{\mathcal{M}},\pi}^{h,h'}(s',a'|s) C_k^{h'}(s',a') \leq 2\epsilon_k
\end{aligned}$$

where we use that the confidence intervals are shrinking with increasing episode number, i.e., $\epsilon_{k'} \leq \epsilon_k$.

By combining this with Lemma B.1.13, we get the result:

$$\begin{aligned} & \max_s \left(V_{\mathcal{M} \cup \hat{\mathcal{R}}_{k'}}^{\pi, h}(s) - V_{\mathcal{M} \cup \hat{\mathcal{R}}_k}^{\pi, h}(s) \right) \\ = & \max_s \left(\underbrace{V_{\mathcal{M} \cup \hat{\mathcal{R}}_{k'}}^{\pi, h}(s) - V_{\widehat{\mathcal{M}} \cup \hat{\mathcal{R}}_{k'}}^{\pi, h}(s)}_{\leq \epsilon_k} + \underbrace{V_{\widehat{\mathcal{M}} \cup \hat{\mathcal{R}}_{k'}}^{\pi, h}(s) - V_{\widehat{\mathcal{M}} \cup \hat{\mathcal{R}}_k}^{\pi, h}(s)}_{\leq 2\epsilon_k} + \underbrace{V_{\widehat{\mathcal{M}} \cup \hat{\mathcal{R}}_k}^{\pi, h}(s) - V_{\mathcal{M} \cup \hat{\mathcal{R}}_k}^{\pi, h}(s)}_{\leq \epsilon_k} \right) \leq 4\epsilon_k \end{aligned}$$

□

Lemma B.1.16. Under the good event \mathcal{E} , if $\hat{\pi}_k^*, \pi \in \widehat{\Pi}_{k-1}$ and $\pi \notin \widehat{\Pi}_k$, then the policy π is suboptimal for some reward $\hat{r}_{k'} \in \mathcal{R}_{\mathfrak{B}_{k'}}$ for all $k' \geq k$.

Proof. We can observe that

$$\begin{aligned} & V_{\mathcal{M} \cup \hat{\mathcal{R}}_{k'}}^{\pi, h}(s_0) - V_{\mathcal{M} \cup \hat{\mathcal{R}}_{k'}}^{*, h}(s_0) = V_{\mathcal{M} \cup \hat{\mathcal{R}}_{k'}}^{\pi, h}(s_0) - V_{\mathcal{M} \cup \hat{\mathcal{R}}_{k'}}^{\hat{\pi}_k^*, h}(s_0) \\ = & \underbrace{V_{\mathcal{M} \cup \hat{\mathcal{R}}_{k'}}^{\pi, h}(s_0) - V_{\mathcal{M} \cup \hat{\mathcal{R}}_k}^{\pi, h}(s_0)}_{\stackrel{(a)}{\leq} 4\epsilon_k} + \underbrace{V_{\mathcal{M} \cup \hat{\mathcal{R}}_k}^{\pi, h}(s_0) - V_{\widehat{\mathcal{M}} \cup \hat{\mathcal{R}}_k}^{\pi, h}(s_0)}_{\stackrel{(b)}{\leq} \epsilon_k} \\ & + \underbrace{V_{\widehat{\mathcal{M}} \cup \hat{\mathcal{R}}_k}^{\pi, h}(s_0) - V_{\widehat{\mathcal{M}} \cup \hat{\mathcal{R}}_k}^{\hat{\pi}_k^*, h}(s_0)}_{\stackrel{(c)}{>} 10\epsilon_k} + \underbrace{V_{\widehat{\mathcal{M}} \cup \hat{\mathcal{R}}_k}^{\hat{\pi}_k^*, h}(s_0) - V_{\mathcal{M} \cup \hat{\mathcal{R}}_k}^{\hat{\pi}_k^*, h}(s_0)}_{\stackrel{(b)}{\leq} \epsilon_k} \\ & + \underbrace{V_{\mathcal{M} \cup \hat{\mathcal{R}}_k}^{\hat{\pi}_k^*, h}(s_0) - V_{\mathcal{M} \cup \hat{\mathcal{R}}_{k'}}^{\hat{\pi}_k^*, h}(s_0)}_{\stackrel{(a)}{\leq} 4\epsilon_k} > 0 \end{aligned}$$

where we applied (a) Lemma B.1.13, (b) Lemma B.1.15, and (c) the definition of $\widehat{\Pi}_k$ and the fact that $\pi \notin \widehat{\Pi}_k$. Consequently, π is suboptimal for at least some reward function $\hat{r}_{k'} \in \mathcal{R}_{\mathfrak{B}_{k'}}$. □

Corollary B.1.4. For $\epsilon_0 = \frac{H}{10}$, for every $k \geq 0$ it holds that both $\pi^*, \hat{\pi}_{k+1}^* \in \widehat{\Pi}_k$.

Proof. We show the statement by induction over k . For $k = 0$, we have $10\epsilon_0 = H$ and therefore $\widehat{\Pi}_0$ contains all policies. Assume that

for $k - 1$ the statement holds, i.e., $\pi^*, \hat{\pi}_k^* \in \hat{\Pi}_{k-1}$, and consider k . By Lemma B.1.14, $\pi^* \in \hat{\Pi}_k$. Note, that $\hat{\pi}_{k+1}^* \in \hat{\Pi}_{k-1}$. Hence, by Lemma B.1.15, it follows that $\hat{\pi}_{k+1}^* \in \hat{\Pi}_k$ because it would be suboptimal otherwise which is a contradiction. \square

The last result we need, is quantifying the size of the policy confidence set.

Lemma B.1.17. Under the good event \mathcal{E} , let

$$\tilde{r} \in \operatorname{argmin}_{r \in \mathcal{R}_{\mathfrak{B}}} \max_{s,a} (r(s,a) - \hat{r}_k(s,a)),$$

where $\hat{r}_k = \mathcal{A}(\mathcal{R}_{\mathfrak{B}_k})$. If $\pi \in \hat{\Pi}_k$, then

$$\max_s (V_{\widehat{\mathcal{M}} \cup \tilde{r}}^{*,h}(s) - V_{\widehat{\mathcal{M}} \cup \tilde{r}}^{\pi,h}(s)) \leq 12\epsilon_k.$$

Proof.

$$\begin{aligned} & V_{\widehat{\mathcal{M}} \cup \tilde{r}}^{*,h}(s) - V_{\widehat{\mathcal{M}} \cup \tilde{r}}^{\pi,h}(s) \\ &= \underbrace{V_{\widehat{\mathcal{M}} \cup \tilde{r}}^{*,h}(s) - V_{\widehat{\mathcal{M}} \cup \hat{r}_k}^{*,h}(s)}_{\leq \epsilon_k} + \underbrace{V_{\widehat{\mathcal{M}} \cup \hat{r}_k}^{*,h}(s) - V_{\widehat{\mathcal{M}} \cup \hat{r}_k}^{\pi,h}(s)}_{\leq 10\epsilon_k} \\ &+ \underbrace{V_{\widehat{\mathcal{M}} \cup \hat{r}_k}^{\pi,h}(s) - V_{\widehat{\mathcal{M}} \cup \tilde{r}}^{\pi,h}(s)}_{\leq \epsilon_k} \epsilon_k \leq 14\epsilon_k \end{aligned}$$

\square

Next, we define the error upper bound based on the policy confidence set.

Definition B.1.6. Using $\hat{\Pi}_k$, we define recursively:

$$\begin{aligned} \hat{E}_k^H(s,a) &= 0 \\ \hat{E}_k^h(s,a) &= \min \left((H-h)r_{\max} C_k^h(s,a) + \sum_{s'} \hat{P}(s'|s,a) \max_{\pi \in \hat{\Pi}_{k-1}} \pi(a'|s') \hat{E}_k^{h+1}(s',a') \right) \end{aligned}$$

where \hat{P} is the estimated transition model of the environment. In contrast to Definition B.1.3, the maximization is over policies in $\hat{\Pi}_k$ rather than all actions.

This definition allows us to derive results that are analogous to the problem independent case.

Lemma B.1.18. Under the good event \mathcal{E} , for all policies $\pi \in \widehat{\Pi}_k$ and reward functions r and all $s, a \in \mathcal{S} \times \mathcal{A}$:

$$|Q_{\widehat{\mathcal{M}} \cup r}^{\pi, h}(s, a) - Q_{\mathcal{M} \cup r}^{\pi, h}(s, a)| \leq \widehat{E}_k^h(s, a)$$

Proof. The proof is the same as for Lemma B.1.7, restricting the set of policies to $\widehat{\Pi}_k$. \square

Lemma B.1.19. Under the good event \mathcal{E} , for all reward function r , all policies $\pi \in \widehat{\Pi}_k$, and all $s, a \in \mathcal{S} \times \mathcal{A}$:

$$|Q_{\widehat{\mathcal{M}} \cup \hat{r}}^{\pi, h}(s, a) - Q_{\mathcal{M} \cup r}^{\pi, h}(s, a)| \leq \widehat{E}_k^h(s, a)$$

Proof. The proof is the same as for Lemma B.1.8, restricting the set of policies to $\widehat{\Pi}_k$. \square

Lemma B.1.20. Under the good event \mathcal{E} , we have for any s, a, h :

$$\widehat{E}_k^h(s, a) \leq 2C_k^h(s, a) + \sum_{s'} P(s'|s, a) \max_{\pi \in \widehat{\Pi}_{k-1}} \pi(a'|s') \widehat{E}_k^{h+1}(s', a')$$

Proof. The proof is the same as for Lemma B.1.20. \square

Finally, we can combine these results to analyze the algorithm's sample complexity.

Theorem 4.4.1. [AceIRL Sample Complexity] AceIRL returns a (ϵ, δ, n) -correct solution with

$$n \leq \tilde{O} \left(\min \left[\frac{H^5 r_{\max}^2 |\mathcal{S}| |\mathcal{A}|}{\epsilon^2}, \frac{H^4 r_{\max}^2 |\mathcal{S}| |\mathcal{A}| \epsilon_{\tau-1}^2}{\min_{s, a, h} (A_{\mathcal{M} \cup r}^{*, h}(s, a))^2 \epsilon^2} \right] \right)$$

where $\epsilon_{\tau-1}$ depends on the choice of N_E , the number of episodes of exploration in each iteration. $A_{\mathcal{M} \cup r}^{*, h}(s, a)$ is the advantage function of $r \in \operatorname{argmin}_{r \in \mathcal{R}_{\mathcal{B}}} \max_{h, s, a} (r_h(s, a) - \hat{r}_{k, h}(s, a))$, the reward function from the feasible set $\mathcal{R}_{\mathcal{B}}$ closest to the estimated reward function \hat{r}_k .

Proof. First note that the analysis of Theorem B.1.2 still applies; so, in the worst case we get the same sample complexity. The key difference is that we no longer use the overall greedy policy w.r.t E_k^h , but restrict ourselves to policies in $\widehat{\Pi}_k$.

Again, we consider the error

$$e_k^{\pi,h}(s,a) := |Q_{\mathcal{M} \cup r}^{\pi^*,h}(s,a) - Q_{\mathcal{M} \cup r}^{\hat{\pi}^*,h}(s,a)|$$

where π^* is the true optimal policy in $\mathcal{M} \cup r$, and $\hat{\pi}^*$ is the optimal policy in $\widehat{\mathcal{M}} \cup \hat{r}$, i.e., in the estimated MDP using the inferred reward function.

Similar, to the proof of Lemma B.1.9, we can use Lemma B.1.18 and Lemma B.1.19 to show for all policies $\pi \in \widehat{\Pi}_k^h$, that:

$$e_k^{\pi,h}(s,a) \leq 4\hat{E}_k^h(s,a)$$

which implies the correctness of the algorithm according to Corollary B.1.2 when stopping at

$$\hat{E}_k^0(s_0, \pi_{k+1}(s_0)) \leq \frac{\epsilon}{4} \tag{B.1}$$

Now, consider the following condition for all s, a, h :

$$C_k^h(s,a) \leq -A_{\mathcal{M} \cup \tilde{r}}^{*,h}(s,a) \frac{\epsilon}{48\epsilon_{k-1}}, \tag{B.2}$$

where $\tilde{r} \in \operatorname{argmin}_{r \in \mathcal{R}_{\mathfrak{B}}} \max_{h,s,a} (r_h(s,a) - \hat{r}_{k,h}(s,a))$. We will (a) show that when this condition holds the previous stopping condition also holds, and (b) analyze after how many iterations this condition will certainly hold. Together this will yield the result.

To show that Equation (B.2) implies Equation (B.1), we assume that Equation (B.2) holds. Then, we get by applying Lemma B.1.20 recursively:

$$\begin{aligned}
& \hat{E}_k^0(s_0, \pi_{k+1}(s_0)) \\
& \leq 2 \max_{\pi \in \hat{\Pi}_{k-1}} \max_a \sum_{h=0}^H \sum_{s', a'} \mu_{\mathcal{M}, \pi}^{0, h}(s', a'; s_0, a) C_k^h(s', a') \\
& \leq 2 \max_{\pi \in \hat{\Pi}_{k-1}} \max_a \sum_{h=0}^H \sum_{s', a'} \mu_{\mathcal{M}, \pi}^{0, h}(s', a'; s_0, a) \left(-A_{\mathcal{M} \cup \bar{r}}^{*, h}(s', a') \frac{\epsilon}{48\epsilon_{k-1}} \right) \\
& \stackrel{(a)}{\leq} 2 \max_{\pi \in \hat{\Pi}_{k-1}} (V_{\mathcal{M} \cup r}^{*, 0}(s_0) - V_{\mathcal{M} \cup r}^{\pi, 0}(s_0)) \frac{\epsilon}{48\epsilon_{k-1}} \stackrel{(b)}{\leq} \frac{\epsilon}{4}
\end{aligned}$$

where (a) uses Lemma B.1.12 and (b) uses Lemma B.1.17.

Next, we analyze after how many iterations Equation (B.2) holds, which will give a lower bound on the sample complexity result. The argument proceeds similar to the proof of Theorem B.1.2.

Before the algorithm terminates at τ , we have for all $k < \tau$:

$$\min_{s, a, h} (-A_{\mathcal{M} \cup \bar{r}}^{*, h}(s, a)) \frac{\epsilon}{48\epsilon_{k-1}} < \max_{s, a, h} C_k^h(s, a) \leq Hr_{\max} \sqrt{\frac{2\ell_k^h(s, a)}{\max(N_k^h(s, a),)}}$$

Using similar argument to the proof of Theorem B.1.2, using the same pseudo-counts, we arrive at:

$$\min_{s, a, h} (-A_{\mathcal{M} \cup \bar{r}}^{*, h}(s, a)) \frac{\epsilon}{48\epsilon_{\tau-1}} \sqrt{\tau+1} \leq Hr_{\max} \sqrt{8|\mathcal{S}||\mathcal{A}| \log(12|\mathcal{S}||\mathcal{A}|H\tau^2/\delta)}$$

Again, we can use Lemma 15 by [85] to find that

$$\tau \leq \tilde{\mathcal{O}} \left(\frac{H^3 r_{\max}^2 |\mathcal{S}| |\mathcal{A}| \epsilon_{\tau-1}^2}{\min_{s, a, h} (A_{\mathcal{M} \cup \bar{r}}^{*, h}(s, a))^2 \epsilon^2} \right)$$

□

B.1.6 Computing the Exploration Policy

To run AceIRL, we need to solve the optimization problem:

$$\pi_k^h = \min_{\pi} \max_{\hat{\pi} \in \hat{\Pi}_{k-1}} \sum_{h=0}^H \sum_{s', a'} \mu_{\mathcal{M}, \hat{\pi}}^{0, h}(s', a'; s_0) \hat{C}_k^h(s', a' | \pi)$$

For simplicity let us denote the state visitation frequencies by

$$\mu_h(s, a) := \mu_{\widehat{\mathcal{M}}, \pi}^{0, h}(s, a; s_0)$$

$$\hat{\mu}_h(s, a) := \mu_{\widehat{\mathcal{M}}, \hat{\pi}}^{0, h}(s, a; s_0)$$

Let us introduce the following matrix notation

$$\tilde{A} = \begin{bmatrix} I & 0 & 0 & 0 & \dots & 0 \\ \widehat{P} & -I & 0 & 0 & \dots & 0 \\ 0 & \widehat{P} & -I & 0 & \dots & 0 \\ & & \dots & & & \\ 0 & 0 & \dots & 0 & \widehat{P} & -I \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ & & \dots & & & \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}, \quad a = \begin{bmatrix} \hat{r}_{k-1}^0 \\ \hat{r}_{k-1}^1 \\ \dots \\ \hat{r}_{k-1}^H \end{bmatrix}, \quad A = \begin{bmatrix} A & 0 \\ a^T & -1 \end{bmatrix},$$

$$x = \begin{bmatrix} \mu_0 \\ \mu_1 \\ \dots \\ \mu_H \\ t \end{bmatrix}, \quad \hat{x} = \begin{bmatrix} \hat{\mu}_0 \\ \hat{\mu}_1 \\ \dots \\ \hat{\mu}_H \end{bmatrix}, \quad b = \begin{bmatrix} \bar{\mu}_0 \\ 0 \\ \dots \\ 0 \\ 1 \\ \dots \\ 1 \\ -10\epsilon_{k-1} \end{bmatrix}, \quad c = \begin{bmatrix} C_0 \\ C_1 \\ \dots \\ C_H \\ 1 \end{bmatrix},$$

where $\bar{\mu}_0$ is the actual initial state distribution of the environment (which we assume to know). We can now write the inner maximization problem above as a linear program:

$$\max_x c^T x \quad \text{s.t.} \quad Ax = b, \quad x \geq 0$$

The corresponding dual problem is:

$$\min_y b^T y \quad \text{s.t.} \quad A^T y \geq c$$

Using this we can write the full min-max problem as:

$$\min_{\hat{x}, y} b^T y \quad \text{s.t.} \quad A^T y \geq c(x), \quad \tilde{A}x = b, \quad x \geq 0$$

which is a convex optimization problem, if we use:

$$C_h(s, a) = 2(H - h)r_{\max} \sqrt{\frac{2 \log(24|\mathcal{S}||\mathcal{A}|H(\max(1, n_k^h(s, a)))^2 / \delta)}{\max(1, \hat{n}_{k+1}^h(s, a))}}$$

where $\hat{n}_{k+1}^h(s, a) = n_k^h(s, a) + \mu_h(s, a) * N_E$ is the number of times we expect h, s, a to be visited at the next iteration.

