2019

Reinforcement Learning With High-Level Task Specifications

Min Wen
University of Pennsylvania, kelanzhen@gmail.com

Reinforcement Learning With High-Level Task Specifications

Abstract

Reinforcement learning (RL) has been widely used, for example, in robotics, recommendation systems, and financial services. Existing RL algorithms typically optimize reward-based surrogates rather than the task performance itself. Therefore, they suffer from several shortcomings in providing guarantees for the task performance of the learned policies: An optimal policy for a surrogate objective may not have optimal task performance. A reward function that helps achieve satisfactory task performance in one environment may not transfer well to another environment. RL algorithms tackle nonlinear and nonconvex optimization problems and may, in general, not able to find globally optimal policies. The goal of this dissertation is to develop RL algorithms that explicitly account for formal high-level task specifications and equip the learned policies with provable guarantees for the satisfaction of these specifications. The resulting RL and inverse RL algorithms utilize multiple representations of task specifications, including conventional reward functions, expert demonstrations, temporal logic formulas, trajectory-based constraint functions as well as their combinations. These algorithms offer several promising capabilities. First, they automatically generate a memory transition system, which is critical for tasks that cannot be implemented by memoryless policies. Second, the formal specifications can act as reliable performance criteria for the learned policies despite the quality of the designed reward functions and variations in the underlying environments. Third, the algorithms enable online RL that never violates critical task and safety requirements, even during exploration.

Degree Type

Dissertation

Degree Name

Doctor of Philosophy (PhD)

Graduate Group

Electrical & Systems Engineering

First Advisor

Ufuk Topcu

Second Advisor

George J. Pappas

Keywords

Game theory, Inverse reinforcement learning, Learning-based control, Learning from demonstration, Reinforcement learning, Temporal logic specifications

Subject Categories

Artificial Intelligence and Robotics | Computer Sciences

REINFORCEMENT LEARNING WITH HIGH-LEVEL TASK SPECIFICATIONS

Min Wen

A DISSERTATION

in

Electrical and Systems Engineering

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2019

Supervisor of Dissertation

Ufuk Topcu, Assistant Professor of Aerospace Engineering and Engineering Mechanics, Univ. of Texas at Austin

Graduate Group Chairperson

Victor Preciado, Associate Professor of Electrical and Systems Engineering

Dissertation Committee

George J. Pappas, Joseph Moore Professor and Chair of Electrical and Systems Engineering

Manfred Morari, Distinguished Faculty Fellow of Electrical and Systems Engineering

Michael Littman, Professor and Associate Chair of Computer Science, Brown University

REINFORCEMENT LEARNING WITH HIGH-LEVEL TASK SPECIFICATIONS

To my family.

ACKNOWLEDGEMENT

First and foremost, I would like to express my deepest gratitude to my advisor, Ufuk Topcu. Without his continuous encouragement, trust, and patience throughout my studies, it would be impossible for me to explore the diverse research topics in my thesis and find my own research story there. His enthusiasm for research and working and his positive attitude in face of all difficulties have shaped my understanding of being a researcher and a reliable person. It is my fortune to have him as my advisor and friend.

I would like to thank my thesis committee members, Professor George Pappas, Manfred Morari, and Michael Littman, for generously sharing their vision and wisdom with me. I also learned a lot by attending George's group meetings in the past few years.

I thank my friends for our memorable times together: Ximing Chen, Bhoram Lee, Jinwook Huh, Sangdon Park, Meng Xu, Siyao Hu, Clark Zhang, Ivan Papusha, Shaoru Chen, Lingjun Chen, and Konstantinos Gatsis. I am also very lucky to have two of my high-school deskmates, Ning Wang and Meng Ye, to pursue our PhDs together in Philadelphia.

Finally, I would like to thank my parents, Dan Liu and Kewu Wen. Thank you for being my parents, raising me with all your love, and always supporting me to pursue my dreams. I also thank my grandfather, Taiji Liu, for sharing his own experience with me when I get puzzled and upset. I will always benefit from his positive attitude towards life. I thank my husband, Jihua Huang, for his love, companion, understanding, and support.

ABSTRACT

REINFORCEMENT LEARNING WITH HIGH-LEVEL TASK SPECIFICATIONS

Min Wen

Ufuk Topcu

Reinforcement learning (RL) has been widely used, for example, in robotics, recommendation systems, and financial services. Existing RL algorithms typically optimize reward-based surrogates rather than the task performance itself. Therefore, they suffer from several shortcomings in providing guarantees for the task performance of the learned policies: An optimal policy for a surrogate objective may not have optimal task performance. A reward function that helps achieve satisfactory task performance in one environment may not transfer well to another environment. RL algorithms tackle nonlinear and nonconvex optimization problems and may, in general, not able to find globally optimal policies. The goal of this dissertation is to develop RL algorithms that explicitly account for formal high-level task specifications and equip the learned policies with provable guarantees for the satisfaction of these specifications. The resulting RL and inverse RL algorithms utilize multiple representations of task specifications, including conventional reward functions, expert demonstrations, temporal logic formulas, trajectory-based constraint functions as well as their combinations. These algorithms offer several promising capabilities. First, they automatically generate a memory transition system, which is critical for tasks that cannot be implemented by memoryless policies. Second, the formal specifications can act as reliable performance criteria for the learned policies despite the quality of the designed reward functions and variations in the underlying environments. Third, the algorithms enable online RL that never violates critical task and safety requirements, even during exploration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

Chapter 1: Introduction

Reinforcement learning (RL) techniques have been used to solve a wide range of real-world tasks such as robotics [55, 68, 83, 85, 91, 196], recommendation systems [109, 172, 199] and financial services [45, 118, 128]. Given a reward function and some mechanism for a learning agent to interact with its environment, the goal of RL is to learn an *optimal* policy that maximizes the expected reward. To solve a robotic task using RL, one first needs to specify a reward function to encode the task objective, which implicitly introduces the following assumption:

The more expected *reward* a policy gets, the better *task performance* it has.

In other words, the assumption states that the expected reward is interpreted as a performance criterion for the given task. This assumption holds trivially for score-optimizing applications such as board games [100, 161, 162] and video games [122, 132], where a lot of impressive RL applications emerge these days. However, this assumption is rarely valid for most real-world tasks, where reward functions are not part of the original task specification and need to be designed. In general, reward design is non-trivial and mostly still an open problem [9, 69, 163, 200]. In many cases, reward design proceeds by trial and error with intense human supervision, yet the learned policies still lack guarantees for task performance. In Section 1.1, we show several reasons why a reward function by itself is not an ideal way to represent task requirements in general.

1.1. Challenges in Representing Task Requirements as Reward Functions

Conceptually, there is a non-negligible gap between the objective of reward design and the requirement of reward utilization. For reward design, the goal is usually to find a candidate reward function such that there *exists* an optimal policy that implements the given task successfully in *training* environments, as in the cases of ad-hoc reward design and inverse reinforcement learning [1]. For reward utilization, it is required that *all* optimal policies with the specified reward function can implement the task successfully in (possibly different) *testing* environments, as any optimal policy will be a sound output of an RL algorithm. An extreme case of this gap is an *all-zero* reward function,

i.e., a reward function that assigns zero rewards to all state-action pairs. A policy that implements the given task successfully is undoubtedly optimal for the all-zero reward function; nevertheless, it is impractical to learn such a policy using such an uninformative reward.

For some relatively simple tasks, it is possible to bridge this gap using *sparse* rewards. Consider a target-reaching task as an example, where the goal is to reach a given target object within ten time steps. Then a reward function that satisfies the above assumption can be designed as follows: The reward is one if the hand has just reached the target object and zero otherwise. With this reward function, any optimal policy that maximizes the expected total reward maximizes the probability to reach the target object within ten steps. However, the sparsity of non-zero reward signals makes exploration very inefficient and thus is undesired for most RL algorithms.

For more complicated tasks, reward functions need to encode multiple (possibly conflicting) long-term objectives, where each objective corresponds to some behavior to be encouraged or prohibited. The objectives can still be quantified separately as reward basis functions, but their weights that signify the importance of each basis function to task performance are not apparent. Even for a specific weight vector, it is not clear how to predict the task performance of the corresponding optimal policies without actually solving the RL problem. Moreover, the trade-off between the basis functions is also affected by the system dynamics and the underlying environment. Therefore, a reward function that leads to good task performance in one environment may not guarantee similar performance in another environment, which suggests that it may not be enough to represent task requirements merely as reward functions.

Last but not least, RL techniques may not always converge to a global optimum, especially when nonlinear parameterization is used, which is almost always the case for problems with continuous state spaces. Recently, deep RL techniques [108] becomes popular, where the agent's policy is parameterized as a neural network. The network weights are computed using backpropagation and updated using some stochastic gradient descent method. There is no wonder that neither the expected reward nor the task performance is convex in network weights, and thus gradient-based methods cannot guarantee convergence to global optima. Researchers have observed that implementation

details of deep RL algorithms such as policy network structure, batch size, learning rate, and even random seeds can drastically alter the performance of deep RL algorithms [71]. This observation again indicates that it is impractical to rely on reward design to guarantee the task performance of the learned policies.

Having discussed the limitations of reward-based task specifications, we also note that reward function is neither the unique way nor the most common way for humans to describe task requirements. Rather than optimizing some quantitative evaluative feedback (such as rewards) from each transition, it is far more natural to learn from high-level task requirements in natural language or to directly imitate some successful demonstrations of the given task. For example, when people learn to drive, they are required to learn the road rules to follow some general guidelines and avoid some common mistakes. In the meantime, they learn about driving customs by both mimicking the other drivers and observing others' feedback. Similarly, we can express the rule-based task requirements as temporal-logic-based specifications or finite-state transition systems, and consider expert demonstrations as a given set of trajectories.

Instead of relying on reward functions as a unique task description, we propose to incorporate miscellaneous sources of task information into RL algorithms, with the following objective in mind:

To learn policies that are *known* to have reliable task performance.

The exploitation of reward-free task information can help improve output task performance in multiple ways. For example, constraints help restrict the search space of output policies: Trajectories generated by a valid output policy should satisfy the given constraints with high probability. Temporal logic specifications show insights into task structure and facilitate the design of memory states. Expert demonstrations provide samples of desired behaviors for policy training and the inference of the expert's preferences. The works in this dissertation are some exploratory steps towards the ultimate goal of learning with reliable task performance.

1.2. Outline and Contributions

The overarching goal of this dissertation is to utilize different sources of available task information besides reward functions and provide the policies learned by reinforcement learning with task performance guarantees. The works in this dissertation contain three parts, where each part uses different formulations and task representations.

1.2.1. Learning from Demonstrations with High-Level Side Information

The first part includes Chapter 2 and Chapter 3, in which we represent the task information in the following two ways: First, there is a co-safe linear temporal logic specification or an equivalent deterministic finite automaton [99], which can be used to directly check whether a given trajectory fails or succeeds in the given task. Additionally, there is a set of demonstrations showing how an experienced expert implements the given task. The overall problem, formulated as an inverse reinforcement learning problem, aims at learning a reward function and an output policy at the same time such that the learned policy satisfies the given temporal logic specification with high probability.