Solving this optimization problem yields the state-visitation frequencies $\hat{\mu}_k(s, a)$. We can then find the exploration policy that induces these state-visitations simply as:

$$\pi_{k,h}(a|s) := \frac{\hat{\mu}_k^h(s, a)}{\sum_{a'} \hat{\mu}_k^h(s, a')}.$$

B.2 EXPERIMENT DETAILS

In this section, we provide more details on our experiments. In particular, we discuss the environments in detail (Appendix B.2.1), and we provide additional plots of all experiments we discussed in the main chapter (Appendix B.2.2).

B.2.1 *Environments*

FOUR PATHS. The four paths environment has 41 states and 4 actions:

$$\mathcal{S} = \{c, l_1, \dots, l_{10}, u_1, \dots, u_{10}, r_1, \dots, r_{10}, d_1, \dots, d_{10}\},$$

$$\mathcal{A} = \{a_1, a_2, a_3, a_4\},$$

and a time horizon of $H = 20$. The agent starts in the center state c , from which can move in four directions: left (a_1), up (a_2), right (a_3), or down (a_4). Each action a_i has a probability p_i of failing. If an action fails it moves in the opposite direction. p_1, \dots, p_4 are sampled uniformly from $(0, 0.3)$. One of the states $(l_{10}, u_{10}, r_{10}, d_{10})$ is chosen as the goal state at random. The reward in the goal state is 1, all other rewards are 0.

DOUBLE CHAIN. The *Double Chain* MDP, proposed by Kaufmann *et al.* [85], consists of L states $\mathcal{S} = \{s_0, \dots, s_{L-1}\}$, and two actions $\mathcal{A} = \{\text{left}, \text{right}\}$, which correspond to a transition to the left or to the right. When the agent takes an action, there is a 0.1 probability of moving to the other direction. The state s_{L-1} has reward 1, all other states have reward 0, and the agent starts in the center of the chain at $s_{(L-1)/2}$. We choose $L = 31$, similar to Kaufmann *et al.* [85]. The environment has horizon $H = 20$.

CHAIN. The *Chain* MDP, proposed by Metelli *et al.* [72] has 6 states $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5, s_u\}$ and 10 actions $\mathcal{A} = \{a_1, \dots, a_{10}\}$. The agent starts in a random initial state. Taking action a_{10} moves it right along the chain with probability 0.7 and to state s_u with probability 0.3. Any other action moves the agent right with probability 0.3 and to state s_u with probability 0.7. If the agent is in state s_u , action a_{10} moves it back to state s_1 with probability 0.05. Any other action moves it to s_1 with probability 0.01. The reward is 1 in all states except s_u where the reward is 0. Metelli *et al.* [72] provide an illustration of the environment in Figure 3. We choose $H = 10$ for the chain.

GRIDWORLD. The *Gridworld*, proposed by Metelli *et al.* [72], is a 3×3 Gridworld with an obstacle in the center cell $(2, 2)$ and a goal

cell at the right center cell $(2, 1)$. The agent starts in a random non-goal cell, and it has 4 action one to move in each direction. If the agent takes an action with probability 0.3 the action fails and the agent moves in a random direction instead. If the agent is in the center cell $(2, 2)$ which has the obstacle, if the agent would move right it instead stays in the center cell with probability 0.8. The reward in the goal cell is 1, all other rewards are 0. Metelli *et al.* [72] provide an illustration of the Gridworld in Figure 6. We choose $H = 10$ for the Gridworld.

RANDOM MDPs. We generate random MDPs by uniformly sampling an initial state distribution and transition matrix and normalizing them. The rewards are sampled uniformly between 0 and 1. Our random MDPs have 9 states, 4 actions and horizon 10.

B.2.2 *Additional Results*

We provide full learning curves for all experiments discussed in the main chapter in Figure B.1.

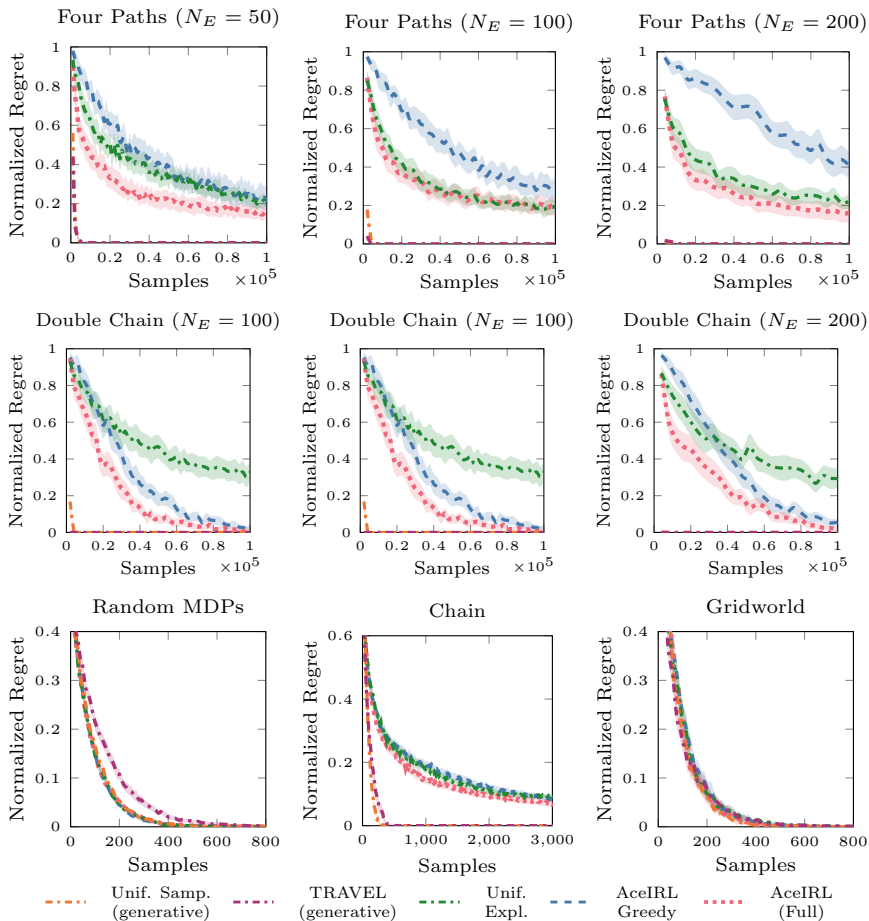


FIGURE B.1: Full learning curves for all experiments shown in Table 4.1. Similar to Figure 4.2, we show the mean and 95% confidence intervals computed over 50 random seeds. In addition to the exploration algorithms, we also show uniform sampling and TRAVEL which are much faster in most cases because they have access to a generative model.

CONVEX CONSTRAINT LEARNING FOR RL

This appendix provides additional information about Chapter 5. We provide proofs of theoretical results (Appendix C.1), more details about our implementation of CoCoRL (Appendix C.2), and more details about the experimental setup (Appendix C.3).

C.1 PROOFS

This section provides all proofs omitted in the main chapter, as well as a few additional results related to estimating feature expectations (Appendix C.1.4).

C.1.1 *Limitations of IRL in CMDPs*

Proposition 5.2.1 (IRL can be unsafe). There are CMDPs $\mathcal{C} = (\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r, \{c_j\}_{j=1}^n, \{\xi_j\}_{j=1}^n)$ such that for any optimal policy π^* in \mathcal{C} and any reward function r_{IRL} that could be returned by an IRL algorithm, the resulting MDP $(\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r_{\text{IRL}})$ has optimal policies that are unsafe in \mathcal{C} .

Proof. This follows from the fact that there are CMDPs for which all feasible policies are stochastic. Every MDP, on the other hand, has a deterministic policy that is optimal [33].

For example, consider a CMDP with a single state $\mathcal{S} = \{s_1\}$ and two actions $\mathcal{A} = \{a_1, a_2\}$ with $P(s_1|s_1, a_1) = P(s_1, |s_1, a_2) = 1$. We have two cost functions $c_1(s_1, a_1) = 1, c_1(s_1, a_2) = 0$ and $c_2(s_1, a_1) = 0, c_2(s_1, a_2) = 1$, with thresholds $\xi_1 = \xi_2 = \frac{1}{2}$, and discount factor $\gamma = 0$. To be feasible, a policy needs to have an occupancy measure μ_π such that $\sum_{s,a} \mu(s, a)c_1(s, a) \leq \xi_1$ and $\sum_{s,a} \mu_\pi(s, a)c_2(s, a) \leq \xi_2$.

In our case with a single state and $\gamma = 0$, this simply means that a feasible policy π needs to satisfy $\pi(a_1|s_1) \leq \frac{1}{2}$ and $\pi(a_2|s_1) \leq \frac{1}{2}$. But this implies that only a uniformly random policy is feasible.

Any IRL algorithm infers a reward function that matches the occupancy measure of the expert policy [45]. Consequently, the inferred reward r_{IRL} needs to give both actions the same reward, because otherwise π^* would not be optimal for the MDP with r_{IRL} . However, that MDP also has a deterministic optimal policy (like any MDP), which is not feasible in the original CMDP. \square

Proposition 5.2.2. Let $\mathcal{C} = (\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, \{c_j\}_{j=1}^n, \{\xi_j\}_{j=1}^n)$ be a CMDP without reward. Let r_1, r_2 be two reward functions and π_1^* and π_2^* corresponding optimal policies in $\mathcal{C} \cup \{r_1\}$ and $\mathcal{C} \cup \{r_2\}$. Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma)$ be the corresponding MDP without reward. Without additional assumptions, we cannot guarantee the existence of a function $\hat{c} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ such that π_1^* is optimal in the MDP $\mathcal{M} \cup \{r_1 + \hat{c}\}$ and π_2^* is optimal in the MDP $\mathcal{M} \cup \{r_2 + \hat{c}\}$.

Proof. As in the proof of Proposition 5.2.1, we consider a single-state CMDP with $\mathcal{S} = \{s_1\}$ and $\mathcal{A} = \{a_1, a_2\}$, where a_1 . We have $c_1(s_1, a_1) = 1, c_1(s_1, a_2) = 0$ and $c_2(s_1, a_1) = 0, c_2(s_1, a_2) = 1, \xi_1 = \xi_2 = \frac{1}{2}$, and $\gamma = 0$. Again, the only feasible policy uniformly randomizes between a_1 and a_2 .

Let $r_1(a_1) = 1, r_1(a_2) = 1, r_2(a_1) = 0, r_2(a_2) = 1$, i.e., the first reward function gives equal reward to both actions and the second reward function gives unequal rewards (for simplicity we write $r(a) = r(s, a)$). To make the uniformly random policy optimal in $\mathcal{M} \cup \{r_1 + \hat{c}\}$ and $\mathcal{M} \cup \{r_2 + \hat{c}\}$, both inferred reward functions $r_1 + \hat{c}$ and $r_2 + \hat{c}$ need to give both actions equal rewards. However this is clearly impossible. \square

C.1.2 Safety Guarantees

Theorem 5.3.2 (Inferred CMDP). We can find cost functions and thresholds such that for any reward function r_{eval} , solving the inferred

CMDP $(\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r_{\text{eval}}, \{\hat{c}_j\}_{j=1}^m, \{\hat{\xi}_j\}_{j=1}^m)$ is equivalent to solving $\pi^* \in \operatorname{argmax}_{\pi \in \mathcal{S}} G_{r_{\text{eval}}}(\pi)$. Consequently, if an optimal policy for the true CMDP is in S , solving the inferred CMDP will find an optimal policy for the true CMDP.

Proof. The convex hull of a set of points is a convex polyhedron, i.e., it is the solution of a set of linear equations (see, e.g., Theorem 2.9 in [112]).

Hence, S is a convex polyhedron in the feature space defined by \mathbf{f} , i.e., we can find $A \in \mathbb{R}^{p \times d}$, $\mathbf{b} \in \mathbb{R}^p$ such that

$$S = \{x \in \mathbb{R}^d \mid Ax \leq \mathbf{b}\}.$$

To construct A and \mathbf{b} , we need to find the facets of S , the convex hull of a set of points \mathcal{D} . This is a standard problem in polyhedral combinatorics (e.g., see [112]).

We can now define p linear constraint functions $\hat{c}_j(s, a) = A_j \mathbf{f}(s, a)$ and thresholds $\hat{\xi}_j = b_j$, where A_j is the j -th row of A and b_j is the j -th component of \mathbf{b} (for $j = 1, \dots, p$). Then, solving the CMDP $(\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r_{\text{eval}}, \{\hat{c}_j\}_{j=1}^p, \{\hat{\xi}_j\}_{j=1}^p)$ corresponds to solving

$$\pi^* \in \operatorname{argmax}_{\hat{J}_1(\pi) \leq \hat{\xi}_1, \dots, \hat{J}_p(\pi) \leq \hat{\xi}_p} G_{r_{\text{eval}}}(\pi).$$

For these linear cost functions, we can rewrite the constraints using the discounted feature expectations as $J_i(\pi) = A_i \mathbf{f}(\pi)$. Hence, π satisfying the constraints $\hat{J}_1(\pi) \leq \hat{\xi}_1, \dots, \hat{J}_p(\pi) \leq \hat{\xi}_p$ is equivalent to $\pi \in S$. \square

Corollary 5.3.1 (S is maximal). If $\pi \notin S$, there exist r_1, \dots, r_k and c_1, \dots, c_n , such that the expert policies π_1^*, \dots, π_k^* are optimal in the CMDPs $(\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r_i, \{c_j\}_{j=1}^n, \{\xi_j\}_{j=1}^n)$, but $\pi \notin \mathcal{F}$.

Proof. Using Theorem 5.3.2, we can construct a set of cost functions $\{c_j\}_{j=1}^n$ and thresholds $\{\xi_j\}_{j=1}^n$ for which $\pi \in S$ is equivalent to $J_1(\pi) \leq \xi_1, \dots, J_n(\pi) \leq \xi_n$. Hence, in a CMDP with these cost functions and thresholds, any policy $\pi \notin S$ is not feasible. Because $\pi_1^*, \dots, \pi_k^* \in S$ by construction, each π_i^* is optimal in the CMDP $(\mathcal{S}, \mathcal{A}, P, \mu_0, \gamma, r_i, \{c_j\}_{j=1}^n, \{\xi_j\}_{j=1}^n)$. \square

C.1.3 Optimality Guarantees

Lemma C.1.1. For any policy π , if $r : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ and $\mathbf{f} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]^d$, we can bound

$$\begin{aligned} \forall i : \mathbf{f}(\pi)_i &\leq \frac{1}{1-\gamma} & \|\mathbf{f}(\pi)\|_2 &\leq \frac{\sqrt{d}}{1-\gamma} \\ G_r(\pi) &\leq \frac{d}{1-\gamma} & \mathcal{R}(\pi, S) &\leq \frac{2d}{1-\gamma} \end{aligned}$$

Proof. First, we have for any component of the feature expectations

$$\mathbf{f}_i(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{f}(s_t, a_t)\right] \leq \sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma},$$

using the limit of the *geometric series*. This immediately gives

$$\|\mathbf{f}(\pi)\|_2 = \sqrt{\sum_{i=1}^d \mathbf{f}(\pi)_i^2} \leq \frac{\sqrt{d}}{1-\gamma}.$$

Similarly bounded rewards $r(s, a) \leq 1$ imply $\|\theta\|_2 \leq \sqrt{d}$. Together, we can bound the returns using the Cauchy-Schwartz inequality

$$G_r(\pi) = \|\theta^T \mathbf{f}(\pi)\|_2 \leq \|\theta\|_2 \cdot \|\mathbf{f}(\pi)\|_2 \leq \frac{d}{1-\gamma}.$$

Further, for any two policies π_1, π_2 , we have $G_r(\pi_1) - G_r(\pi_2) \leq \frac{2d}{1-\gamma}$, which implies the same for the regret $\mathcal{R}(r, S) \leq \frac{2d}{1-\gamma}$. \square

Theorem 5.3.3 (Convergence, exact optimality). Under Assumption 5.3.1, for any $\delta > 0$, after $k \geq \log(\delta / f_v(d, n)) / \log(1 - \delta / f_v(d, n))$, we have $P(\mathcal{R}(r, S_k) > 0) \leq \delta$, where $f_v(d, n)$ is an upper bound on the number of vertices of the true safe set. In particular,

$$\lim_{k \rightarrow \infty} \mathbb{E}_r [\mathcal{R}(r, S_k)] = 0.$$

Proof. Consider the distribution over optimal policies $P(\mathbf{f}(\pi^*))$ induced by $P(r)$. For each reward r , there is a vertex of the true safe set that is optimal and thus is in the support of $P(\mathbf{f}(\pi^*))$.¹

We can distinguish two cases.

- **Case 1:** we have seen a vertex corresponding to an optimal policy for r in the first k demonstrations.
- **Case 2:** we have not.

In **Case 1**, we incur 0 regret, and only in **Case 2** we can incur regret greater than 0.

So, we can bound the probability of incurring regret by the probability of **Case 2**:

$$P(\mathcal{R}(r, S_k) > 0) \leq \sum_{\text{vertex } \mathbf{v}} (1 - P(\mathbf{v}))^k P(\mathbf{v}).$$

To ensure $P(\mathcal{R}(r, S_k) > 0) \leq \delta$, it is sufficient to ensure that each term of the sum satisfies $(1 - P(\mathbf{v}))^k P(\mathbf{v}) \leq \delta/N_v$ where N_v is the number of vertices. For terms with $P(\mathbf{v}) \leq \delta/N_v$ this is true for all k . For the remaining terms with $P(\mathbf{v}) > \delta/N_v$, we can write

$$(1 - P(\mathbf{v}))^k P(\mathbf{v}) \leq (1 - \delta/N_v)^k.$$

So, it is sufficient to ensure $(1 - \delta/N_v)^k \leq \delta/N_v$, which is satisfied once

$$k > \frac{\log(\delta/N_v)}{\log(1 - \delta/N_v)}.$$

By replacing N_v with a suitable upper bound on the number of vertices, we arrive at the first result.