Chapter 2 shows how to extend an existing inverse reinforcement learning algorithm (the example used in Chapter 2 is the maximum likelihood inverse reinforcement learning algorithm [12]) to explicitly take advantage of the extra input of task information as a co-safe linear temporal logic specification. The proposed algorithm incorporates the task information in several steps. First, it transforms the specification into an equivalent deterministic finite automaton, such that a trajectory satisfies the specification if and only if it is accepted by the deterministic finite automaton. Then the algorithm constructs a product automaton by extending the state space of the original environment with the deterministic finite automaton. Essentially, the deterministic finite automaton acts as a memory transition system that tracks the current progress of the task, since the policies of the product automaton depend on both the current state in the environment Markov decision process and the state in the task deterministic finite automaton. Moreover, there is a one-to-one mapping between the trajectories of the environment Markov decision process and the trajectories of the product automaton. Therefore, the task performance of the learned policy, i.e., the probability that the

learned policy satisfies the temporal logic specification, can be evaluated in the product automaton and is independent of the recovered reward function. As the task performance is differentiable, it can be used to augment the objective function of the underlying inverse reinforcement learning algorithm. The resulting algorithm parameterizes the reward function as a linear combination of a set of manually designed reward features in the constructed product automaton.

Chapter 3 extends the above framework to nonlinearly parameterized reward functions such as reward networks. The proposed algorithm relies on an existing deep maximum-entropy inverse reinforcement learning algorithm [195]. While the previous work focused on the generalization performance at states without expert demonstrations in the same environment, this work evaluates the generalization performance in new testing environments with no expert demonstrations. Compared with linearly parameterized rewards, reward networks gain the capability to construct reward features automatically and thus transferable to new environments. After transferring the learned reward function, the algorithm computes a new policy separately for each testing environment. A comparison of the generalization performance of the proposed algorithm with that of a memory-based behavioral cloning algorithm shows that, with the same set of expert demonstrations, policies generated by the learned reward function have near-perfect task performance in both training and testing environments, while the policy learned by the memory-based behavioral cloning algorithm deteriorates significantly in testing environments.

1.2.2. Constrained Reinforcement Learning

The second part corresponds to Chapter 4. In this part, we represent task information as objective and constraint functions and formulate the problem as a constrained reinforcement learning problem. The constraint functions act as the role of temporal logic specifications in the first part: a policy is *feasible* if and only if all constraint functions evaluated with this policy are within the specified ranges. The objective function encodes preferences over different feasible policies. Both objective and constraint functions are evaluated over finite-step trajectories and thus can encode even non-Markovian objectives, which is more general than standard reward functions.

We treat both objective and constraint functions as black boxes and propose a constrained cross-entropy-based method. The key idea is to transform the original constrained optimization problem into an unconstrained one with a surrogate objective. The method explicitly tracks its performance for constraint satisfaction and thus is well-suited for safety-critical applications. We show that the asymptotic behavior of the proposed algorithm converges almost-surely to that of an ordinary differential equation, and also provide sufficient conditions on the differential equation for the convergence of the proposed algorithm. We illustrate the performance of the proposed algorithm with two simulation examples. The first one is a constrained linear quadratic regulator problem, in which the algorithm converges to the global optimum with high probability. The second example is a 2D navigation problem, for which the proposed algorithm manages to learn feasible policies effectively without assumptions on the feasibility of initial policies, even with non-Markovian objective functions and constraint functions.

1.2.3. Reinforcement Learning with Temporal Logic Constraints in Two-Player Games

The third part includes Chapter 5 and 6. In this part, we consider two reinforcement learning problems in two-player games, where the task requirements are represented by one qualitative and one quantitative objective. Similar to the first part, we represent the qualitative objective as a temporal logic specification that is not limited to the co-safe ones. No matter which policy the other uncontrolled player takes, the learning agent should guarantee to satisfy the given temporal logic specification even during the learning procedure. The quantitative objective is the worst-case expected discounted reward, which is unknown a priori and learned by interacting with the other player. The two objectives are independent of each other such that they may be conflicting, somewhat similar, or irrelevant at all.

In Chapter 5, we model the interaction between a controlled agent (referred to as the *system* agent) and an uncontrolled agent (referred to as the *environment* agent) a deterministic two-player turn-based zero-sum game. Since the two objectives are independent, we decouple the two objectives and address them separately. In this work, the algorithm first computes, for the system agent, a (maximally) *permissive* policy from the given temporal logic specification. A permissive policy

includes multiple (possibly all) policies that guarantee the system to satisfy the given specification, despite the policy of the potentially adversarial environment agent. The algorithm restricts the system agent to take the policies included in the permissive strategy and solves an optimal system policy over the restricted set using any RL algorithms for zero-sum two-player games. For a particular case where the given temporal logic specification encodes a safety property, this two-step technique secures both correctness (with the safety property) and optimality (with the a priori unknown reward function). For other specifications, the learned policy still satisfies the given specification but may be sub-optimal.

Chapter 6 generalizes the previous problem in two ways. First, the game transitions can be stochastic instead of deterministic; second, it addresses a broader type of temporal logic specifications without loss of optimality. In this work, we assume that the given temporal logic specification is representable as a deterministic Büchi automaton, which strictly includes the safety property. The quantitative objective is to maximize the expected discounted reward over an infinite horizon. We prove that there always exists a *memoryless almost-sure winning* strategy that is ε-optimal for any arbitrary positive ε. Based on the idea of the R-MAX algorithm [28], a probably approximately correct (PAC) learning algorithm is proposed to learn such a strategy efficiently in an online manner with a priori unknown reward functions and unknown transition distributions. To the best of our knowledge, this is the first result on PAC learning in stochastic games with independent quantitative and qualitative objectives.

1.3. Related Work

In this section, we introduce two fields of works, namely learning from demonstrations and learning-based control with safety requirements, that are closely related to the high-level idea of this thesis. For each field, we describe the high-level ideas, main existing approaches, and their connections with high-level task specifications.