¹ Technically, there are degenerate cases where $P(r)$ is only supported on reward functions that are orthogonal to the constraint boundaries and we never see demonstrations at the vertices of the true safe set. This is an artifact of the relatively unnatural assumption of noise-free demonstrations. We can avoid such degenerate cases by mild assumptions: either assuming some minimal noise in $P(r)$ or assuming the algorithm generating the demonstrations has non-zero probability for all policies optimal for a given reward.

By Lemma C.1.1, the maximum regret is upper-bounded by $\frac{2d}{1-\gamma}$, and, we can decompose the regret as

$$\mathbb{E}_r[\mathcal{R}(r, S_k)] \leq \frac{2d}{1-\gamma} \sum_{\text{vertex } \mathbf{v}} (1 - P(\mathbf{v}))^k P(\mathbf{v}) \leq \frac{2d}{1-\gamma} \sum_{\mathbf{v}: P(\mathbf{v}) > 0} (1 - P(\mathbf{v}))^k.$$

Because $P(\mathbf{v})$ is a fixed distribution induced by $P(r)$, the r.h.s. converges to 0 as $k \rightarrow \infty$. \square

Lemma C.1.2. Under Assumption 5.3.2, we have for any reward function r and any feasible policy $\pi \in \mathcal{F}$

$$P(\pi) \geq \exp(-\beta d / (1 - \gamma)).$$

Proof. We have $\beta > 0$ and $G_r(\pi) \geq 0$, which implies $\exp(\beta G_r(\pi)) \geq 1$, and

$$P(\pi|r) = \frac{\exp(\beta G_r(\pi))}{Z(\theta)} \geq \frac{1}{Z(\theta)}.$$

By Lemma C.1.1, we have $G_r(\pi) \leq d / (1 - \gamma)$. Therefore,

$$Z(\theta) = \int_{\pi \in \mathcal{F}} \exp(\beta \theta^T \mathbf{f}(\pi)) \, d\pi \leq \exp(\beta d / (1 - \gamma)),$$

and

$$P(\pi|r) \geq \exp(-\beta d^2 / (1 - \gamma)).$$

\square

Theorem 5.3.4 (Convergence, Boltzmann-rationality). Under Assumption 5.3.2, for any $\delta > 0$, after $k \geq \log(\delta / f_v(d, n)) / \log(1 - \exp(-\beta d / (1 - \gamma)))$, we have $P(\mathcal{R}(r, S_k) > 0) \leq \delta$, where $f_v(d, n)$ is an upper bound on the number of vertices of the true safe set. In particular,

$$\lim_{k \rightarrow \infty} \mathbb{E}_r[\mathcal{R}(r, S_k)] = 0.$$

Proof. We can upper-bound the probability of having non-zero regret similar to the noise free case by

$$P(\mathcal{R}(r, S_k) > 0) \leq \sum_{\text{vertex } \mathbf{v}} (1 - P(\mathbf{v}))^k P(\mathbf{v}).$$

Under Assumption 5.3.2, we can use Lemma C.1.2, to obtain

$$P(\mathbf{v}) \geq \exp(-\beta d/(1-\gamma)) := \Delta.$$

Importantly, $0 < \Delta < 1$ is a constant, and we have

$$P(\mathcal{R}(r, S_k) > 0) \leq \sum_{\text{vertex } \mathbf{v}} (1-\Delta)^k.$$

To ensure $P(\mathcal{R}(r, S_k) > 0) \leq \delta$, it is sufficient to ensure $(1-\Delta)^k \leq \delta/N_v$, where N_v is the number of vertices of the true safe set. This is true once

$$k \geq \frac{\log(\delta/N_v)}{\log(1-\exp(-\beta d/(1-\gamma)))},$$

where we can again replace N_v by a suitable upper bound.

Bounding the maximum regret via Lemma C.1.1, we get

$$\mathbb{E}_r[\mathcal{R}(r, S_k)] \leq \frac{2d}{1-\gamma} \sum_{\text{vertex } \mathbf{v}} (1-\Delta)^k,$$

which converges to 0 as $k \rightarrow \infty$. □

c.1.4 Estimating Feature Expectations

Lemma C.1.3. Let π be a policy with true feature expectation $\mathbf{f}(\pi)$. We estimate the feature expectation using n_{traj} trajectories τ_i collected by rolling out π in the environment: $\hat{\mathbf{f}}(\pi) = \frac{1}{n_{\text{traj}}} \sum_{i=1}^{n_{\text{traj}}} \mathbf{f}(\tau_i)$. This estimate is unbiased, i.e., $\mathbb{E}[\hat{\mathbf{f}}(\pi)] = \mathbf{f}(\pi)$. Further let $\phi \in \mathbb{R}^d$ be a vector, such that, $0 \leq \phi^T \mathbf{f}(s, a) \leq 1$ for any state s and action a . Then, we have for any $\epsilon > 0$

$$\begin{aligned} P(\phi^T \hat{\mathbf{f}}(\pi) - \phi^T \mathbf{f}(\pi) \geq \epsilon) &\leq \exp(-2\epsilon^2 n_{\text{traj}}(1-\gamma)/d), \\ P(\phi^T \mathbf{f}(\pi) - \phi^T \hat{\mathbf{f}}(\pi) \geq \epsilon) &\leq \exp(-2\epsilon^2 n_{\text{traj}}(1-\gamma)/d). \end{aligned}$$

Proof. Because the expectation is linear, the estimate is unbiased, and, $\mathbb{E}[\phi^T \hat{\mathbf{f}}(\pi)] = \phi^T \mathbf{f}(\pi)$.

For any trajectory τ and any $1 \leq i \leq d$, we have $0 \leq \mathbf{f}_i(\tau) \leq 1/(1 - \gamma)$, and, consequently, $0 \leq \phi^T \mathbf{f}(\tau) \leq d/(1 - \gamma)$, analogously to Lemma C.1.1). Thus, $\phi^T \hat{\mathbf{f}}(\pi)$ satisfies the bounded difference property. In particular, by changing one of the observed trajectories, the value of $\phi^T \hat{\mathbf{f}}(\pi)$ can change at most by $d/(n_{\text{traj}}(1 - \gamma))$. Hence, by McDiarmid's inequality, we have for any $\epsilon > 0$

$$P(\theta^T \hat{\mathbf{f}}(\pi) - \mathbb{E}[\theta^T \hat{\mathbf{f}}(\pi)] \geq \epsilon) \leq \exp\left(-\frac{2\epsilon^2}{d/(n_{\text{traj}}(1 - \gamma))}\right),$$

and

$$P(\theta^T \hat{\mathbf{f}}(\pi) - \theta^T \mathbf{f}(\pi) \geq \epsilon) \leq \exp\left(-\frac{2\epsilon^2 n_{\text{traj}}(1 - \gamma)}{d}\right).$$

We can bound $P(\phi^T \mathbf{f}(\pi) - \phi^T \hat{\mathbf{f}}(\pi) \geq \epsilon)$ analogously. \square

Theorem 5.3.5 (ϵ -safety with estimated feature expectations). Suppose we estimate the feature expectations of each policy π_i^* using at least $n_{\text{traj}} > d \log(nk/\delta)/(2\epsilon^2(1 - \gamma))$ samples, and construct the estimated safe set $\hat{S} = \text{conv}(\hat{\mathbf{f}}(\pi_1^*), \dots, \hat{\mathbf{f}}(\pi_k^*))$. Then, we have $P(\max_j (J_j(\pi) - \zeta_j) > \epsilon | \pi \in \hat{S}) < \delta$.

Proof. Let us define the “good” event that for policy π_i^* we accurately estimate the value of the cost function c_j and denote it by $\mathcal{E}_{ij} = \{\phi_j^T \hat{\mathbf{f}}(\pi_i^*) - \phi_j^T \mathbf{f}(\pi_i^*) \leq \epsilon\}$, where ϕ_j parameterizes c_j . Conditioned on \mathcal{E}_{ij} for all i and j , we have

$$\begin{aligned} & P(\max_j (J_j(\pi) - \zeta_j) > \epsilon | \pi \in \hat{S}, \{\mathcal{E}_{ij}\}) \\ & \leq \sum_j P((J_j(\pi) - \zeta_j) > \epsilon | \pi \in \hat{S}, \{\mathcal{E}_{ij}\}) \\ & = \sum_j P(\phi_j^T \sum_l \lambda_l \hat{\mathbf{f}}(\pi_l^*) > \zeta_j + \epsilon | \{\mathcal{E}_{ij}\}) = 0. \end{aligned}$$

Hence, we can bound the probability of having an unsafe policy by the probability of the “bad” event, which we can bound by Lemma C.1.3 to obtain

$$\begin{aligned} & P(\max_j (J_j(\pi) - \xi_j) > \epsilon | \pi \in \hat{S}) \\ & \leq \sum_{ij} P(\text{not } \mathcal{E}_{ij}) \leq n \cdot k \cdot \exp(-2\epsilon^2 n_{\text{traj}}(1 - \gamma)/d). \end{aligned}$$

So, to ensure $P(\max_j (J_j(\pi) - \xi_j) > \epsilon | \pi \in \hat{S}) \leq \delta$, it is sufficient to ensure

$$\exp(-2\epsilon^2 n_{\text{traj}}(1 - \gamma)/d) \leq \frac{\delta}{nk},$$

which is true if we collect at least

$$n_{\text{traj}} > \frac{d \log(nk/\delta)}{2\epsilon^2(1 - \gamma)}$$

trajectories for each policy. \square

Theorem C.1.1 (Convergence, noise-free, estimated feature expectations). Under Assumption 5.3.1, if we estimate the feature expectations of at least $k > \log(\delta/(2f_v(d, n)))/\log(1 - \delta/(2f_v(d, n)))$ expert policies using at least $n_{\text{traj}} > d \log(2f_v(d, n)/\delta)/(2\epsilon^2(1 - \gamma))$ samples each, we have $P(\mathcal{R}(r, S_k) > \epsilon) \leq \delta$, where $f_v(d, n)$ is an upper bound on the number of vertices of the true safe set. In particular, we have $\lim_{\min(k, n_{\text{traj}}) \rightarrow \infty} \mathbb{E}_r [\mathcal{R}(r, S_k)] = 0$.

Proof. To reason about the possibility of a large estimation error for the feature expectations, we use Lemma C.1.3 to lower bound the probability of the good event $\mathcal{E}_{\mathbf{v}} = \{\theta^T \mathbf{v} - \theta^T \hat{\mathbf{f}}(\pi_i^*) \leq \epsilon\}$, for each vertex \mathbf{v} of the true safe set.

We can then extend the argument we used to prove Theorem 5.3.3 to incorporate possible estimation error. In particular, for an evaluation reward r_{eval} , we can incur regret in one of three cases:

- **Case 1:** we have *not* seen a vertex corresponding to an optimal policy for r_{eval} .

- **Case 2:** we have seen an optimal vertex but our estimation of that vertex is *bad*.
- **Case 3:** we have seen an optimal vertex and our estimation of that vertex is *good*.

We show that the probability of **Case 1** shrinks as we increase k , the probability of **Case 2** shrinks as we increase n_{traj} , and the regret that we can incur in **Case 3** is small.

Case 1: This case is independent of the estimated feature expectations. Similar to Theorem 5.3.3, we can bound its probability for each vertex \mathbf{v} by $(1 - P(\mathbf{v}))^k P(\mathbf{v})$.

Case 2: In case of the bad event, i.e., the complement of $\mathcal{E}_{\mathbf{v}}$, we can upper-bound the probability of incurring high regret using Lemma C.1.3, to obtain

$$\begin{aligned} P(\theta^T \mathbf{v} - \theta^T \hat{\mathbf{f}}(\pi_i^*) > \epsilon | \text{not } \mathcal{E}_{\mathbf{v}}) & P(\text{not } \mathcal{E}_{\mathbf{v}}) \\ & \leq P(\text{not } \mathcal{E}_{\mathbf{v}}) \\ & \leq \exp(-2\epsilon^2 n_{\text{traj}}(1 - \gamma)/d). \end{aligned}$$

Case 3: Under the good event $\mathcal{E}_{\mathbf{v}}$, we automatically have regret less than ϵ , i.e.,

$$P(\theta^T \mathbf{v} - \theta^T \hat{\mathbf{f}}(\pi_i^*) > \epsilon | \mathcal{E}_{\mathbf{v}}) P(\mathcal{E}_{\mathbf{v}}) = 0.$$

Using these three cases, we can now bound the probability of having regret greater than ϵ as

$$\begin{aligned} & P(\mathcal{R}(r, S_k) > \epsilon) \\ & \leq \sum_{\text{vertex } \mathbf{v}} \left(\underbrace{(1 - P(\mathbf{v}))^k P(\mathbf{v})}_{\text{Case 1}} + \underbrace{\exp(-2\epsilon^2 n_{\text{traj}}(1 - \gamma)/d)}_{\text{Case 2}} + \underbrace{0}_{\text{Case 3}} \right). \end{aligned}$$

To ensure that $P(\mathcal{R}(r, S_k) > \epsilon) \leq \delta$, it is sufficient to ensure each term of the sum is less than δ/N_v where N_v is the number of vertices. We have two terms inside the sum, so it is sufficient to ensure either term is less than $\delta/(2N_v)$.

Case 1: We argue analogously to Theorem 5.3.3. For terms with $P(\mathbf{v}) \leq \delta/(2N_v)$ it is true for all k . For the remaining terms with $P(\mathbf{v}) > \delta/(2N_v)$, we can use

$$(1 - P(\mathbf{v}))^k P(\mathbf{v}) \leq (1 - \delta/(2N_v))^k$$

so it is sufficient to ensure

$$(1 - \delta/(2N_v))^k \leq \delta/(2N_v)$$

which is satisfied once

$$k > \frac{\log(\delta/(2N_v))}{\log(1 - \delta/(2N_v))}.$$

This is our first condition.

Case 2: To ensure $\exp(-2\epsilon^2 n_{\text{traj}}(1 - \gamma)/d) \leq \delta/(2N_v)$, we need

$$n_{\text{traj}} > \frac{d \log(2N_v/\delta)}{2\epsilon^2(1 - \gamma)}.$$

This is our second condition.

If both conditions hold simultaneously, we have $P(\mathcal{R}(r, S_k) > \epsilon) \leq \delta$.

To analyse the regret, let us consider how ϵ shrinks as a function of n_{traj} . The analysis above holds for any ϵ . So, for a given k and n_{traj} , we need to choose

$$\epsilon^2 \geq \frac{d \log(2nkN_v/\delta)}{2n_{\text{traj}}(1 - \gamma)}$$

to guarantee $P(\mathcal{R}(r, S_k) > \epsilon) \leq \delta$. Hence, the smallest ϵ we can choose is

$$\epsilon_{\min} = \sqrt{\frac{d \log(2nkN_v/\delta)}{2n_{\text{traj}}(1 - \gamma)}}.$$

Using Lemma C.1.1 to upper-bound the maximum regret by $\frac{2d^2}{1-\gamma}$, we can upper-bound the expected regret by

$$\begin{aligned}
\mathbb{E}_r[\mathcal{R}(r, S_k)] &\leq \frac{2d^2}{1-\gamma} P(\mathcal{R}(r, S_k) > \epsilon_{\min}) + \epsilon_{\min} P(\mathcal{R}(r, S_k) \leq \epsilon_{\min}) \\
&\leq \frac{2d}{1-\gamma} \sum_{\mathbf{v}: P(\mathbf{v}) > 0} (1 - P(\mathbf{v}))^k + \frac{2dN_v}{1-\gamma} \exp(-2\epsilon_{\min}^2(1-\gamma)/d) \frac{k}{\exp(n_{\text{traj}})} \\
&\quad + \sqrt{\frac{d \log(2nkN_v/\delta)}{2n_{\text{traj}}(1-\gamma)}}.
\end{aligned}$$

For each term individually, we can see that it converges to 0 as $\min(k, n_{\text{traj}}) \rightarrow \infty$:

$$\lim_{\min(k, n_{\text{traj}}) \rightarrow \infty} \sum_{\mathbf{v}: P(\mathbf{v}) > 0} (1 - P(\mathbf{v}))^k = \lim_{k \rightarrow \infty} \sum_{\mathbf{v}: P(\mathbf{v}) > 0} (1 - P(\mathbf{v}))^k = 0,$$

$$\lim_{\min(k, n_{\text{traj}}) \rightarrow \infty} \underbrace{\exp(-2\epsilon_{\min}^2(1-\gamma)/d)}_{\leq 1} \frac{k}{\exp(n_{\text{traj}})}$$

$$\leq \lim_{\min(k, n_{\text{traj}}) \rightarrow \infty} \frac{k}{\exp(n_{\text{traj}})} = 0,$$

$$\lim_{\min(k, n_{\text{traj}}) \rightarrow \infty} \sqrt{\frac{d \log(2nkN_v/\delta)}{2n_{\text{traj}}(1-\gamma)}} = 0.$$

Hence, $\lim_{\min(k, n_{\text{traj}}) \rightarrow \infty} \mathbb{E}_r[\mathcal{R}(r, S_k)] = 0$. \square

Theorem C.1.2 (Convergence under Boltzmann noise, estimated feature expectations). Under Assumption 5.3.2, if we estimate the feature expectations of at least

$$k > \log(\delta / (2f_v(d, n))) / (\log(1 - \exp(-\beta d / (1 - \gamma))))$$

expert policies using at least

$$n_{\text{traj}} > d \log(2f_v(d, n) / \delta) / (2\epsilon^2(1 - \gamma))$$

samples each, we have $P(\mathcal{R}(r, S_k) > \epsilon) \leq \delta$, where $f_v(d, n)$ is an upper bound on the number of vertices of the true safe set. In particular, we have

$$\lim_{\min(k, n_{\text{traj}}) \rightarrow \infty} \mathbb{E}_r[\mathcal{R}(r, S_k)] = 0.$$

Proof. We can decompose the probability of incurring regret greater than ϵ into the same three cases discussed in the proof of Theorem C.1.1 to get the upper-bound

$$P(\mathcal{R}(r, S_k) > \epsilon) \leq \sum_{\text{vertex } \mathbf{v}} \left(\underbrace{(1 - P(\mathbf{v}))^k P(\mathbf{v})}_{\text{Case 1}} + \underbrace{\exp(-2\epsilon^2 n_{\text{traj}}(1 - \gamma)/d)}_{\text{Case 2}} + \underbrace{0}_{\text{Case 3}} \right).$$

Under Assumption 5.3.2, we can use Lemma C.1.2, to get

$$P(\mathbf{v}) \geq \exp(-\beta d/(1 - \gamma)) := \Delta.$$

Combining both bounds, we get

$$P(\mathcal{R}(r, S_k) > \epsilon) \leq \sum_{\text{vertex } \mathbf{v}} \left((1 - \Delta)^k + \exp(-2\epsilon^2 n_{\text{traj}}(1 - \gamma)/d) \right).$$

To ensure $P(\mathcal{R}(r, S_k) > \epsilon) \leq \delta$ it is sufficient to ensure both $(1 - \Delta)^k \leq \delta/(2N_v)$ and $\exp(-2\epsilon^2 n_{\text{traj}}(1 - \gamma)/d) \leq \delta/(2N_v)$ where N_v is the number of vertices of the true safe set. This is true once

$$k > \frac{\log(\delta/(2N_v))}{\log(1 - \Delta)} = \frac{\log(\delta/(2N_v))}{\log(1 - \exp(-\beta d/(1 - \gamma)))},$$

and

$$n_{\text{traj}} > \frac{d \log(2N_v/\delta)}{2\epsilon^2(1 - \gamma)},$$

where can again replace N_v by a suitable upper bound.