1.3.1. Learning from Demonstrations

Chapter 2 and 3 of this thesis is closely related to the topic of learning from demonstrations (LfD) [10], which studies the following problem: Given a set of expert demonstrations that are sampled from

some unknown expert policy, learn a policy to *imitate* the expert demonstrations. The interpretation of imitation varies among different LfD methods. In general, there are two types of approaches to LfD problems: behavioral cloning, and inverse reinforcement learning.

Behavioral cloning. Behavioral cloning methods directly formulate the LfD problem as a supervised learning problem. In other words, behavioral cloning directly reproduces a mapping from states to actions or action distributions that will be taken by the expert. Depending on whether actions are continuous or discrete, behavioral cloning methods can be classified into regression or classification methods. The strength of behavioral cloning has been demonstrated by various applications that range from basic navigation tasks such as lane keeping [142] to complicated tasks such as obstacle avoidance [78, 151] and end-to-end autonomous driving [25, 126].

Behavioral cloning methods suffer from the following two noticeable limitations.

The first limitation is on *compounding errors* or *cascading errors* [149]: The difference between the state distributions induced by the expert policy and the learned policy compounds over time. For behavioral cloning, testing performance is not evaluated over a randomly drawn subset of states from expert demonstrations (which is a common practice for standard supervised learning problems), but rather over the distribution of state-action pairs generated by executing the learned policy. As the action distributions at previous steps affect the state distributions at later steps, the state distribution induced by a learned policy gradually diverges from that of the expert policy as time goes on. As a result, the learned policy often guides the agent to reach states that are distinct or far away from the states in the expert demonstrations. It is not clear to the agent how to behave at these states, or how to return to states visited in demonstrations.

Another limitation of behavioral cloning is the inability to replan in new environments. For the policies trained by supervised learning methods, the predicted action distribution at each state is only decided by the local features of that state and independent of the possible states to be visited later. As a result, the learned policies ignore the long-term effect of each action and "(behavioral cloning) often leads to myopic and poor-quality robot performance" [148].

Various attempts have been made to overcome the above two limitations. The key idea is to allow the learning agent to query the expert's policy at any given state. Based on this idea, Ross and Bagnell [149] first proposed the SMILe algorithm to stochastically combine the expert policy and the policies learned in each iteration, and gradually reduce the probability to query the expert over time. Ross et al. [150] proposed another algorithm called DAgger that augments the demonstration data in each iteration with the expert's new demonstrations on all states visited by the learned policy. Laskey et al. [101] forced the expert to demonstrate how to recover from errors by injecting noise into the expert's policy during the demonstrating process.

Inverse reinforcement learning. Inverse reinforcement learning (IRL) [129], also called inverse optimal control [84], solves the LfD problem in an indirect manner: It assumes that the expert policy optimizes the expected reward with some unknown reward function, and thus IRL methods aim at inferring a reward function from the given expert demonstrations. IRL is closely related to (forward) RL, which solves optimal policies for some given reward function. Many IRL algorithms need to solve an RL problem after each update of the learned reward function. IRL methods have been used in many applications such as flying helicopters [2, 3], navigation of mobile robots [96, 187, 195], goal inference [14], behavior modeling of pedestrians [52] and drivers [98], to name a few.

IRL problems suffer from the problem of *ill-posedness*: For example, the all-zero reward function admits any policy as its optimal policy and thus is a sound solution for any expert demonstrations. A given policy can be optimal simultaneously for many different reward functions, and the optimal policies for these reward functions are generally not equivalent. Therefore, it is upon each IRL algorithm to decide how to interpret the expert demonstrations with the expert's reward function or with the output policy. These algorithms typically resort to some common heuristics: First, the expected reward of the expert policy (estimated as the average value over expert demonstration) matches that of the learned policy [1, 201]. Second, the expected reward of the expert policy is higher than that of any other policies by a given margin [147]. Third, the output policy maximizes the likelihood or posterior probability of generating expert demonstrations [12, 105, 146, 195, 202].

Challenges of LfD for task implementation. Remember that our goal is to learn how to implement high-level tasks from expert demonstrations. However, most existing LfD algorithms merely rely on statistical analyses of expert demonstrations and thus have no explicit representation of tasks. We briefly explain several challenges faced by these algorithms that prevent them from achieving satisfactory task performance, especially at new states or new environments where there are no demonstrations.

First, there is a lack of reliable criteria for the task performance of the learned policies. For behavioral cloning methods, loss functions that quantify the error between the expert policy and the learned policy are not useful: The expert policy may not be accessible at the all the states that the learning agent visits while taking the learned policy. Additionally, low training loss for the demonstration data does not guarantee low testing loss due to compounding errors. For IRL, researchers have designed multiple performance criteria, which are functions of the ground-truth expert reward function or even the ground-truth expert policy. Examples include the L1 or L2 distance between the learned reward and the expert reward [146], the suboptimality of the learned policy with respect to the expert reward function [41, 105, 192], and the average KL divergence between the output policy and the expert policy [72]. However, it is not clear how to relate the losses mentioned above to the corresponding task performance. Furthermore, the expert reward function and expert policy may be unavailable during testing.

Second, for both behavioral cloning and IRL, policies are assumed to be mappings from each state to an action or an action distribution. In other words, these policies are *memoryless*. This primary setting may not suffice for task implementation. Indeed, a lot of everyday tasks can be decomposed into multiple subtasks and thus beyond the scope of memoryless policies. Such tasks can be as simple as dialing in a phone number: the next digit is not a function of the last dialed digit, but the number of digits that have been dialed so far. However, it is ambiguous how to generate a memory system for an unknown task from a finite set of expert demonstrations.

Third, the expert policy is dependent on not only the current state but also the whole environment. Even with the same high-level task, the expert needs to recompute its policy in different environments.