For the regret, we get asymptotic optimality with the same argument from the proof of Theorem C.1.1, replacing $P(\mathbf{v})$ with the constant Δ . \square

C.2 IMPLEMENTATION DETAILS

This section discusses a few details of our implementation of CoCoRL. In particular, we highlight practical modifications related to constructing the convex hull S : how we handle degenerate sets of demonstrations and how we greedily select which points to use to construct the

convex hull. Algorithm 9 shows full pseudocode for our implementation of CoCoRL.

C.2.1 Constructing the Convex Hull

Algorithm 9 Convex Constraint Learning for Reinforcement Learning

```

1: function constraint_learning( $\mathcal{D}$ ,  $n_{\text{points}}$ ,  $d_{\text{stop}}$ )
2:    $(i, d_{\text{next}}) \leftarrow (0, \infty)$ 
3:    $x_{\text{next}} \leftarrow$  random starting point from  $\mathcal{D}$ 
4:   while  $i \leq n_{\text{points}}$  and  $|\mathcal{D}| > 0$  and  $d_{\text{next}} > d_{\text{stop}}$  do
5:      $\bar{\mathcal{D}} \leftarrow \bar{\mathcal{D}} \cup \{x_{\text{next}}\}$ ,  $\mathcal{D} \leftarrow \mathcal{D} \setminus \{x_{\text{next}}\}$ ,  $i \leftarrow i + 1$ 
6:      $S \leftarrow \text{convex\_hull}(\bar{\mathcal{D}})$ 
7:      $(x_{\text{next}}, d_{\text{next}}) \leftarrow \text{furthest\_point}(\mathcal{D}, S)$ 
8:      $\mathcal{U} \leftarrow \text{unsafe\_set}(S)$ 
9:   return  $S, \mathcal{U}$ 

10: function furthest_point( $\mathcal{X}, \mathcal{P}$ )
11:    $d_{\text{max}} = -\infty$ 
12:   for  $x \in \mathcal{X}$  do
13:     Solve QP to find  $d \in \text{argmin}_{p \in \mathcal{P}} (x - p)^2$ 
14:     if  $d > d_{\text{max}}$  then  $d_{\text{max}} \leftarrow d$ ,  $x_{\text{max}} \leftarrow x$ 
15:   return  $(x_{\text{max}}, d_{\text{max}})$ 

16: function convex_hull( $\mathcal{X}$ )
17:   if  $|\mathcal{X}| < 3$  then return special case solution  $A, \mathbf{b}$ 
18:   Determine effective dimension of  $\mathcal{X}$ 
19:    $\mathcal{X}_{\text{proj}} \leftarrow$  project  $\mathcal{X}$  to lower dimensional space
20:    $H_{\text{proj}} \leftarrow$  call Qhull to obtain convex hull of  $\mathcal{X}_{\text{proj}}$ 
21:    $A_{\text{proj}}, \mathbf{b}_{\text{proj}} \leftarrow$  call Qhull to obtain linear equations for  $H_{\text{proj}}$ 
22:    $A, \mathbf{b} \leftarrow$  project  $A_{\text{proj}}, \mathbf{b}_{\text{proj}}$  back to  $\mathbb{R}^d$ 
23:   return  $A, \mathbf{b}$ 

```

We use the Quickhull algorithm [117] to construct convex hulls. In particular, we use a standard implementation, called *Qhull*, which can construct the convex hull from a set of points and determine the linear equations. However, it assumes at least 3 input points, and non-degenerate inputs, i.e., it assumes that $\text{rank}(\mathbf{f}(\pi_1^*), \dots, \mathbf{f}(\pi_k^*)) = d$. To avoid these limiting cases, we make two practical modifications.

SPECIAL-CASE SOLUTIONS FOR 1 AND 2 INPUTS. For less than 3 input points it is easy to construct the linear equations describing the convex hull manually. If we only see a single demonstration π_1^* , the convex hull is simply, $\text{conv}(\mathbf{f}(\pi_1^*)) = \{\mathbf{f}(\pi_1^*)\} = \{x | Ax \leq \mathbf{b}\}$ with

$$A = [I_d, -I_d]^T$$

$$\mathbf{b} = [\mathbf{f}(\pi_1^*)^T, \mathbf{f}(\pi_1^*)^T]^T.$$

If we observe 2 demonstrations π_1^*, π_2^* , the convex hull is given by

$$\text{conv}(\mathbf{f}(\pi_1^*), \mathbf{f}(\pi_2^*))$$

$$= \{\mathbf{f}(\pi_1^*) + \lambda(\mathbf{f}(\pi_2^*) - \mathbf{f}(\pi_1^*)) | 0 \leq \lambda \leq 1\} = \{x | Ax \leq \mathbf{b}\},$$

where

$$A = [W, -W, \mathbf{v}^T, -\mathbf{v}^T]^T$$

$$\mathbf{b} = [W\mathbf{f}(\pi_1^*), -W\mathbf{f}(\pi_1^*), \mathbf{v}^T\mathbf{f}(\pi_2^*), -\mathbf{v}^T\mathbf{f}(\pi_1^*)]^T.$$

Here, $\mathbf{v} = \mathbf{f}(\pi_2^*) - \mathbf{f}(\pi_1^*)$, and $W = [\mathbf{w}_1, \dots, \mathbf{w}_{d-1}]$ spans the space orthogonal to \mathbf{v} , i.e., $\mathbf{w}_i^T \mathbf{v} = 0$.

HANDLING DEGENERATE SAFE SETS. If $\text{rank}(\mathbf{f}(\pi_1^*), \dots, \mathbf{f}(\pi_k^*)) < d$, we first project the demonstrations to a lower-dimensional subspace in which they are full rank, then we call *Qhull* to construct the convex hull in this space. Finally, we project back the linear equations describing the convex hull to \mathbb{R}^d . To determine the correct projection, let us define the matrix $D = [\mathbf{f}(\pi_1^*)^T, \dots, \mathbf{f}(\pi_k^*)^T]^T \in \mathbb{R}^{k \times d}$. Now, we can do the singular value decomposition (SVD): $D = U^T \Sigma V$, where $U = [\mathbf{u}_1, \dots, \mathbf{u}_k] \in \mathbb{R}^{k \times k}$, $V = [\mathbf{v}_1, \dots, \mathbf{v}_d] \in \mathbb{R}^{d \times d}$, and

$\Sigma = \text{diag}(\sigma_1, \dots, \sigma_d) \in \mathbb{R}^{k \times d}$. Assuming the singular values are ordered in decreasing order by magnitude, we can determine the effective dimension \tilde{d} of the demonstrations such that $\sigma_{\tilde{d}} > 0$ and $\sigma_{\tilde{d}+1} = 0$ (for numerical stability, we use a small positive number instead of 0). Now, we can define projection matrices

$$X_{\text{ef}} = [\mathbf{u}_1, \dots, \mathbf{u}_{\tilde{d}}]^T \in \mathbb{R}^{\tilde{d} \times d},$$

$$X_{\text{orth}} = [\mathbf{u}_{\tilde{d}+1}, \dots, \mathbf{u}_d]^T \in \mathbb{R}^{(d-\tilde{d}) \times d},$$

where X_{ef} projects to the span of the demonstrations, and X_{orth} projects to the complement. Using X_{ef} , we can project the demonstrations to their span and construct the convex hull in that projection, $\text{conv}(X_{\text{ef}}D) = \{x | A_{\text{proj}}x \leq \mathbf{b}_{\text{proj}}\}$. To project the convex hull back to \mathbb{R}^d , we construct linear equations in \mathbb{R}^d , such that $\text{conv}(D) = \{x | Ax \leq \mathbf{b}\}$:

$$A = [(X_{\text{ef}}^T A_{\text{proj}})^T, X_{\text{orth}}, -X_{\text{orth}}]^T$$

$$\mathbf{b} = [\mathbf{b}_{\text{proj}}, X_{\text{orth}}\mathbf{f}(\pi_1^*), -X_{\text{orth}}\mathbf{f}(\pi_1^*)],$$

where $X_{\text{ef}}^T A_{\text{proj}}$ projects the linear equations back to \mathbb{R}^d , and the other two components ensure that $X_{\text{orth}}x = X_{\text{orth}}\mathbf{f}(\pi_1^*)$, i.e., restricts the orthogonal components to the convex hull. In practice, we change this constraint to $-X_{\text{orth}}\mathbf{f}(\pi_1^*) - \epsilon \leq X_{\text{orth}}x \leq X_{\text{orth}}\mathbf{f}(\pi_1^*) + \epsilon$ for some small $\epsilon > 0$.

c.2.2 Iteratively Adding Points

Constructing the safe set from all demonstrations can sometimes be problematic, especially if it results in too many inferred constraints. This scenario is more likely to occur when demonstrations are clustered closely together in feature space, which occurs, e.g., if they are not exactly optimal. To mitigate such problems, we adopt an iterative approach for adding points to the safe set. We start with a random point from the set of demonstrations, and subsequently, we iteratively add the point farthest away from the existing safe set. We

can determine the distance of a given demonstration π by solving the quadratic program

$$d \in \operatorname{argmin}_{x \in S} (x - \mathbf{f}(\pi))^2.$$

We solve this problem for each remaining demonstration from the safe set to determine which point to add next. We stop adding points once the distance of the next point we would add gets too small. By changing the stopping distance as a hyperparameter, we can trade-off between adding all points important for expanding the safe set and regularizing the safe set.

C.3 EXPERIMENT DETAILS

In this section, we provide more details about our experimental setup in the Gridworld environments (Appendix C.3.1) and the driving environment (Appendix C.3.2). In particular, we provide details on the environments, how we solve them, and the computational resources used.

C.3.1 Gridworld Experiments

ENVIRONMENT DETAILS. An $N \times N$ Gridworld has N^2 discrete states and a discrete action space with 5 actions: *left*, *right*, *up*, *down*, and *stay*. Each action corresponds to a movement of the agent in the grid. Given an action, the agent moves in the intended direction except for two cases: (1) if the agent would leave the grid, it instead stays in its current cell, and (2) with probability p , the agent takes a random action instead of the intended one.

In the Gridworld environments, we use the state-action occupancies as features and define rewards and costs independently per state. We uniformly sample n_{goal} and n_{limited} goal tiles and limited tiles, respectively. Each goal cell has an average reward of 1; other tiles have an average reward of 0. Constraints are associated with limited tiles, which the agent must avoid. We uniformly sample the threshold

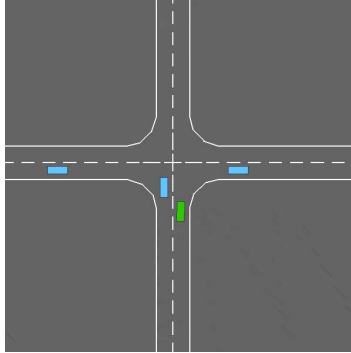


FIGURE C.1: highway-env intersection environment.

for each constraint. We ensure feasibility using rejection sampling, i.e., we resample the thresholds if there is no feasible policy. In our experiments, we use $N = 3$, $n_{\text{goal}} = 2$, $n_{\text{limited}} = 3$, and we have $n = 2$ different constraints. We choose discount factor $\gamma = 0.9$.

While the constraints are fixed and shared for one Gridworld instance, we sample different reward functions from a Gaussian with standard deviation 0.1 and mean 1 for goal tiles and 0 for non-goal tiles. To test reward transfer, we sample a different set of goal tiles from the grid for evaluation.

For the experiment without constraint transfer, we choose $p = 0$, i.e., a deterministic environment. To test transfer to a new environment, we use $p = 0$ during training and $p = 0.2$ during evaluation.

C.3.2 Driving Experiments

ENVIRONMENT DETAILS. We use the intersection environment provided by highway-env [118] shown in Figure C.1. The agent controls the green car, while the environment controls the blue cars. The action space has high-level actions for speeding up and slowing down and three actions for choosing one of three trajectories: turning left, turning right, and going straight. The observation space for the policy contains the position and velocity of the agent and other cars.

The default environment has a well-tuned reward function with positive terms for reaching a goal and high velocity and negative terms to avoid crashes and staying on the road. We change the environment to have a reward function that rewards reaching the goal and includes driver preferences on velocity and heading angle and multiple cost functions that limit bad events, such as driving too fast or off the road. To define the reward and cost functions, we define a set of features:

$$\mathbf{f}(s, a) = \begin{pmatrix} f_{\text{left}}(s, a) \\ f_{\text{straight}}(s, a) \\ f_{\text{right}}(s, a) \\ f_{\text{vel}}(s, a) \\ f_{\text{heading}}(s, a) \\ f_{\text{toofast}}(s, a) \\ f_{\text{tooclose}}(s, a) \\ f_{\text{collision}}(s, a) \\ f_{\text{offroad}}(s, a) \end{pmatrix} \in \mathbb{R}^9$$

where $f_{\text{left}}, f_{\text{straight}}, f_{\text{right}}$ are 1 if the agent reached the goal after turning left, straight, or right respectively, and 0 else. f_{vel} is the agents velocity and f_{heading} is its heading angle.

The last four features are indicators of undesired events. f_{toofast} is 1 if the agent exceeds the speed limit, f_{tooclose} is 1 if the agent is too close to another car, $f_{\text{collision}}$ is 1 if the agent has collided with another car, and f_{offroad} is 1 if the agent is not on the road.

The ground truth constraint in all of our experiments is defined by:

$$\begin{aligned} \phi_1 &= (0, 0, 0, 0, 0, 1, 0, 0, 0)^T \\ \phi_2 &= (0, 0, 0, 0, 0, 0, 1, 0, 0)^T \\ \phi_3 &= (0, 0, 0, 0, 0, 0, 0, 1, 0)^T \\ \phi_4 &= (0, 0, 0, 0, 0, 0, 0, 0, 1)^T \end{aligned}$$

and thresholds

$$\zeta_1 = 0.2, \quad \zeta_2 = 0.2, \quad \zeta_3 = 0.05, \quad \zeta_4 = 0.1,$$

i.e., we have one constraint for each of the last four features, and each constraint restricts how often this feature can occur. For example, collisions should occur in fewer than 5% of steps and speed limit violations in fewer than 20% of steps.

The driver preferences are a function of the first five features. Each driver wants to reach one of the three goals (sampled uniformly) and has a preference about velocity and heading angle sampled from $\theta_{\text{vel}} \sim \mathcal{N}(0.1, 0.1)$, and $\theta_{\text{heading}} \sim \mathcal{N}(-0.2, 0.1)$. These preferences simulate variety between the demonstrations from different drivers.

When inferring constraints, for simplicity, we restrict the feature space to $(f_{\text{toofast}}, f_{\text{tooclose}}, f_{\text{collision}}, f_{\text{offroad}})$. Our approach also works for all features but needs more (and more diverse) samples to learn that the other features are safe.

To test constraint transfer to a new reward function, we sample goals in demonstrations to be either f_{left} or f_{right} , while during evaluation the goal is always f_{straight} .

To test constraint transfer to a modified environment, we collect demonstrations with more defensive drivers (that keep larger distances to other vehicles) and evaluate the learned constraints with more aggressive drivers (that keep smaller distances to other vehicles).

DRIVING CONTROLLERS. We use a family of parameterized driving controllers for two purposes: (1) to optimize over driving policies, and (2) to control other vehicles in the environment. The controllers greedily decide which trajectory to choose, i.e., choose the goal with the highest reward, and they control acceleration via a linear equation with parameter vector $\omega \in \mathbb{R}^5$. As the vehicles generally follow a fixed trajectory, these controllers behave similarly to an *intelligent driver model* (IDM). Specifically, we use linearized controllers proposed by in Leurent *et al.* [145]; see their Appendix B for details about the parameterization.

CONSTRAINED CROSS-ENTROPY METHOD SOLVER. We solve the driving environment by optimizing over the parametric controllers, using a constrained cross-entropy method (CEM; [119, 146]). The constrained CEM ranks candidate solutions first by constraint violations and then ranks feasible solutions by reward. It aims first to find a feasible solution and then improve upon it within the feasible set. We extend the algorithm by Wen and Topcu [119] to handle multiple constraints by ranking first by the number of violated constraints, then the magnitude of the violation, and only then by reward. Algorithm 10 shows the pseudocode for this algorithm.

COMPUTATIONAL RESOURCES. The computational cost of our experiments is dominated by optimizing policies using the CEM. The IRL baselines are significantly more expensive because they need to do so in the inner loop. To combat this, we parallelize evaluating candidate policies in the CEM, performing 50 roll-outs in parallel. We run experiments on AMD EPYC 64-Core processors. Any single experiment using CoCoRL finishes in approximately 5 hours, while any experiment using the IRL baselines finishes in approximately 20 hours. For each of the 3 constraint transfer setups, we run 5 random seeds for 30 different numbers of demonstrations k , i.e., we run 450 experiments in total for each method we test.

Algorithm 10 Cross-entropy method for (constrained) RL, based on Wen and Topcu [119].

Require: $n_{\text{iter}}, n_{\text{samp}}, n_{\text{elite}}$

- 1: Initialize policy parameters $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d$.
 - 2: **for** iteration = 1, 2, \dots , n_{iter} **do**
 - 3: Sample n_{samp} samples of $\omega_i \sim \mathcal{N}(\mu, \text{diag}(\sigma))$
 - 4: Evaluate policies $\omega_1, \dots, \omega_{n_{\text{samp}}}$ in the environment
 - 5: **if** constrained problem **then**
 - 6: Compute number of violations $N_{\text{viol}}(\omega_i) = \sum_j \mathbb{1}_{\{J_j(\omega_i) > 0\}}$
 - 7: Compute total violation $T_{\text{viol}}(\omega_i) = \sum_j \max(J_j(\omega_i), 0)$
 - 8: Sort ω_i in descending order first by N_{viol} , then by T_{viol}
 - 9: Let E be the first n_{elite} policies
 - 10: **if** $N_{\text{viol}}(\omega_{n_{\text{elite}}}) = 0$ **then**
 - 11: Sort $\{\omega_i | N_{\text{viol}}(\omega_i) = 0\}$ in descending order of $G(\omega_i)$
 - 12: Let E be the first n_{elite} policies
 - 13: **else**
 - 14: Sort ω_i in descending order of return $G(\omega_i)$
 - 15: Let E be the first n_{elite} policies
 - 16: Fit Gaussian with mean μ and diagonal covariance σ to E
 - 17: **return** μ
-

ADAPTIVE CONSTRAINT LEARNING

This appendix provides additional details about Chapter 6. We provide proofs of all theoretical results (Appendix D.1), and more details about the experimental setup (Appendix D.2).

D.1 PROOFS

This section provides the full proofs of our key results of Chapter 6: the sample complexity lower bound for CBAI problems (Appendix D.1.1) and the sample complexity of ACOL (Appendix D.1.2).

D.1.1 Lower Bounds

Theorem 6.2.1 (CBAI lower bound). Assume $\eta_x \sim \mathcal{N}(0, 1)$ for all $x \in \mathcal{X}$. For any CBAI problem $\nu = (\mathcal{X}, \theta, \phi)$, there exists another CBAI problem $\nu' = (\mathcal{X}, \theta, \phi')$ with the same set of actions \mathcal{X} and reward parameter θ but a different constraint parameter and optimal arm, such that the expected number of iterations τ needed by any allocation strategy that can distinguish between ν and ν' with probability at least $1 - \delta$ is lower bounded as

$$\mathbb{E}[\tau] \geq 2 \log \left(\frac{1}{2.4\delta} \right) \max_{x \in \mathcal{X}_\theta^\geq(x_v^*)} \frac{\|x\|_{A_\lambda^{-1}}^2}{(\phi^T x)^2},$$

where λ is a probability distribution over arms of the allocation strategy, i.e., $\lambda(x)$ is the probability that it pulls arm x , $A_\lambda = \sum_x \lambda(x) x x^T$ is the design matrix, and $\mathcal{X}_\theta^\geq(x_v^*) = \{x' \in \mathcal{X} \mid \theta^T x' \geq \theta^T x_v^*\}$ is the set of all arms with reward no less than x_v^* , the optimal arm for ν .

Proof. Our proof has a similar structure to the proof of Theorem 3.1 by Soare [134]. Let us denote the optimal arm of problem ν with

x_ν^* and the optimal arm of ν' with $x_{\nu'}^*$. Let \mathcal{A} be a δ -PAC algorithm to solve constrained linear bandit problems, and let A be the event that \mathcal{A} recommends x_ν^* as the optimal arm. If we denote by $P_\nu(A)$ the probability of A happening for instance ν , and by $P_{\nu'}(A)$ the probability for instance ν' , we have $P_\nu(A) \geq 1 - \delta$ and $P_{\nu'}(A) \leq \delta$.

Let $\tilde{\varepsilon} = \phi' - \phi$, and let τ be the stopping time of \mathcal{A} . Let (x_1, \dots, x_τ) be the sequence of arms \mathcal{A} pulls and (z_1, \dots, z_τ) the corresponding observed noisy constraint values $z_i = x_i^T \phi + \eta_{x_i}$ with $\eta_{x_i} \sim \mathcal{N}(0, 1)$ being independent Gaussian noise.

Now, consider the log-likelihood ratio of these observations under algorithm \mathcal{A} :

$$\begin{aligned}
L_\tau &= \log \left(\prod_{s=1}^{\tau} \frac{P_\nu(z_s | x_s)}{P_{\nu'}(z_s | x_s)} \right) = \sum_{s=1}^{\tau} \log \left(\frac{P_\nu(z_s | x_s)}{P_{\nu'}(z_s | x_s)} \right) \\
&= \sum_{s=1}^{\tau} \log \left(\frac{P_\nu(\eta_s)}{P_{\nu'}(\eta'_s)} \right) = \sum_{s=1}^{\tau} \log \left(\frac{\exp(-\eta_s^2/2)}{\exp(-\eta'_s{}^2/2)} \right) \\
&= \sum_{s=1}^{\tau} \frac{1}{2} ((z_s - x_s^T \phi')^2 - (z_s - x_s^T \phi)^2) \\
&= \sum_{s=1}^{\tau} \frac{1}{2} (z_s^2 - 2z_s x_s^T \phi' + (x_s^T \phi')^2 - z_s^2 + 2z_s x_s^T \phi - (x_s^T \phi)^2) \\
&= \sum_{s=1}^{\tau} \frac{1}{2} (2z_s x_s^T (\phi - \phi') + (x_s^T \phi' - x_s^T \phi)(x_s^T \phi' + x_s^T \phi)) \\
&= \sum_{s=1}^{\tau} \frac{1}{2} (-2z_s x_s^T \tilde{\varepsilon} + x_s^T \tilde{\varepsilon} (x_s^T \phi + x_s^T \tilde{\varepsilon} + x_s^T \phi)) \\
&= \sum_{s=1}^{\tau} (x_s^T \tilde{\varepsilon}) \frac{-2z_s + 2x_s^T \phi + x_s^T \tilde{\varepsilon}}{2} = \sum_{s=1}^{\tau} (x_s^T \tilde{\varepsilon}) \left(\frac{x_s^T \tilde{\varepsilon}}{2} - \eta_s \right)
\end{aligned}$$

Taking the expectation of this log-likelihood ratio gives:

$$\begin{aligned}
\mathbb{E}_\nu[L_\tau] &= \mathbb{E}_\nu \left[\sum_{s=1}^{\tau} (x_s^T \tilde{\varepsilon}) \left(\frac{x_s^T \tilde{\varepsilon}}{2} - \eta_s \right) \right] = \frac{1}{2} \mathbb{E}_\nu \left[\sum_{s=1}^{\tau} (x_s^T \tilde{\varepsilon})^2 \right] - \underbrace{\mathbb{E}_\nu[\eta_s]}_{=0} \\
&= \frac{1}{2} \mathbb{E}_\nu \left[\sum_{s=1}^{\tau} \tilde{\varepsilon}^T x_s x_s^T \tilde{\varepsilon} \right] = \frac{1}{2} \mathbb{E}_\nu \left[\sum_{x \in \mathcal{X}} \mathbb{E}_\nu[\tau] \lambda(x) \tilde{\varepsilon}^T x x^T \tilde{\varepsilon} \right] \\
&= \frac{1}{2} \mathbb{E}_\nu[\tau] \mathbb{E}_\nu \left[\sum_{x \in \mathcal{X}} \lambda(x) \tilde{\varepsilon}^T x x^T \tilde{\varepsilon} \right] = \frac{1}{2} \mathbb{E}_\nu[\tau] \tilde{\varepsilon}^T A_\lambda \tilde{\varepsilon}
\end{aligned}$$

Next, we can apply Lemma 19 from Kaufmann *et al.* [147]:

$$\begin{aligned}
\mathbb{E}_\nu[L_\tau] &= \frac{1}{2} \mathbb{E}_\nu[\tau] \tilde{\varepsilon}^T A_\lambda \tilde{\varepsilon} \geq D_{\text{KL}}((\|P)_\nu(A), P_{\nu'}(A)) \geq \log \frac{1}{2.4\delta} \\
\mathbb{E}_\nu[\tau] &\geq 2 \log \left(\frac{1}{2.4\delta} \right) \frac{1}{\tilde{\varepsilon}^T A_\lambda \tilde{\varepsilon}} \tag{D.1}
\end{aligned}$$

To obtain a lower bound, we now aim to find the smallest $\tilde{\varepsilon}$ such that ν and ν' have different constrained optimal arms.

Let $\mathcal{X}_\theta^\geq(x) = \{x' \in \mathcal{X} \mid \theta^T x' \geq \theta^T x\}$ be the set of arms with higher reward than x . There are two ways we can modify ν to change its optimal arm. We can change ϕ to ϕ' such that either, **Case (i)**, the previous optimum x_ν^* becomes infeasible in ν' , or, **Case (ii)**, a solution $x_\nu^* \in \mathcal{X}_\theta^\geq(x_\nu^*)$ that was infeasible in ν is now feasible in ν' . We will consider both cases separately, and aim to find an $\tilde{\varepsilon}$ for each case that minimizes $\tilde{\varepsilon}^T A_\lambda \tilde{\varepsilon}$.

CASE (I). We want to find $\tilde{\varepsilon}$ that minimizes $\frac{1}{2} \tilde{\varepsilon}^T A_\lambda \tilde{\varepsilon}$ such that $\phi'^T x_\nu^* > 0$, i.e., the previously optimal arm becomes infeasible. We can write this constraint equivalently as

$$\begin{aligned}
\phi'^T x_\nu^* > 0 &\Leftrightarrow \phi^T x_\nu^* - \phi'^T x_\nu^* < \phi^T x_\nu^* \\
&\Leftrightarrow \varepsilon^T x_\nu^* < \phi^T x_\nu^* \Leftrightarrow \varepsilon^T x_\nu^* - \phi^T x_\nu^* < 0
\end{aligned}$$

Which results in the following optimization problem:

$$\min_{\varepsilon} \frac{1}{2} \varepsilon^T A_{\lambda} \varepsilon \quad \text{s.t.} \quad \varepsilon^T x_v^* - \phi^T x_v^* + \alpha \leq 0,$$

where $\alpha > 0$. The Lagrangian is $L(\varepsilon, \gamma) = \frac{1}{2} \varepsilon^T A_{\lambda} \varepsilon - \gamma(\varepsilon^T x_v^* - \phi^T x_v^* + \alpha)$, and requiring $\frac{\partial L}{\partial \varepsilon} = \frac{\partial L}{\partial \gamma} = 0$ yields:

$$\begin{aligned} \frac{\partial L}{\partial \varepsilon} = A_{\lambda} \varepsilon - \gamma x_v^* = 0 &\Leftrightarrow A_{\lambda} \varepsilon = \gamma x_v^* \Leftrightarrow A_{\lambda}^{\frac{1}{2}} \varepsilon = \gamma A_{\lambda}^{-\frac{1}{2}} x_v^* \\ \frac{\partial L}{\partial \gamma} = \varepsilon^T x_v^* - \phi^T x_v^* + \alpha = 0 &\Leftrightarrow \varepsilon^T x_v^* = \phi^T x_v^* - \alpha \end{aligned}$$

From the first equation, it follows that

$$\begin{aligned} x_v^{*T} \varepsilon &= x_v^{*T} A_{\lambda}^{-\frac{1}{2}} A_{\lambda}^{\frac{1}{2}} \varepsilon = \gamma x_v^{*T} A_{\lambda}^{-1} x_v^* = \gamma \|x_v^*\|_{A_{\lambda}^{-1}}^2 \\ x_v^{*T} \varepsilon &= x_v^{*T} A_{\lambda}^{-\frac{1}{2}} A_{\lambda}^{\frac{1}{2}} \varepsilon = \frac{1}{\gamma} \varepsilon^T A_{\lambda} \varepsilon = \frac{1}{\gamma} \|\varepsilon\|_{A_{\lambda}}^2 \end{aligned}$$

and therefore

$$\begin{aligned} x_v^{*T} \varepsilon &= \|x_v^*\|_{A_{\lambda}^{-1}} \|\varepsilon\|_{A_{\lambda}} = \phi^T x_v^* - \alpha \\ \|\varepsilon\|_{A_{\lambda}} &= \frac{\phi^T x_v^* - \alpha}{\|x_v^*\|_{A_{\lambda}^{-1}}} > \frac{\phi^T x_v^*}{\|x_v^*\|_{A_{\lambda}^{-1}}} \end{aligned}$$

where the last inequality follows because $\alpha > 0$ and A_{λ} is positive definite.

CASE (II). We want to find $\tilde{\varepsilon}$ that minimizes $\frac{1}{2} \varepsilon^T A_{\lambda} \varepsilon$ such that there exists an $x \in \mathcal{X}$ for which $\theta^T x > \theta^T x_v^*$ and $\phi'^T x \leq 0$, i.e., x has higher reward than x_v^* and it is feasible in v' . We can write these constraints as

$$\begin{aligned} \theta^T x > \theta^T x_v^* &\Leftrightarrow \theta^T (x - x_v^*) + \alpha \leq 0 \\ \phi'^T x \leq 0 &\Leftrightarrow \varepsilon^T x + \phi'^T x \leq 0 \end{aligned}$$

with $\alpha > 0$. This results in the following optimization problem:

$$\begin{aligned} \min_{\varepsilon} \quad & \frac{1}{2} \varepsilon^T A_{\lambda} \varepsilon \\ \text{s.t.} \quad & \exists x : \theta^T(x_v^* - x) + \alpha \leq 0 \\ & \varepsilon^T x + \phi^T x \leq 0 \end{aligned}$$

The Lagrangian of this problem is

$$L(\varepsilon, \gamma, \delta) = \frac{1}{2} \varepsilon^T A_{\lambda} \varepsilon - \gamma(\theta^T(x_v^* - x) + \alpha) - \delta(\varepsilon^T x + \phi^T x)$$

Requiring $\frac{\partial L}{\partial \varepsilon} = \frac{\partial L}{\partial \delta} = 0$ results in

$$\begin{aligned} \frac{\partial L}{\partial \varepsilon} = A_{\lambda} \varepsilon - \delta x = 0 & \Leftrightarrow A_{\lambda} \varepsilon = \delta x \Leftrightarrow A_{\lambda}^{\frac{1}{2}} \varepsilon = \delta A_{\lambda}^{-\frac{1}{2}} x \\ \frac{\partial L}{\partial \delta} = \varepsilon^T x + \phi^T x = 0 & \Leftrightarrow \varepsilon^T x = -\phi^T x \end{aligned}$$

It follows that

$$\begin{aligned} x^T \varepsilon &= x^T A_{\lambda}^{-\frac{1}{2}} A_{\lambda}^{\frac{1}{2}} \varepsilon = \delta x^T A_{\lambda}^{-1} x = \delta \|x\|_{A_{\lambda}^{-1}}^2 \\ x^T \varepsilon &= x^T A_{\lambda}^{-\frac{1}{2}} A_{\lambda}^{\frac{1}{2}} \varepsilon = \frac{1}{\delta} \varepsilon^T A_{\lambda} \varepsilon = \frac{1}{\delta} \|\varepsilon\|_{A_{\lambda}}^2 \end{aligned}$$

and therefore

$$x^T \varepsilon = \|x\|_{A_{\lambda}^{-1}} \|\varepsilon\|_{A_{\lambda}} = \phi^T x \Rightarrow \|\varepsilon\|_{A_{\lambda}} = \frac{\phi^T x}{\|x\|_{A_{\lambda}^{-1}}}$$

Combining this result with the remaining constraint $\theta^T x > \theta^T x_v^*$ which implies $x \in \mathcal{X}_{\theta}^{\geq}(x_v^*)$, we can conclude

$$\|\varepsilon\|_{A_{\lambda}} \geq \min_{x \in \mathcal{X}_{\theta}^{\geq}(x_v^*)} \frac{\phi^T x}{\|x\|_{A_{\lambda}^{-1}}}$$

COMBINING CASES (I) AND (II). We can conclude that the ε that minimizes $\|\varepsilon\|_{A_\lambda}^2$ while still ensuring that ν' has a different solution than ν , satisfies:

$$\|\varepsilon\|_{A_\lambda} \geq \min \left[\underbrace{\frac{\phi^T x_\nu^*}{\|x_\nu^*\|_{A_\lambda^{-1}}}}_{\text{Case (i)}}, \underbrace{\min_{x \in \mathcal{X}_\theta^\geq(x_\nu^*)} \frac{\phi^T x}{\|x\|_{A_\lambda^{-1}}}}_{\text{Case (ii)}} \right]$$

But because $x_\nu^* \in \mathcal{X}_\theta^\geq(x_\nu^*)$, it is simply

$$\|\varepsilon\|_{A_\lambda} \geq \min_{x \in \mathcal{X}_\theta^\geq(x_\nu^*)} \frac{\phi^T x}{\|x\|_{A_\lambda^{-1}}}$$

Combining this result with equation (D.1), gives the final bound:

$$\begin{aligned} \mathbb{E}_\nu[\tau] &\geq 2 \log \left(\frac{1}{2.4\delta} \right) \frac{1}{\left(\min_{x \in \mathcal{X}_\theta^\geq(x_\nu^*)} \frac{\phi^T x}{\|x\|_{A_\lambda^{-1}}} \right)^2} \\ &= 2 \log \left(\frac{1}{2.4\delta} \right) \max_{x \in \mathcal{X}_\theta^\geq(x_\nu^*)} \frac{\|x\|_{A_\lambda^{-1}}^2}{(\phi^T x)^2} \end{aligned}$$

□

Next, we derive the worst case bound on the quantity making up the CBAI lower bound.

Proposition 6.2.1. For any CBAI problem ν , we have $H_{\text{CLB}}(\nu) \leq d/(C_{\min}^+)^2$, where $C_{\min}^+ = \min_{x \in \mathcal{X}} |\phi^T x|$. This bound is tight, i.e, there is an instance ν , such that we have $H_{\text{CLB}}(\nu) = d/(C_{\min}^+)^2$.

Proof.

$$\begin{aligned} H_{\text{CLB}}(\nu) &= \min_{\lambda} \max_{x \in \mathcal{X}_\theta^\geq(x_\nu^*)} \frac{\|x\|_{A_{\lambda^*}^{-1}}^2}{(\phi^T x)^2} \\ &\leq \frac{1}{C_{\min}^+{}^2} \min_{\lambda} \max_{x \in \mathcal{X}_\theta^\geq(x_\nu^*)} \|x\|_{A_{\lambda^*}^{-1}}^2 \leq \frac{d}{C_{\min}^+{}^2} \end{aligned}$$

where the last inequality uses the well-known result by Kiefer and Wolfowitz [148]. Equality holds, for example, if all $x \in \mathcal{X}$ are linearly independent and have the same constraint value C_{\min}^+ . □

D.1.2 Adaptive Constraint Learning

In this section, we analyse the sample complexity of ACOL and prove our main result.