For behavioral cloning methods that do not allow replanning in new environments, it is improbable that a policy learned in one environment can be transferred successfully in a new environment without modification. For IRL algorithms, reward functions learned from expert demonstrations are "merely observations about what the designer actually wants" [69] in the demonstrated environments. However, environments such as Markov decision processes, a class of models commonly used in RL and later in this thesis, are not directly encoded as an input to reward functions or policies. As a result, it is not clear if the inputs of reward functions contain enough local environment information to allow successful transfers to new environments.

LfD with some task information. There have been several attempts to introduce tasks into LfD problems. With diverse problem formulations, existing works mainly focus on the first two challenges: to evaluate policies and to introduce memory states. There are two types of approaches, which introduce tasks in different ways. The first type introduces tasks by augmenting demonstrations. Lee et al. [102] add a Boolean tag to each demonstration trajectory as an indication of whether the trajectory is satisfactory or not. In some other works [30, 51], each demonstration trajectory augmented with a continuous score, which can further indicate an expert's preference over different demonstration trajectories. Instead of ratings, Pan and Shen [134] augment each demonstration trajectory with a subset of the visited states, highlighted as the subgoals to visit in order to implement some implicit high-level task. The second type of work partitions each demonstration trajectory into several segments, where each segment corresponds to a different policy. The number of segments in each demonstration is either given [87, 158] or inferred automatically from demonstrations [119, 131]. All the proposed algorithms of this type are behavioral cloning methods. Therefore, it is not clear how to utilize the learned knowledge in new environments.

1.3.2. Learning-Based Control with Safety Requirements

The topic of this thesis is also closely related to the field of learning-based control with safety requirements, including safe RL [62]. Depending on each specific work, the word *safety* may have drastically different meanings.

Interpretation of safety. In general, safety properties are of interest for both the learned policies and the policies taken during exploration.

For learned policies, some commonly used safety requirements include *constraint satisfaction* and *stability*. Given a cost function and a budget value, one may express safety constraints in many different ways: The expected total cost cannot exceed a given budget (expectation constraint) [4, 43]; the probability that the total cost exceeds budget is bounded (chance constraint) [42, 81]; or the expected total cost over the worst α-proportion of worst cases is bounded for a given constant $\alpha \in (0, 1)$ (conditional value-at-risk) [42]. Another commonly used safety requirement is (asymptotic) stability [5, 11], which is a fundamental objective in control theory. Intuitively, asymptotic stability indicates that there exists a set of initial states and a policy, such that taking the policy from any state in the initial state set, the agent will eventually return to the origin at some point, despite the existence of modeling errors and disturbances. Stability requirements may be used to represent tasks such as goal reaching and reference tracking.

Safety may also be essential during exploration, especially for applications that involve physical systems. There has been a significant amount of work on the topic of *safe exploration*, in which all policies that a learning agent implements during the training should be safe. Some examples are as follows: The learning agent can never visit a set of failure states during exploration, which coincides with the interpretation of safety properties in model checking [99], i.e., a trajectory is safe if and only if it does not have an unsafe prefix. Usually, the failure states can be identified immediately by observing a safety function [44, 176, 180] or state labeling function [7]. Another type of safety requirement is to maintain *returnability*, that is, the ability to return to currently known non-failure states after visiting a new state, which is critical for continual exploration. For example, Moldovan and Abbeel [124] require the exploration policies to preserve ergodicity with high probability. Turchetta et al. [176] and Wachi et al. [180] restrict exploration to the unknown states that are highly likely to be safe, can be reached from some known state in one step and only visit highly-likely-safe states before returning to a known safety state eventually. Berkenkamp et al. [18] enforce learning agents to stay within the region of attraction, which is decided by a given

Lyapunov function. Alshiekh et al. [7] pre-compute the winning region for learning agents and only allow a learning agent to take actions that are guaranteed to stay within the winning region. Safety may refer to other properties such as monotonic improvement of the updated policies [137] or learned policies always perform better than a given baseline [63].

Summary of existing approaches. One of the main difficulties in learning with safety requirements is the trade-off between prior knowledge and new exploration. Prior knowledge is reliable yet restrictive, while exploration is adaptive yet uncertain. Intuitively, the lack of necessary information may prevent learning agents from safe exploration. Suppose that there is an unknown safety function that identifies the safe states, and the safety values at different states are independent. Consequently, a learning agent has to risk taking unsafe states in order to explore the safety function. To bound the risk of violating safety requirements during exploration, one may need to introduce additional assumptions to infer the safety values of new states from previous observations. In the remaining part of this section, we briefly summarize some existing work based on the introduced prior knowledge and the roles of learning.

One commonly used framework for learning-based control is learning-based model predictive control (MPC), where the system model is deterministic with bounded modeling error. Without learning, MPC techniques construct an online controller with guaranteed recursive feasibility and stability for all models with the given error bound. However, the resulting controller may be overly conservative or even may not exist, if the model allows too much uncertainty. To address this limitation, researchers have investigated to incorporate learning modules that refine dynamics model and improve control objectives, while keeping the safety and stability guarantees with high confidence. For example, Aswani et al. [11] propose to maintain two models simultaneously, one is the given linear dynamics model and the other is a refined model learned from observed transitions. With the given model, they resort to the idea of tube MPC to guarantee stability, safety, and recursive feasibility. With the refined model, they make better predictions to the induced trajectory given a sequence of control inputs, which helps improve the objective value of the learned policy. Koller et al. [92] study a similar problem as in [11] but use learning differently. Instead of learning a deterministic dynamics

model, they approximate the unknown dynamics as a Gaussian process (GP) model and further use it to over-approximate the system trajectory as a sequence of set-valued confidence regions with high probability. Akametalu et al. [5] propose a reachability-based framework to guarantee closed-loop stability for control-affine systems with unknown bounded modeling error. With the given part of the dynamics model and an overestimation of modeling errors, they design a safety value function V whose nonnegative superlevel sets are all controlled invariant. Given a critical safety value V_L, an agent can take any valid action if V at the current state is much greater than V_L and is forced to switch to a conservative policy otherwise. They use online measurements to learn a GP-based modeling error function to help select V_L such that the resultant control invariant set is as large as possible.