Theorem 6.2.4 (ACOL sample complexity). Assume Algorithm 5 is implemented with an ε -approximate rounding strategy. Then, after N iterations the algorithm returns an optimal arm with probability at least $1 - \delta$, and we have:

$$\begin{aligned} N &\leq 8 \log \left(\frac{|\mathcal{X}| \bar{t}^2}{\delta^2} \right) (1 + \varepsilon) \sum_{t=1}^{\bar{t}} \min_{\lambda} \max_{x \in \mathcal{U}_t} \frac{\|x\|_{A_{\lambda}^{-1}}^2}{(\phi^T x)^2} + \bar{t} \\ &\leq 8 \log \left(\frac{|\mathcal{X}| \bar{t}^2}{\delta^2} \right) (1 + \varepsilon) \bar{t} \bar{H}_{\text{CLB}}(v) + \bar{t} \end{aligned}$$

where

$$\begin{aligned} \bar{H}_{\text{CLB}}(v) &= \min_{\lambda} \max_{x \in \mathcal{X}} \|x\|_{A_{\lambda}^{-1}}^2 / (\phi^T x)^2, \\ \bar{t} &= \lceil -\log_2 C_{\min}^+ \rceil. \end{aligned}$$

Moreover, $\bar{H}_{\text{CLB}}(v) \leq d / (C_{\min}^+)^2$.

Proof. Let $\mathcal{E}_t := \{\mathcal{U}_t \subseteq \mathcal{F}_t\}$ where $\mathcal{F}_t := \{x \in \mathcal{X} | u_{\phi}^t(x) - l_{\phi}^t(x) \leq 2^{-t}\}$. So, \mathcal{E}_t is the event that all arms in \mathcal{U}_t have confidence interval smaller than 2^{-t} . We will first show that $P(\mathcal{E}_1) \geq 1 - \delta_1$ and $P(\mathcal{E}_t | \mathcal{E}_{t-1}) \geq 1 - \delta_t$, which ensures that the set of arms we are uncertain about shrinks exponentially in the rounds t .

Let $x \in \mathcal{U}_t$. Then, using Proposition 6.2.2, and the ε -approximate rounding strategy, it holds with probability at least $1 - \delta_t$ that:

$$u_{\phi}^t(x) - l_{\phi}^t(x) \leq 2 \sqrt{2 \log \left(\frac{|\mathcal{X}|}{\delta_t} \right) \frac{1 + \varepsilon}{N_t}} \|x\|_{A_{\lambda_t^*}^{-1}}$$

Using the length of a round $N_t = \left\lceil 2^{2t+3} \log \left(\frac{|\mathcal{X}|}{\delta_t} \right) (1 + \varepsilon) \rho_t^* \right\rceil$, and that we select arms to reduce uncertainty in \mathcal{U}_t , we get

$$\begin{aligned} u_\phi^t(x) - l_\phi^t(x) &\leq 2^{-t} \sqrt{\left(\min_{\lambda} \max_{\tilde{x} \in \mathcal{U}_t} \|\tilde{x}\|_{A_\lambda^{-1}}^2 \right)^{-1}} \|x\|_{A_{\lambda_t^*}^{-1}} \\ &\leq 2^{-t} \sqrt{\left(\min_{\lambda} \max_{\tilde{x} \in \mathcal{U}_t} \|\tilde{x}\|_{A_\lambda^{-1}}^2 \right)^{-1}} \left(\min_{\lambda} \max_{\tilde{x} \in \mathcal{U}_t} \|\tilde{x}\|_{A_\lambda^{-1}} \right) \leq 2^{-t} \end{aligned}$$

Note, that x can only be in \mathcal{U}_t if $u_\phi^t(x) > 0$ and $l_\phi^t(x) \leq 0$. It follows that $P(\mathcal{E}_t | \mathcal{E}_{t-1}) \geq 1 - \delta_t$.

Now consider round $\bar{t} := \left\lceil \log_2 \frac{1}{C_{\min}^+} \right\rceil$. We show $P(\mathcal{U}_{\bar{t}} = \{\} | \mathcal{E}_{\bar{t}}) = 1$. Assume $\mathcal{E}_{\bar{t}}$, i.e., $\mathcal{U}_t \subseteq \mathcal{F}_t$. Let $x \in \mathcal{U}_{\bar{t}}$, then:

$$|\phi^T x| \leq 2^{-\bar{t}} \leq 2^{-\log_2 1/C_{\min}^+} = C_{\min}^+$$

which is a contradiction because otherwise x would have a smaller constraint value than C_{\min}^+ . Consequently, the set of uncertain arms $\mathcal{U}_{\bar{t}}$ is empty and the algorithm returns the correct solution given $\mathcal{E}_{\bar{t}}$. Lemma D.1.1 shows that the unconditional probability of the algorithm returning the correct solution after round \bar{t} is at least $1 - \delta$.

Finally, we can compute the total number of samples the algorithm needs to return the correct solution:

$$\begin{aligned}
N &= \sum_{t=1}^{\bar{t}} \lceil 2^{2t+3} \log \left(\frac{|\mathcal{X}|}{\delta_t} \right) (1 + \varepsilon) \rho_t^* \rceil \\
&\leq \sum_{t=1}^{\bar{t}} 2^{2t+3} \log \left(\frac{|\mathcal{X}|}{\delta_t} \right) (1 + \varepsilon) \rho_t^* + \bar{t} \\
&\leq 8 \log \left(\frac{|\mathcal{X}| \bar{t}^2}{\delta^2} \right) (1 + \varepsilon) \sum_{t=1}^{\bar{t}} (2^t)^2 \rho_t^* + \bar{t} \\
&= 8 \log \left(\frac{|\mathcal{X}| \bar{t}^2}{\delta^2} \right) (1 + \varepsilon) \sum_{t=1}^{\bar{t}} (2^t)^2 \min_{\lambda} \max_{\tilde{x} \in \mathcal{U}_t} \|\tilde{x}\|_{A_{\lambda}^{-1}}^2 + \bar{t} \\
&= 8 \log \left(\frac{|\mathcal{X}| \bar{t}^2}{\delta^2} \right) (1 + \varepsilon) \sum_{t=1}^{\bar{t}} \min_{\lambda} \max_{\tilde{x} \in \mathcal{U}_t} \frac{\|\tilde{x}\|_{A_{\lambda}^{-1}}^2}{(2^{-t})^2} + \bar{t} \\
&\stackrel{(a)}{\leq} 8 \log \left(\frac{|\mathcal{X}| \bar{t}^2}{\delta^2} \right) (1 + \varepsilon) \sum_{t=1}^{\bar{t}} \min_{\lambda} \max_{\tilde{x} \in \mathcal{U}_t} \frac{\|\tilde{x}\|_{A_{\lambda}^{-1}}^2}{(\phi^T \tilde{x})^2} + \bar{t} \\
&\stackrel{(b)}{\leq} 8 \log \left(\frac{|\mathcal{X}| \bar{t}^2}{\delta^2} \right) (1 + \varepsilon) \sum_{t=1}^{\bar{t}} \min_{\lambda} \max_{\tilde{x} \in \mathcal{X}} \frac{\|\tilde{x}\|_{A_{\lambda}^{-1}}^2}{(\phi^T \tilde{x})^2} + \bar{t} \\
&\leq 8 \log \left(\frac{|\mathcal{X}| \bar{t}^2}{\delta^2} \right) (1 + \varepsilon) \bar{t} \bar{H}_{\text{CLB}}(\nu) + \bar{t}
\end{aligned}$$

where (a) follows because we showed that $|\phi^T x| \leq 2^{-t}$ w.h.p. for $x \in \mathcal{U}_t$, and (b) follows simply because $\mathcal{U}_t \subseteq \mathcal{X}$. In the last step, we

defined $\bar{H}_{\text{CLB}}(\nu) = \min_{\lambda} \max_{\tilde{x} \in \mathcal{X}} \frac{\|\tilde{x}\|_{A_{\lambda}^{-1}}^2}{(\phi^T \tilde{x})^2}$

Moreover,

$$\begin{aligned}
\bar{H}_{\text{CLB}}(\nu) &= \min_{\lambda} \max_{x \in \mathcal{X}} \frac{\|x\|_{A_{\lambda}^{-1}}^2}{(\phi^T x)^2} \leq \frac{1}{C_{\min}^+} \min_{\lambda} \max_{x \in \mathcal{X}} \|x\|_{A_{\lambda}^{-1}}^2 \\
&\leq \frac{1}{C_{\min}^+} \min_{\lambda} \max_{x \in \mathbb{R}^d} \|x\|_{A_{\lambda}^{-1}}^2 \leq \frac{d}{C_{\min}^+}
\end{aligned}$$

using the result by Kiefer and Wolfowitz [148].

□

Lemma D.1.1. Let $\mathcal{E}_1, \dots, \mathcal{E}_T$ be a Markovian sequence of events such that $P(\mathcal{E}_1) \geq 1 - \delta_1$ and $P(\mathcal{E}_t | \mathcal{E}_{t-1}) \geq 1 - \delta_t$ for all $t = 2, \dots, T$, where $\delta_t = \delta^2/t^2$ and $\delta \in (0, 1)$. \mathcal{E}_t is independent of other events conditioned on \mathcal{E}_{t-1} . Then $P(\mathcal{E}_T) \geq 1 - \delta$.

Proof.

$$\begin{aligned} P(\mathcal{E}_T) &= \left(\prod_{t=2}^T P(\mathcal{E}_t | \mathcal{E}_{t-1}) \right) P(\mathcal{E}_1) \geq \left(\prod_{t=2}^T (1 - \delta_t) \right) (1 - \delta_1) \\ &\geq \prod_{t=1}^{\infty} \left(1 - \frac{\delta^2}{t^2} \right) = \frac{\sin(\pi\delta)}{\pi\delta} \geq 1 - \delta \end{aligned}$$

where the last inequality holds for $0 \leq \delta \leq 1$. \square

D.1.3 Oracle and G-Allocation

Given any static design λ^* , we can consider different round-based algorithms using the static confidence intervals from Proposition 6.2.2. Algorithm 11 shows the general algorithm. It uses the same stopping condition as ACOL but uses a more straightforward round length of $v^t \log(|\mathcal{X}|/\delta_t)$ with v a hyperparameter, and a fixed static allocation. We analyze two versions of this generic algorithm that are of particular interest: the *oracle* solution and *G-Allocation*.

Theorem 6.2.2 (Oracle sample complexity). The oracle algorithm finds the optimal solution to a constrained linear best-arm identification problem $\nu = (\mathcal{X}, \theta, \phi)$ within $N \propto H_{\text{CLB}}(\nu)$ with probability at least $1 - \delta$.

Proof. Assuming a $(1 + \varepsilon)$ -approximate rounding procedure, in round t we have: $\|x\|_{A_{xN_t}^{-1}}^2 \leq \frac{1+\varepsilon}{N_t} \|x\|_{A_{\lambda^*}^{-1}}^2$. It follows, similar to the proof of

Theorem 6.2.4, that in round t , for each $x \in \mathcal{X}_\theta^{\geq}$ if $\phi^T x > \phi^T x_\nu^*$:

$$\begin{aligned} \phi^T x - l_\phi^t(x) &\leq \sqrt{2 \log(|\mathcal{X}|/\delta_t)} \|x\|_{A_{xN_t}^{-1}} \\ &\leq \sqrt{2(1 + \varepsilon) \log(|\mathcal{X}|/\delta_t)/N_t} \|x\|_{A_{\lambda^*}^{-1}} \end{aligned}$$

Algorithm 11 Round based algorithm with a generic allocation λ^* with hyperparameter $v \in (1, 2)$. For $\lambda^* \in \operatorname{argmin}_{\lambda} \max_{x \in \mathcal{X}_{\theta}^{\geq}(x_v^*)} \|x\|_{A_{\lambda}^{-1}} / |\phi^T x|$ this algorithm becomes the oracle solution. For $\lambda^* \in \operatorname{argmin}_{\lambda} \max_{x \in \mathcal{X}} \|x\|_{A_{\lambda}^{-1}}$ it becomes *G-Allocation*.

Require: static design λ^* , significance δ

- 1: $\mathcal{U}_1 \leftarrow \mathcal{X}$ (uncertain arms)
 - 2: $S_1 \leftarrow \{\}$ (feasible arms)
 - 3: $t \leftarrow 1$ (round)
 - 4: **while** $\mathcal{U}_t \neq \{\}$ **do**
 - 5: $\delta_t \leftarrow \delta^2 / t^2$
 - 6: $N_t \leftarrow \lceil v^t \log(|\mathcal{X}| / \delta_t) \rceil$
 - 7: $\mathbf{x}_{N_t} \leftarrow \operatorname{Round}(\lambda^*, N_t)$
 - 8: Pull arms x_1, \dots, x_{N_t} and observe constraint values
 - 9: $t \leftarrow t + 1$
 - 10: Update $\hat{\phi}_t$ and A based on new data
 - 11: $l_{\phi}^t(x) \leftarrow \hat{\phi}_t^T x - \sqrt{\beta_t} \|x\|_{A^{-1}}$ for all arms $x \in \mathcal{X}$
 - 12: $u_{\phi}^t(x) \leftarrow \hat{\phi}_t^T x + \sqrt{\beta_t} \|x\|_{A^{-1}}$ for all arms $x \in \mathcal{X}$
 - 13: $S_t \leftarrow S_{t-1} \cup \{x | u_{\phi}^t(x) \leq 0\}$
 - 14: $\bar{r} \leftarrow \max_{x \in S_t} \theta^T x$
 - 15: $\mathcal{U}_t \leftarrow \mathcal{U}_{t-1} \setminus \{x | l_{\phi}^t(x) > 0\} \setminus \{x | u_{\phi}^t(x) \leq 0\} \setminus \{x | \theta^T x < \bar{r}\}$
 - 16: **return** $x^* \in \operatorname{argmax}_{x \in S_t} \theta^T x$
-

A similar argument gives for x_v^* :

$$\begin{aligned} u_{\phi}^t(x_v^*) - \phi^T x_v^* &\leq \sqrt{2 \log(|\mathcal{X}| / \delta_t) \|x_v^*\|_{A_{x_v^*}^{-1}}} \\ &\leq \sqrt{2(1 + \varepsilon) \log(|\mathcal{X}| / \delta_t) / N_t \|x_v^*\|_{A_{\lambda^*}^{-1}}} \end{aligned}$$

Let us call the event that these confidence bounds hold \mathcal{E}_t . We have $P(\mathcal{E}_t | \mathcal{E}_{t-1}) \geq 1 - \delta_t$. Now, consider round $\bar{t} = \lceil \log_v(2(1 + \varepsilon)H_{\text{CLB}}(\nu)) \rceil$

with length $N_{\bar{t}} = \lceil 2(1 + \varepsilon) \log(|\mathcal{X}|/\delta_{\bar{t}})H_{\text{CLB}}(v) \rceil$. For all $x \in \mathcal{X}_{\theta}^{\geq}$ if $\phi^T x > \phi^T x_v^*$:

$$\phi^T x - l_{\phi}^{\bar{t}}(x) \leq \sqrt{\frac{1}{H_{\text{CLB}}(v)}} \|x\|_{A_{\lambda^*}^{-1}} \leq \sqrt{\frac{(\phi^T x)^2}{\|x\|_{A_{\lambda^*}^{-1}}^2}} \|x\|_{A_{\lambda^*}^{-1}} \leq |\phi^T x|$$

Note that x is infeasible and $\phi^T x$, which implies $l_{\phi}^{\bar{t}}(x) \geq 0$ and in turn $x \notin \mathcal{U}_{\bar{t}}$. Similarly, $u_{\phi}^{\bar{t}}(x_v^*) - \phi^T x_v^* \leq |\phi^T x_v^*|$. x_v^* is feasible and $\phi^T x_v^* \leq 0$. Hence, $u_{\phi}^{\bar{t}}(x_v^*) \leq 0$ and $x_v^* \notin \mathcal{U}_{\bar{t}}$. This implies that $\mathcal{U}_{\bar{t}} = \{\}$ and, conditioned on $\mathcal{E}_{\bar{t}}$, the oracle algorithm solves the problem in round \bar{t} with probability 1. We can apply Lemma D.1.1 to conclude that, unconditionally, the algorithm solves the problem in round \bar{t} with a probability of at least $1 - \delta$.

Let us compute the total iterations necessary:

$$\begin{aligned} N &= \sum_{t=1}^{\bar{t}} \lceil 2(1 + \varepsilon) \log(|\mathcal{X}|/\delta_t)H_{\text{CLB}}(v) \rceil \\ &\leq \bar{t}(1 + 2(1 + \varepsilon) \log(|\mathcal{X}|\bar{t}^2/\delta^2)H_{\text{CLB}}(v)) \propto H_{\text{CLB}}(v) \end{aligned}$$

So, N is on order $H_{\text{CLB}}(v)$ except for logarithmic factors, concluding the proof. \square

Theorem 6.2.3 (G-Allocation sample complexity). G-Allocation finds the optimal arm within $N \propto d/C_{\min}^+{}^2$ iterations with probability at least $1 - \delta$.