Learning is also used to explore a priori unknown safety function, which is a usual practice for safe exploration. Since learning agents are not allowed to visit unsafe states, it is critical to reliably infer the safety value of unvisited states from previous observations and only explore new safe states. Moreover, the newly explored transition should not prevent the agent from eventually reaching the origin (stability) or returning to other explored states safely (returnability).

For example, Turchetta et al. [176] study a safe exploration problem for finite-step Markov decision processes with known deterministic transitions. The proposed solution depends on both prior knowledge, such as the transition function, and new information inferred from previous observations. They approximate the safety function with a GP to gain statistical confidence about the safety values of unvisited transitions and only explore the states that are safe with high probability. With the given transition function, they can further guarantee the returnability to previously visited states after a new state is explored: Among the new states that are highly likely to be safe, they only explore the states that can both reach and be reachable from known states. The setting in [179] is very similar to that in [176] except that the authors of [179] also explicitly optimize an expected discounted reward with an unknown reward function. In each iteration, the learning agent partitions the state space into three parts: safe states, unsafe states, and uncertain states. The agent is only allowed to take transitions that lead to safe states. Thus the learned policy is guaranteed to be safe with high

probability, although it may not be globally optimal before the learning agent fully explores the state space. Berkenkamp et al. [18] propose to learn system dynamics from measurements without ever leaving the region of attraction. Although the region of attraction is not known a priori, a Lyapunov function V is available such that any sublevel set of V is a subset of the region of attraction. They use GPs to estimate a confidence interval of V at the next state for each transition. They only allow taking transitions for which the V-value of the next state is upper bounded by the current threshold value, which guarantees stability.

Another way to handle learning-based control is to decouple the learning part from safety objectives. Prior knowledge is used to limit the policy search space for the learning agent, either by explicitly restricting the actions that the learning agent can take or by providing a set of candidate policies. All the policies that remain in the policy space satisfy the given safety objectives. Junges et al. [81] follow this idea and design a safe reinforcement learning algorithm to minimize the expected total cost (optimization objective) while satisfying an independently specified probabilistic reachability constraint (safety objective). In each iteration, they compute a *safe permissive policy*, which is a set of memoryless policies that satisfy the safety objective, then use RL to evaluate all policies that are compliant with the permissive policy and pick one with the lowest expected cost. The selected candidate policy provides an upper bound of the minimal cost. They also estimate a lower bound of the minimal cost by solving an unconstrained RL problem and stop the iteration if the two bounds are close enough. Alshiekh et al. [7] study a similar problem of learning an optimal policy (optimization objective) while satisfying a safety specification during learning (safety objective), but they decouple the two objectives differently. The system dynamics is modeled both as a Markov decision process and as a deterministic transition system called abstraction. They consider the worst-case scenario for the safety objective and interpret the Markov decision process as a two-player game, where the learning agent picks an action at the current state, and an environment player chooses the successor state. By computing the winning region of the safety game, they identify the safe actions that prevent the agent from leaving the winning region and construct a *shield* accordingly. The shield monitors the actions of the agent and substitutes the selected actions by safe actions whenever it is necessary to avoid violating the safety specification. Note that such a shield can be constructed merely using

prior knowledge and is independent on the cost function for the optimization objective.

There is also work on solving safe RL problems using policy gradient methods [4, 42], with the safety objective encoded as a constraint function of the output policy. The key idea is to address the constraints by solving the dual problems. Without concerning safety during learning, these algorithms require little prior knowledge compared with other methods. Achiam et al. [4] extend an existing policy gradient algorithm to constrained RL problems with bounded expected cost. The key idea is to upper bound the expected costs of a new policy using the difference between the new policy and the current one, and to approximate the upper bound as a linear function. They manage to relax the primal problem as a convex optimization problem, which can be solved efficiently by solving the dual problem. Chow et al. [42] propose several primal-dual algorithms for finite Markov decision processes with percentile risk constraints. They derive unbiased estimators of the gradients of the Lagrangian function over all primal and dual variables and update them with different time scales. Their algorithms convergence almost surely to local saddle points of the Lagrangian function.

Chapter 2: Learning from Demonstrations with High-Level Side Information

2.1. Introduction

Learning from demonstration [10], also referred to as imitation learning or apprenticeship learning [1], aims at learning a *policy* to implement some *task*, using samples of an expert's behaviors as demonstrations. There is a wide range of applications of learning from demonstration in robotics, such as navigation and manipulation tasks.

One common approach to learning from demonstration is inverse reinforcement learning (IRL) [129], in which the agent relies on rewards to interpret the expert's behaviors. The environment is modeled as a Markov decision process (MDP) with known transition dynamics. Given the environment MDP and expert's demonstrations as trajectories, IRL recovers a reward function and constructs policies based on the estimated reward function. Formulations of IRL in the literature differ primarily in their interpretation of expert demonstrations, or the "similarity" between the expert's policy and desired policies expressed in terms of rewards. Some common assumptions are, for example, that both the expert's policy and all desired policies are optimal [1, 48, 146, 147]; or the expected total rewards of output policies should match the sample mean of total rewards of trajectories in demonstrations [23, 27, 202].

Although human experts can directly provide low-level demonstrations to implicitly specify the learning task, it is usually beneficial to explicitly indicate high-level task requirements, which we naturally rely on to assess the performance of the learned policies. A high-level task can be "grasp an object without touching anything else," or "obey traffic lights and road signs while driving from A to B," which may not be sufficient to encode all desired properties of an ideal policy, but is crucial to the task performance. Existing IRL methods do not infer underlying high-level tasks and thus the agent's behavior at newly visited states may not satisfy the actual task requirements.

In this work, we join the strengths of high-level task requirements and demonstrations and formalize

the problem of IRL with high-level side information. Given task specification as a co-safe linear temporal logic (LTL) formula, and a collection of optimal expert demonstrations consistent with the task specification, we describe a learning framework that recovers both a reward function, as well as a deterministic finite automaton (DFA), which together guarantee a quantitative level of probability that the learned policy will satisfy the task. Crucially, the addition of an LTL side specification allows us to learn general policies that work even when the expert examples are scarce.

Following the many applications of formal methods to robotics and control [24, 53, 94, 189], we encode the task requirements in LTL [139], which is an expressive formal logic suitable for task requirements. These include reaching-a-goal, stability, obstacle avoidance, sequentially visiting regions of interest, and conditional reactive behaviors. Generally, LTL specifications can be evaluated on trajectories with infinite length; but since expert demonstrations are finite, we focus on a set of tasks to be implemented in finite time, which can be specified by a subset of LTL called co-safe LTL [99].

We adopt the framework of maximum-likelihood inverse reinforcement learning (MLIRL) [12] as a baseline approach, and learn policies in both the original environment MDP and in the product space of the MDP and the specification automaton. We further propose an algorithm that evaluates the learned policy using the co-safe LTL formula during learning. We report numerical results on a navigation example, in which policies are learned with MLIRL, MLIRL with a specification automaton, and with our own algorithm. We show that the learned policy benefits from both the construction of product automaton and the evaluation with co-safe LTL formula, because it attains higher probabilities of successfully implementing the task, and provides formal guarantees on task completion even in regions of state space not covered by the expert examples.

2.2. Preliminaries

For a finite set B and a nonnegative integer $l \in \mathbb{N}^+$, define B^l as the set of all sequences of length l composed by elements in B. In addition, define B^* (resp. B^ω) as the set of all finite (infinite) sequences composed by elements of B. Finally, define $\mathcal{D}(B)$ as the set of all probability distributions

over B.

2.2.1. Markov Decision Processes and Policies

Let $\mathcal{M} = \langle S, S_I, A, T, R, \gamma \rangle$ be a *Markov decision process (MDP)*, where S is a finite set of states; $S_I \subseteq S$ is a set of initial states; A is a finite set of actions; $T : S \times A \times S \to [0, 1]$ is a transition function such that for each $(s, a) \in S \times A$, $T(s, a, \cdot) \in \mathcal{D}(S)$; $R : S \times A \to \mathbb{R}$ is a reward function, and $\gamma \in (0, 1)$ is a discounting factor.

A *path* or *trajectory* τ of $\mathcal{M}$ is an infinite alternating sequence of states and actions, $\tau = s_0, a_0, s_1, a_1, \ldots$, such that $s_0 \in S_I$, and for all $k \geq 0$, we have $a_k \in A$ and $T(s_k, a_k, s_{k+1}) > 0$. Given two states $s, s' \in S$, we say s' is *reachable* from s, denoted by $s \rightsquigarrow s'$, if and only if there exists a path $\tau = s_0, a_0, s_1, a_1, \ldots$ with $s = s_i$ and $s' = s_j$ for some integers $0 \leq i \leq j$. For any set of states $B \subseteq S$, define $Reach(B) = \{s' \in S : \exists s \in B, s' \rightsquigarrow s\}$ as the set of states from which B is reachable.

A (memoryless) *policy* π for $\mathcal{M}$ is a mapping from states to distributions over actions: $\pi : S \times A \to [0, 1]$ such that for any $s \in S$, $\pi(s, \cdot) \in \mathcal{D}(A)$. Given any policy π, we can define a state value function $V_\pi : S \to \mathbb{R}$ such that for each state $s \in S$, $V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \mid s_0 = s\right]$ is the expected future discounted reward that an agent can get by applying policy π from state s. Correspondingly, we can define an action value function $Q_\pi : S \times A \to \mathbb{R}$ such that for any state-action pair $(s, a) \in S \times A$, $Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \mid s_0 = s, a_0 = a\right]$ is the expected discounted reward if the agent takes policy π after taking action a from state s. The functions V_π, Q_π, R, and π satisfy the Bellman relations:

$$V_\pi(s) = \sum_a \pi(s, a) Q_\pi(s, a),$$

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') V_\pi(s').$$

These two equations can be combined into

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \sum_{a'} \pi(s',a') Q_\pi(s',a'). \tag{2.1}$$

2.2.2. Linear Temporal Logic Specifications

In order to evaluate policies with linear temporal logic (LTL) specifications, we attach labels to states. The labels, consisting of *atomic propositions*, are boolean variables defined on S. Let AP be a set of atomic propositions. The *labeling function* $\mathcal{L} : S \to 2^{AP}$ maps each state $s \in S$ to its labels $\mathcal{L}(s) \subseteq AP$, which is the set of atomic propositions that are true at state s. With slight abuse of notation, we also use $\mathcal{L}(\tau)$ to denote the sequence of atomic propositions that hold at states in path $\tau = s_0, a_0, s_1, a_1, \cdots$ of $\mathcal{M}$, i.e., $\mathcal{L}(\tau) = \mathcal{L}(s_0), \mathcal{L}(s_1), \cdots$.

An LTL formula φ over AP is defined recursively by:

$$\varphi := \text{true} \mid p \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \mathtt{X}\,\varphi_1 \mid \varphi_1\,\mathtt{U}\,\varphi_2,$$

where $p \in AP$, and φ_1, φ_2 are LTL formulas. The logical and temporal operators above can be combined to define other useful operators such as $\wedge$, $\rightarrow$, `G` and `F`. See [139] for a detailed explanation of the semantics of LTL.