Proof. As in the proof of Theorem 6.2.2, we have in round t , for each $x \in \mathcal{X}_{\theta}^{\geq}$ if $\phi^T x > \phi^T x_v^*$:

$$\begin{aligned} u_{\phi}^t(x) - l_{\phi}^t(x) &\leq 2\sqrt{2 \log(|\mathcal{X}|/\delta_t)} \|x\|_{A_{\lambda_t^*}^{-1}} \\ &\leq 2\sqrt{2(1 + \varepsilon) \log(|\mathcal{X}|/\delta_t)/N_t} \|x\|_{A_{\lambda^*}^{-1}} \end{aligned}$$

Again, we call the event that these confidence bounds hold \mathcal{E}_t , and have $P(\mathcal{E}_t | \mathcal{E}_{t-1}) \geq 1 - \delta_t$. Now, consider round

$$\bar{t} = \log_v \left(8(1 + \varepsilon) \operatorname{argmin}_{\lambda} \max_{x \in \mathcal{X}} \|x\|_{A_{\lambda}^{-1}} / C_{\min}^{+2} \right)$$

$$N_{\bar{t}} = \left\lceil 8(1 + \varepsilon) \log(|\mathcal{X}| / \delta_{\bar{t}}) \operatorname{argmin}_{\lambda} \max_{x \in \mathcal{X}} \|x\|_{A_{\lambda}^{-1}} / C_{\min}^{+2} \right\rceil$$

For all $x \in \mathcal{U}_{\bar{t}}$ it follows that:

$$u_{\phi}^{\bar{t}}(x) - l_{\phi}^{\bar{t}}(x) \leq \sqrt{\frac{C_{\min}^{+2}}{\|x\|_{A_{\lambda^*}^{-1}}} \|x\|_{A_{\lambda^*}^{-1}}} \leq C_{\min}^{+}$$

This implies that G-Allocation solves the problem in round \bar{t} with probability 1, similar to the proof of Theorem 6.2.4. We can apply Lemma D.1.1 to conclude that, unconditionally, the algorithm solves the problem in round \bar{t} with a probability of at least $1 - \delta$.

Let us compute the total iterations necessary:

$$N = \sum_{t=1}^{\bar{t}} \left\lceil 8(1 + \varepsilon) \log(|\mathcal{X}| / \delta_t) \operatorname{argmin}_{\lambda} \max_{x \in \mathcal{X}} \|x\|_{A_{\lambda}^{-1}} / C_{\min}^{+2} \right\rceil$$

$$\leq \bar{t} \left(1 + 8(1 + \varepsilon) \log(|\mathcal{X}| / \delta_{\bar{t}}) \operatorname{argmin}_{\lambda} \max_{x \in \mathcal{X}} \|x\|_{A_{\lambda}^{-1}} / C_{\min}^{+2} \right)$$

$$\leq \bar{t} \left(1 + 8(1 + \varepsilon) \log(|\mathcal{X}| / \delta_{\bar{t}}) d / C_{\min}^{+2} \right) \propto d / C_{\min}^{+2}$$

where the last inequality uses the result by Kiefer and Wolfowitz [148]. \square

D.2 EXPERIMENT DETAILS

D.2.1 Driving Environment

The driving environment we use in Section 6.3.3 is the same as described in Appendix A.3.1; however, here we have rewards *and* constraints that are linear in a set of features

$$f(s) = (f_1(s), f_2(s), f_3(s), f_4(s), f_5(s), f_6(s), f_7(s), f_8(s), 1)$$

that are described in detail in Table D.1.

In the experiments from Section 6.3.3, we use a fixed time horizon $T = 20$, and policies are represented simply as sequences of 20 actions because the environment is deterministic.

We find policies in the Driver environment with a given reward function using the constrained cross-entropy method, discussed in Appendix C.3.2 and shown in Algorithm 10.

BINARY FEEDBACK. So far, we considered numerical observations of the constraint value $\phi^T x + \eta$ where η is sub-Gaussian noise. In the driving environment, we (more realistic) binary observations in $\{-1, 1\}$.

If we assume that all true constraint values are in $[-1, 1]$, we can define the observation model $P(y = 1 | \phi, x) = (\phi^T x + 1)/2$. We can consider this as bounded, sub-Gaussian noise on the constraint value, and so all our analysis still applies.

SETUP. To translate learning the unknown constraint function in the Driver environment into a constrained linear best arm identification problem, we consider a set of pre-computed policies Π . This set of policies corresponds to the arms of a linear bandit problem, and both the return $G(\pi)$ of a policy and the constraint function $J(\pi)$ are linear in the expected feature counts of the policy: $G(\pi) = \mathbf{f}(\pi) \cdot \mathbf{r}$ and $J(\pi) = \mathbf{f}(\pi) \cdot \mathbf{c}$.

For binary observations, we normalize the features of all policies such that all constraint values are between -1 and 1 .

D.2.2 Additional Results

Here, we provide the additional results for the experiments in Chapter 6. Full results are shown in Figure D.1 for the bandit results and Figure D.2 for the driving scenario. Table D.2 contains an overview of all algorithms and baselines that we evaluated.

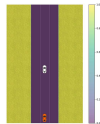
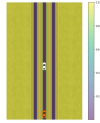
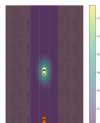
Feature	Description	Type	Definition	θ_1	θ_2	θ_3	ϕ	
$f_1(s)$	Target velocity	Numerical	$-(v - 0.4)^2$	1	0	1	0	
$f_2(s)$	Target location	Numerical	$-(x - x_r)^2$ x_r center of right lane	0	1	0	0	
$f_3(s)$	Stay on street	Binary	1 iff off street		0	0	0	0.3
$f_4(s)$	Stay in lane	Numeric	$\frac{1}{1 + \exp(-bd+a)}$ d distance to closest lane center, $b = 10000, a = 10$		0	0	0	0.05
$f_5(s)$	Stay aligned with street	Numeric	$ \cos(\theta) $	0	0	0	0.02	
$f_6(s)$	Don't drive backwards	Binary	1 iff $v < 0$	0	0	0	0.5	
$f_7(s)$	Stay within speed limit	Binary	1 iff $v > 0.6$	0	0	0	0.3	
$f_8(s)$	Don't get too close to other cars	Numeric	$\exp(-b(c_1d_x^2 + c_2d_y^2) + ba),$ $a = 0.01, b = 30,$ $c_1 = 4, c_2 = 1$		0	0	0	0.8

TABLE D.1: Features for representing the reward and constraint function in the Driver environment. The last four columns contain the reward weights for the three scenarios and the shared constraint weight.















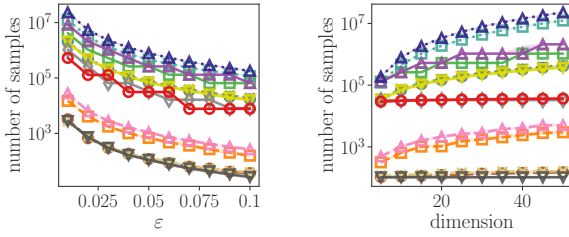
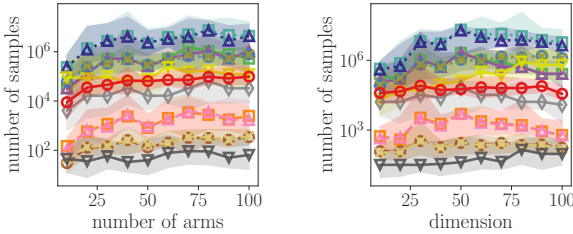
Name	Confidence Intervals	Selection Criterion	Select From Arms	Plot Color
Oracle	static	Oracle	All	
G-Allocation	static	MaxVar	All	
Uniform	static	Uniform	All	
ACOL	static	MaxVar	Uncertain	
Greedy MaxVar	adaptive	MaxVar	All	
Adaptive Uniform	adaptive	Uniform	All	
MaxRew- \mathcal{U}	adaptive	Max Rew	Uncertain	
MaxRew- \mathcal{S}	adaptive	Max Rew	Feasible	-
G-ACOL	adaptive	MaxVar	Uncertain	
G-ACOL Uniform	adaptive	Uniform	Uncertain	
Greedy MaxVar (tuned)	adaptive tuned	MaxVar	All	
Adaptive Uniform (tuned)	adaptive tuned	Uniform	All	
G-ACOL (tuned)	adaptive tuned	MaxVar	Uncertain	
G-ACOL Uniform (tuned)	adaptive tuned	Uniform	Uncertain	
MaxRew- \mathcal{U} (tuned)	adaptive tuned	Max Rew	Uncertain	
MaxRew- \mathcal{S} (tuned)	adaptive tuned	Max Rew	Feasible	-

TABLE D.2: Overview of all algorithms we evaluate.

We find that methods that select arms from \mathcal{U} randomly (G-ACOL Uniform) or by maximizing the reward (MaxRew- \mathcal{U}) can perform quite well in some cases with tuned confidence intervals. Indeed, MaxRew- \mathcal{U} outperforms G-ACOL in the unit sphere experiment. This is not consistent across environments, and G-ACOL performs comparable or better in all other environments. Still, in some cases, when theoretical guarantees are not required, these heuristic approaches might be valuable alternatives.



(a) Irrelevant dimensions



(b) Unit sphere

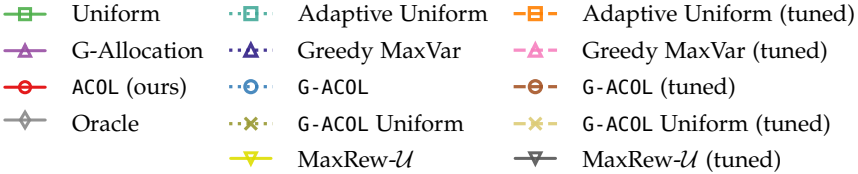
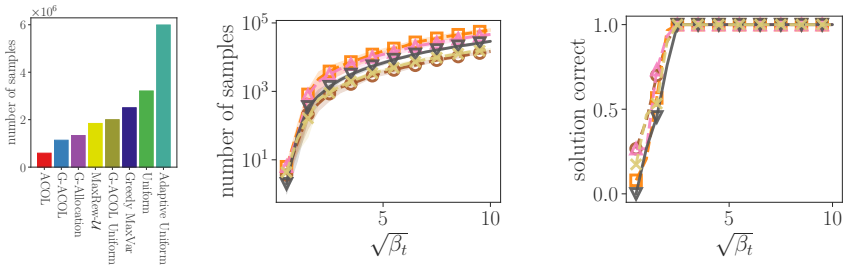
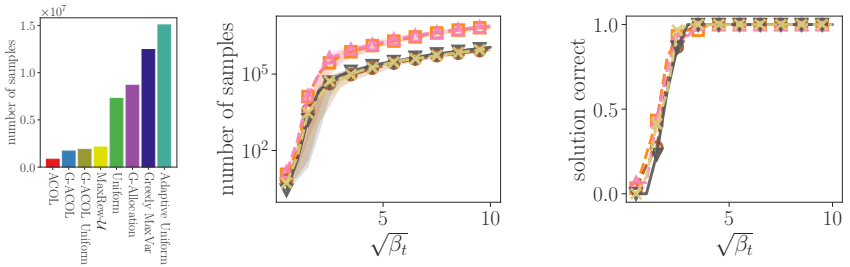


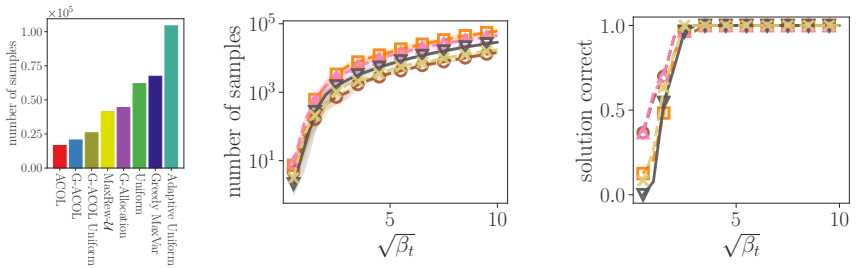
FIGURE D.1: Similar plots to Figure 6.2, including some additional algorithms: the “non-tuned” versions of algorithms use the confidence interval from Proposition 6.2.3, and *G-ACOL Uniform* is G-ACOL with uniform sampling instead of the maximum variance objective. Table D.2 provides an overview of all baselines. Moreover, the plots here show the 25th and 75th percentiles over 30 random seeds. For “irrelevant dimensions”, these are close to the median, but for “unit sphere”, there is much more randomness because the instances are randomly generated.



(a) "Base scenario"



(b) "Different reward"



(c) "Different environment"

-□- Adaptive Uniform
 -△- Greedy MaxVar
 -⊖- G-ACOL
-▽- MaxRew- \mathcal{U}
 -×- G-ACOL Uniform

FIGURE D.2: Similar plots as Figure 6.5 for all three driving scenarios from Figure 6.1, showing G-ACOL Uniform as an additional baseline.

BIBLIOGRAPHY

- [1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2nd edition. MIT press, 2018.
- [2] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, “Mastering Atari, Go, Chess and Shogi by planning with a learned model”, *Nature*, vol. 588, no. 7839, 604, 2020.
- [3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search”, *Nature*, vol. 529, no. 7587, 484, 2016.
- [4] J. Perolat, B. De Vylder, D. Hennes, E. Tarassov, F. Strub, V. de Boer, P. Muller, J. T. Connor, N. Burch, T. Anthony, *et al.*, “Mastering the game of Stratego with model-free multiagent reinforcement learning”, *Science*, vol. 378, no. 6623, 990, 2022.
- [5] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, “Dota 2 with large scale deep reinforcement learning”, *arXiv preprint arXiv:1912.06680*, 2019.
- [6] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, “Grandmaster level in StarCraft II using multi-agent reinforcement learning”, *Nature*, vol. 575, no. 7782, 350, 2019.

- [7] C. Gamble and J. Gao, *Safety-first AI for autonomous data centre cooling and industrial control*, <https://www.deepmind.com/blog/safety-first-ai-for-autonomous-data-centre-cooling-and-industrial-control>, 2018.
- [8] 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.
- [9] D. J. Mankowitz, A. Michi, A. Zhernov, M. Gelmi, M. Selvi, C. Paduraru, E. Leurent, S. Iqbal, J.-B. Lespiau, A. Ahern, *et al.*, “Faster sorting algorithms discovered using deep reinforcement learning”, *Nature*, vol. 618, no. 7964, 257, 2023.
- [10] V. Krakovna, J. Uesato, V. Mikulik, M. Rahtz, T. Everitt, R. Kumar, Z. Kenton, J. Leike, and S. Legg, *Specification gaming: The flip side of AI ingenuity*, <https://www.deepmind.com/blog/specification-gaming-the-flip-side-of-ai-ingenuity>, 2020.
- [11] W. B. Knox, A. Allievi, H. Banzhaf, F. Schmitt, and P. Stone, “Reward (mis) design for autonomous driving”, *Artificial Intelligence*, vol. 316, 103829, 2023.
- [12] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, “Deep reinforcement learning from human preferences”, in *Advances in Neural Information Processing Systems*, 2017.
- [13] N. Stiennon, L. Ouyang, J. Wu, D. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano, “Learning to summarize with human feedback”, in *Advances in Neural Information Processing Systems*, 2020.
- [14] 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”, in *Advances in Neural Information Processing Systems*, 2022.

- [15] A. Glaese, N. McAleese, M. Trębacz, J. Aslanides, V. Firoiu, T. Ewalds, M. Rauh, L. Weidinger, M. Chadwick, P. Thacker, *et al.*, “Improving alignment of dialogue agents via targeted human judgements”, *arXiv preprint arXiv:2209.14375*, 2022.
- [16] S. Casper, X. Davies, C. Shi, T. K. Gilbert, J. Scheurer, J. Rando, R. Freedman, T. Korbak, D. Lindner, P. Freire, T. Wang, S. Marks, C.-R. Segerie, M. Carroll, A. Peng, P. Christoffersen, M. Damani, S. Slocum, U. Anwar, A. Siththaranjan, M. Nadeau, E. J. Michaud, J. Pfau, D. Krasheninnikov, X. Chen, L. Langosco, P. Hase, E. Bıyık, A. Dragan, D. Krueger, D. Sadigh, and D. Hadfield-Menell, “Open problems and fundamental limitations of reinforcement learning from human feedback”, *arXiv preprint arXiv:2307.15217*, 2023.
- [17] K. Lee, L. Smith, A. Dragan, and P. Abbeel, “B-pref: Benchmarking preference-based reinforcement learning”, in *Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.
- [18] D. Lindner and M. El-Assady, “Humans are not Boltzmann distributions: Challenges and opportunities for modelling human feedback and interaction in reinforcement learning”, in *Communication in Human-AI Interaction Workshop (CHAI) at IJCAI-ECAI*, 2022.
- [19] F. Sperrle, M. El-Assady, G. Guo, R. Borgo, D. H. Chau, A. Ender, and D. Keim, “A survey of human-centered evaluations in human-centered machine learning”, in *Computer Graphics Forum*, Wiley Online Library, vol. 40, 2021, 543.
- [20] F. Y. Kung and A. A. Scholer, “The pursuit of multiple goals”, *Social and Personality Psychology Compass*, vol. 14, no. 1, 2020.
- [21] B. Settles, *Active Learning*. Morgan & Claypool Publishers, 2012.
- [22] D. Sadigh, A. D. Dragan, S. Sastry, and S. A. Seshia, “Active preference-based learning of reward functions”, in *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

- [23] S. Bubeck, R. Munos, and G. Stoltz, "Pure exploration in multi-armed bandits problems", in *International Conference on Algorithmic Learning Theory*, 2009.
- [24] T. Lattimore and C. Szepesvári, *Bandit algorithms*. Cambridge University Press, 2020.
- [25] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem", *SIAM journal on computing*, vol. 32, no. 1, 48, 2002.
- [26] W. Chu, L. Li, L. Reyzin, and R. Schapire, "Contextual bandits with linear payoff functions", in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.
- [27] S. Agrawal and N. Goyal, "Near-optimal regret bounds for Thompson sampling", *Journal of the ACM (JACM)*, vol. 64, no. 5, 1, 2017.
- [28] D. Bouneffouf, I. Rish, and C. Aggarwal, "Survey on applications of multi-armed and contextual bandits", in *IEEE Congress on Evolutionary Computation (CEC)*, 2020, 1.
- [29] R. Garnett, *Bayesian Optimization*. Cambridge University Press, 2023.
- [30] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
- [31] J. Mockus, V. Tiesis, and A. Zilinskas, "The application of Bayesian methods for seeking the extremum", *Towards Global Optimization*, vol. 2, 1978.
- [32] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design", in *Proceedings of International Conference on Machine Learning (ICML)*, 2010.
- [33] M. L. Puterman, *Markov decision processes: Discrete stochastic dynamic programming*, 2nd edition. John Wiley & Sons, 2014.
- [34] D. Bertsekas, *Dynamic programming and optimal control: Volume I*, 4th edition. Athena scientific, 2017.

- [35] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation", *Advances in neural information processing systems*, vol. 12, 1999.
- [36] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.*, "An introduction to deep reinforcement learning", *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, 219, 2018.
- [37] E. Altman, *Constrained Markov decision processes*. CRC press, 1999, vol. 7.
- [38] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*. Athena scientific Belmont, MA, 1997.
- [39] C. J. Watkins and P. Dayan, "Q-learning", *Machine learning*, vol. 8, 279, 1992.
- [40] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, 2015.
- [41] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization", in *Proceedings of International Conference on Machine Learning (ICML)*, 2015.
- [42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms", *arXiv preprint arXiv:1707.06347*, 2017.
- [43] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization", in *Proceedings of International Conference on Machine Learning (ICML)*, 2017.
- [44] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, and S. Legg, "Scalable agent alignment via reward modeling: A research direction", *arXiv preprint arXiv:1811.07871*, 2018.

- [45] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning", in *Proceedings of International Conference on Machine Learning (ICML)*, 2004.
- [46] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning.", in *Proceedings of International Conference on Machine Learning (ICML)*, 2000.
- [47] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, "Maximum entropy inverse reinforcement learning.", in *AAAI Conference on Artificial Intelligence*, 2008.
- [48] A. Gleave and S. Toyer, "A primer on maximum causal entropy inverse reinforcement learning", *arXiv preprint arXiv:2203.11409*, 2022.
- [49] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: The TAMER framework", in *International Conference on Knowledge Capture*, 2009.
- [50] J. MacGlashan, M. K. Ho, R. Loftin, B. Peng, G. Wang, D. L. Roberts, M. E. Taylor, and M. L. Littman, "Interactive learning from policy-dependent human feedback", in *Proceedings of International Conference on Machine Learning (ICML)*, 2017.
- [51] C. Arzate Cruz and T. Igarashi, "A survey on interactive reinforcement learning: Design principles and open challenges", in *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, ser. DIS '20, Eindhoven, Netherlands: Association for Computing Machinery, 2020, 1195.
- [52] C. Daniel, O. Kroemer, M. Viering, J. Metz, and J. Peters, "Active reward learning with a novel acquisition function", *Autonomous Robots*, vol. 39, no. 3, 389, 2015.
- [53] P.-H. Su, M. Gašić, N. Mrkšić, L. M. Rojas-Barahona, S. Ultes, D. Vandyke, T.-H. Wen, and S. Young, "On-line active reward learning for policy optimisation in spoken dialogue systems", in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin,

- Germany: Association for Computational Linguistics, 2016, 2431.
- [54] E. Hüllermeier, J. Fürnkranz, W. Cheng, and K. Brinker, “Label ranking by learning pairwise preferences”, *Artificial Intelligence*, vol. 172, no. 16, 1897, 2008.
- [55] C. Wirth, R. Akrouf, G. Neumann, and J. Fürnkranz, “A survey of preference-based reinforcement learning methods”, *The Journal of Machine Learning Research*, vol. 18, no. 1, 4945, 2017.
- [56] R. A. Bradley and M. E. Terry, “Rank analysis of incomplete block designs: I. The method of paired comparisons”, *Biometrika*, vol. 39, no. 3/4, 324, 1952.
- [57] J. Kulick, M. Toussaint, T. Lang, and M. Lopes, “Active learning for teaching a robot grounded relational symbols”, in *Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI)*, 2013.
- [58] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei, “Reward learning from human preferences and demonstrations in Atari”, in *Advances in Neural Information Processing Systems*, 2018.
- [59] E. Bıyık, M. Palan, N. C. Landolfi, D. P. Losey, and D. Sadigh, “Asking easy questions: A user-friendly approach to active reward learning”, in *Conference on Robot Learning (CoRL)*, 2020.
- [60] E. Bıyık, N. Huynh, M. J. Kochenderfer, and D. Sadigh, “Active preference-based Gaussian process regression for reward learning”, in *Proceedings of Robotics: Science and Systems (RSS)*, 2020.
- [61] N. Wilde, D. Kulic, and S. L. Smith, “Active preference learning using maximum regret”, in *International Conference on Intelligent Robots and Systems (IROS)*, 2020.

- [62] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples", *Biometrika*, vol. 25, no. 3/4, 285, 1933.
- [63] J. Kirschner, T. Lattimore, and A. Krause, "Information directed sampling for linear partial monitoring", in *Conference on Learning Theory*, 2020.
- [64] D. Russo and B. Van Roy, "Learning to optimize via information-directed sampling", in *Advances in Neural Information Processing Systems*, 2014.
- [65] A. Rustichini, "Minimizing regret: The general case", *Games and Economic Behavior*, vol. 29, no. 1-2, 224, 1999.
- [66] T. Fiez, L. Jain, K. G. Jamieson, and L. Ratliff, "Sequential experimental design for transductive linear bandits", in *Advances in Neural Information Processing Systems*, 2019.
- [67] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, "Scalable Bayesian optimization using deep neural networks", in *Proceedings of International Conference on Machine Learning (ICML)*, 2015.
- [68] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor", in *Proceedings of International Conference on Machine Learning (ICML)*, 2018.
- [69] 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*, IEEE, 2012, 5026.
- [70] H. Mania, A. Guy, and B. Recht, "Simple random search of static linear policies is competitive for reinforcement learning", in *Advances in Neural Information Processing Systems*, 2018.
- [71] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym", *arXiv preprint arXiv:1606.01540*, 2016.

- [72] A. M. Metelli, G. Ramponi, A. Concetti, and M. Restelli, "Provably efficient learning of transferable rewards", in *Proceedings of International Conference on Machine Learning (ICML)*, 2021.
- [73] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning", in *Proceedings of International Conference on Machine Learning (ICML)*, 2006.
- [74] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning.", in *Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI)*, 2007.
- [75] S. Levine, Z. Popovic, and V. Koltun, "Nonlinear inverse reinforcement learning with Gaussian processes", *Advances in Neural Information Processing Systems*, 2011.
- [76] G. Dexter, K. Bello, and J. Honorio, "Inverse reinforcement learning in a continuous state space with formal guarantees", in *Advances in Neural Information Processing Systems*, 2021.
- [77] M. Lopes, F. Melo, and L. Montesano, "Active learning for reward estimation in inverse reinforcement learning", in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2009.
- [78] R. Cohn, E. Durfee, and S. Singh, "Comparing action-query strategies in semi-autonomous agents", in *AAAI Conference on Artificial Intelligence*, 2011.
- [79] D. S. Brown, Y. Cui, and S. Niekum, "Risk-aware active inverse reinforcement learning", in *Conference on Robot Learning (CoRL)*, 2018.
- [80] D. P. Losey and M. K. O'Malley, "Including uncertainty when learning from human corrections", in *Conference on Robot Learning (CoRL)*, 2018.
- [81] N. Rajaraman, L. Yang, J. Jiao, and K. Ramchandran, "Toward the fundamental limits of imitation learning", in *Advances in Neural Information Processing Systems*, 2020.

- [82] T. Xu, Z. Li, and Y. Yu, "Error bounds of imitating policies and environments", in *Advances in Neural Information Processing Systems*, 2020.
- [83] P. Abbeel and A. Y. Ng, "Exploration and apprenticeship learning in reinforcement learning", in *Proceedings of International Conference on Machine Learning (ICML)*, 2005.
- [84] L. Shani, T. Zahavy, and S. Mannor, "Online apprenticeship learning", in *AAAI Conference on Artificial Intelligence*, 2022.
- [85] E. Kaufmann, P. Ménard, O. D. Domingues, A. Jonsson, E. Leurent, and M. Valko, "Adaptive reward-free exploration", in *Algorithmic Learning Theory*, 2021.
- [86] P. Auer, T. Jaksch, and R. Ortner, "Near-optimal regret bounds for reinforcement learning", *Advances in Neural Information Processing Systems*, 2008.
- [87] A. Zanette, M. J. Kochenderfer, and E. Brunskill, "Almost horizon-free structure-aware best policy identification with a generative model", in *Advances in Neural Information Processing Systems*, 2019.
- [88] C. Jin, A. Krishnamurthy, M. Simchowitz, and T. Yu, "Reward-free exploration for reinforcement learning", in *Proceedings of International Conference on Machine Learning (ICML)*, 2020.
- [89] D. R. Scobee and S. S. Sastry, "Maximum likelihood constraint inference for inverse reinforcement learning", in *International Conference on Learning Representations (ICLR)*, 2020.
- [90] S. Malik, U. Anwar, A. Aghasi, and A. Ahmed, "Inverse constrained reinforcement learning", in *Proceedings of International Conference on Machine Learning (ICML)*, 2021.
- [91] J. Garcia and F. Fernández, "A comprehensive survey on safe reinforcement learning", *Journal of Machine Learning Research*, vol. 16, no. 1, 1437, 2015.

- [92] K. C. Stocking, D. L. McPherson, R. P. Matthew, and C. J. Tomlin, "Discretizing dynamics for maximum likelihood constraint inference", *arXiv preprint arXiv:2109.04874*, 2021.
- [93] 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.
- [94] D. L. McPherson, K. C. Stocking, and S. S. Sastry, "Maximum likelihood constraint inference from stochastic demonstrations", in *Conference on Control Technology and Applications (CCTA)*, 2021.
- [95] M. Baert, P. Mazzaglia, S. Leroux, and P. Simoens, "Maximum causal entropy inverse constrained reinforcement learning", *arXiv preprint arXiv:2305.02857*, 2023.
- [96] D. Papadimitriou, U. Anwar, and D. S. Brown, "Bayesian inverse constrained reinforcement learning", in *Transactions on Machine Learning Research (TMLR)*, 2022.
- [97] G. Chou, N. Ozay, and D. Berenson, "Learning constraints from locally-optimal demonstrations under cost function uncertainty", *IEEE Robotics and Automation Letters*, vol. 5, no. 2, 3682, 2020.
- [98] M. Babes, V. Marivate, K. Subramanian, and M. L. Littman, "Apprenticeship learning about multiple intentions", in *Proceedings of International Conference on Machine Learning (ICML)*, 2011.
- [99] J. Choi and K.-E. Kim, "Nonparametric Bayesian inverse reinforcement learning for multiple reward functions", in *Advances in Neural Information Processing Systems*, 2012.
- [100] C. Dimitrakakis and C. A. Rothkopf, "Bayesian multitask inverse reinforcement learning", in *Recent Advances in Reinforcement Learning: 9th European Workshop, EWRL 2011, Athens, Greece, September 9-11, 2011, Revised Selected Papers 9*, Springer, 2012, 273.

- [101] K. Xu, E. Ratner, A. Dragan, S. Levine, and C. Finn, "Learning a prior over intent via meta-inverse reinforcement learning", in *Proceedings of International Conference on Machine Learning (ICML)*, 2019.
- [102] L. Yu, T. Yu, C. Finn, and S. Ermon, "Meta-inverse reinforcement learning with probabilistic context variables", in *Advances in Neural Information Processing Systems*, 2019.
- [103] P. Wang, H. Li, and C.-Y. Chan, "Meta-adversarial inverse reinforcement learning for decision-making tasks", in *International Conference on Robotics and Automation (ICRA)*, 2021.
- [104] K. Rezaee and P. Yadmellat, "How to not drive: Learning driving constraints from demonstration", in *2022 IEEE Intelligent Vehicles Symposium (IV)*, 2022.
- [105] 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.
- [106] 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)*, 2018, 2118.
- [107] A. Gaurav, K. Rezaee, G. Liu, and P. Poupart, "Learning soft constraints from constrained expert demonstrations", in *International Conference on Learning Representations (ICLR)*, 2023.
- [108] R. Camilleri, A. Wagenmaker, J. Morgenstern, L. Jain, and K. Jamieson, "Active learning with safety constraints", in *Advances in Neural Information Processing Systems*, 2022.
- [109] S. S. Khan and M. G. Madden, "One-class classification: Taxonomy of study and review of techniques", *The Knowledge Engineering Review*, vol. 29, no. 3, 345, 2014.

- [110] A. Baddeley, I. Bárány, and R. Schneider, “Random polytopes, convex bodies, and approximation”, *Stochastic Geometry: Lectures given at the CIME Summer School held in Martina Franca, Italy, September 13–18, 2004*, 77, 2007.
- [111] L.-A. Gottlieb, E. Kaufman, A. Kontorovich, and G. Nivasch, “Learning convex polytopes with margin”, in *Advances in Neural Information Processing Systems*, 2018.
- [112] W. R. Pulleyblank, *Polyhedral combinatorics*. Springer, 1983.
- [113] P. McMullen, “The maximum numbers of faces of a convex polytope”, *Mathematika*, vol. 17, no. 2, 179, 1970.
- [114] J. Sember, “Guarantees concerning geometric objects with imprecise points”, Ph.D. dissertation, University of British Columbia, 2011.
- [115] A. Edalat, A. A. Khanban, and A. Lieutier, “Computability in computational geometry”, in *New Computational Paradigms: First Conference on Computability in Europe, CiE 2005, Amsterdam, The Netherlands, June 8-12, 2005. Proceedings 1*, Springer, 2005, 117.
- [116] A. Edalat, A. Lieutier, and E. Kashefi, “The convex hull in a new model of computation”, in *Proceedings of the 13th Canadian Conference on Computational Geometry, University of Waterloo, Ontario, Canada, August 13-15, 2001*, 2001, 93.
- [117] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, “The quickhull algorithm for convex hulls”, *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 4, 469, 1996.
- [118] E. Leurent, *An environment for autonomous driving decision-making*, <https://github.com/eleurent/highway-env>, 2018.
- [119] M. Wen and U. Topcu, “Constrained cross-entropy method for safe reinforcement learning”, *IEEE Transactions on Automatic Control*, 2020.

- [120] J.-Y. Audibert, S. Bubeck, and R. Munos, "Best arm identification in multi-armed bandits.", in *Conference on Learning Theory (COLT)*, 2010.
- [121] M. Soare, A. Lazaric, and R. Munos, "Best-arm identification in linear bandits", in *Advances in Neural Information Processing Systems*, 2014.
- [122] A. Pacchiano, M. Ghavamzadeh, P. Bartlett, and H. Jiang, "Stochastic bandits with linear constraints", in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.
- [123] A. Locatelli, M. Gutzeit, and A. Carpentier, "An optimal algorithm for the thresholding bandit problem", in *Proceedings of International Conference on Machine Learning (ICML)*, 2016.
- [124] A. Kazerouni, M. Ghavamzadeh, Y. Abbasi Yadkori, and B. Van Roy, "Conservative contextual linear bandits", in *Advances in Neural Information Processing Systems*, 2017.
- [125] H. Kano, J. Honda, K. Sakamaki, K. Matsuura, A. Nakamura, and M. Sugiyama, "Good arm identification via bandit feedback", *Machine Learning*, vol. 108, 2019.
- [126] K. Khezeli and E. Bitar, "Safe linear stochastic bandits", in *AAAI Conference on Artificial Intelligence*, 2020.
- [127] S. Amani, M. Alizadeh, and C. Thrampoulidis, "Linear stochastic bandits under safety constraints", in *Advances in Neural Information Processing Systems*, 2019.
- [128] A. Moradipari, S. Amani, M. Alizadeh, and C. Thrampoulidis, "Safe linear Thompson sampling with side information", *IEEE Transactions on Signal Processing*, 2021.
- [129] Z. Wang, A. Wagenmaker, and K. Jamieson, "Best arm identification with safety constraints", in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2022.

- [130] J. R. Gardner, M. J. Kusner, Z. E. Xu, K. Q. Weinberger, and J. P. Cunningham, "Bayesian optimization with inequality constraints", in *Proceedings of International Conference on Machine Learning (ICML)*, 2014.
- [131] J. M. Hernández-Lobato, M. A. Gelbart, R. P. Adams, M. W. Hoffman, and Z. Ghahramani, "A general framework for constrained Bayesian optimization using information-based search", *Journal of Machine Learning Research*, vol. 17, 2016.
- [132] V. Perrone, I. Shcherbatyi, R. Jenatton, C. Archambeau, and M. Seeger, "Constrained Bayesian optimization with max-value entropy search", in *NeurIPS 2019 Workshop on Metalearning*, 2019.
- [133] Y. Sui, A. Gotovos, J. Burdick, and A. Krause, "Safe exploration for optimization with Gaussian processes", in *Proceedings of International Conference on Machine Learning (ICML)*, 2015.
- [134] M. Soare, "Sequential resource allocation in linear stochastic bandits", Ph.D. dissertation, Université Lille 1-Sciences et Technologies, 2015.
- [135] Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári, "Improved algorithms for linear stochastic bandits", in *Advances in Neural Information Processing Systems*, 2011.
- [136] P. Friedrich, *Optimal design of experiments*. Siam. Society for industrial and applied mathematics, 2006.
- [137] A. Gleave and G. Irving, "Uncertainty estimation for language reward models", *arXiv preprint arXiv:2203.07472*, 2022.
- [138] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.
- [139] L. Wasserman, *All of statistics: A concise course in statistical inference*. Springer Science & Business Media, 2004.
- [140] T. M. Cover and J. A. Thomas, *Elements of information theory*, 2nd edition. John Wiley & Sons, 2006.

- [141] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, *Stable baselines3*, <https://github.com/DLR-RM/stable-baselines3>, 2019.
- [142] Y. Duan, X. Chen, R. Houthoofd, J. Schulman, and P. Abbeel, “Benchmarking deep reinforcement learning for continuous control”, in *Proceedings of International Conference on Machine Learning (ICML)*, 2016.
- [143] T. Jaksch, R. Ortner, and P. Auer, “Near-optimal regret bounds for reinforcement learning”, *Journal of Machine Learning Research*, vol. 11, 1563, 2010.
- [144] S. Kakade and J. Langford, “Approximately optimal approximate reinforcement learning”, in *Proceedings of International Conference on Machine Learning (ICML)*, 2002.
- [145] E. Leurent, Y. Blanco, D. Efimov, and O.-A. Maillard, “Approximate robust control of uncertain dynamical systems”, in *Advances in Neural Information Processing Systems*, 2018.
- [146] R. Y. Rubinstein and D. P. Kroese, *The cross-entropy method: A unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Springer, 2004, vol. 133.
- [147] E. Kaufmann, O. Cappé, and A. Garivier, “On the complexity of best-arm identification in multi-armed bandit models”, *The Journal of Machine Learning Research*, vol. 17, 2016.
- [148] J. Kiefer and J. Wolfowitz, “The equivalence of two extremum problems”, *Canadian Journal of Mathematics*, vol. 12, 1960.