In general, an LTL formula φ is evaluated on $(2^{AP})^\omega$, i.e., infinite sequences of elements in 2^{AP}. To better suit the need to encode tasks that are implemented over finite horizons, we focus on a subset of LTL formulas, namely co-safe LTL formulas. These formulas are characterized by the key feature that every (infinite) sequence that satisfies the formula has a finite prefix [99]. A wide range of learning from demonstration tasks that can be encoded as co-safe LTL formulas, for example: $\varphi_1 = (\neg\text{obstacle}\,\mathtt{U}\,\text{goal}) \wedge \mathtt{F}\,\textit{goal}$ means "reach the goal without running into obstacles," and $\varphi_2 = \big((\neg\text{object}_2)\,\mathtt{U}\,\text{object}_1\big) \wedge (\,\mathtt{F}\,\text{object}_1) \wedge (\,\mathtt{F}\,\text{object}_2)$ means "grab object 1 first and then grab object 2."

Given a co-safe LTL formula φ_{cs}, we can construct a (non-unique) deterministic finite automaton (DFA) $\mathcal{A}_{\varphi_{\text{cs}}} = \langle Q, q_I, Q_F, 2^{AP}, \delta \rangle$ that accepts the finite prefixes all runs that satisfy φ_{cs}, where Q is a finite set of states, $q_I \in Q$ is the initial state, $Q_F \subseteq Q$ is a set of accepting (final) states, 2^{AP} is the alphabet, and $\delta : Q \times 2^{AP} \to Q$ is a deterministic transition function. All states in Q_F are absorbing states, i.e., for all $L \in 2^{AP}, q \in Q_F, \delta(q, L) = q$.

2.3. Maximum-Likelihood Inverse Reinforcement Learning (MLIRL)

We adopt the framework of maximum-likelihood inverse reinforcement learning (MLIRL) [12] as the baseline algorithm that does not use any high-level side information. In this section we introduce the key components of MLIRL: policy structure, reward parameterization, and optimization objective.

Softmax policy. We restrict the policy search to the subclass of policies for which the probability to take an action a at state s is a softmax function of the action-value function. For any function $Q : S \times A \to \mathbb{R}$, define the *softmax* policy as

$$\pi_Q(s, a) := \frac{\exp\big(Q(s, a)\big)}{\sum_{\tilde{a}} \exp\big(Q(s, \tilde{a})\big)}, \quad \forall (s, a) \in S \times A. \tag{2.2}$$

The softmax policy, as a special case of the Boltzmann exploration policy [79], has been used in several instances of IRL [12, 114, 127]. It defines a valid distribution π_Q for all Q, i.e., $\pi_Q(s, a) \geq 0$ for all $s \in S$ and $a \in A$, and $\sum_{a \in A} \pi_Q(s, a) = 1$ for all $s \in S$. It is also smooth in the components of Q, allowing easy computation of a policy gradients. With the softmax policy, the agent prefers to select actions with higher action-values, but still has the freedom to explore suboptimal actions. Such freedom is particularly important in accommodating any inconsistency in the expert's demonstrations.

Reward parameterization. The reward function R of $\mathcal{M}$ is approximated by a linear combination of k pre-designed features, with parameter $\theta \in \mathbb{R}^{k \times 1}$. We denote the feature matrix as $F = [\mathbf{f}_1, \cdots \mathbf{f}_k] \in \mathbb{R}^{|S| \cdot |A| \times k}$ with $\mathbf{f}_i$ representing the i^{th} reward feature vector. For convenience, we denote the row of F corresponding to the state-action pair (s, a) as $F(s, a) \in \mathbb{R}^{1 \times k}$. The overall reward matrix is $R = F\theta$ for some feature weight $\theta \in \mathbb{R}^k$. Substituting the softmax policy and the reward

matrix into (2.1) yields

$$Q(s,a) = F(s,a)\theta + \gamma \sum_{s'} T(s,a,s') \sum_{a'} \pi_Q(s',a') Q(s',a'). \tag{2.3}$$

In the following, we treat θ as the free variable, and denote the action value function Q and policy π_Q satisfying (2.2) and (2.3) as Q_θ and π_θ.

Expert demonstrations. The demonstrations consist of a set $D = \{\tau_1, \cdots \tau_m\}$ of m finite prefixes of trajectories in $\mathcal{M}$. For each $l \in \{1, \cdots m\}$, $\tau_l = s_{l,0}, a_{l,0}, \cdots s_{l,t_l}, a_{l,t_l}$ is the l^{th} demonstration trajectory, which is an ordered sequence of $t_l + 1$ state-action pairs. We refer to such demonstrated trajectories as expert trajectories.

Maximum-likelihood objective. The goal is to find θ and an induced policy π_θ that maximize the likelihood of observing the expert demonstrations. An equivalent optimization objective is to minimize the negative log-likelihood

$$\begin{aligned} J^{\text{mle}}(\theta \mid \mathcal{M}, D) &:= -\sum_{l=1}^{m} \sum_{t=1}^{t_l} \log \pi_\theta(s_{l,t}, a_{l,t}) \\ &= -\sum_{l=1}^{m} \sum_{t=1}^{t_l} \Big(Q_\theta(s_{l,t}, a_{l,t}) - \log \big(\sum_{\tilde{a}} \exp \big(Q_\theta(s_{l,t}, \tilde{a}) \big) \big) \Big), \end{aligned} \tag{2.4}$$

with equality constraints given by Eq. (2.2) and (2.3). The objective function is smooth and convex in θ, but regularization on θ may be needed to avoid separation problems, and to get a finite solution [6].

2.4. MLIRL with High-Level Side Information

Assume that in addition to the standard inputs to MLIRL problems, i.e., a reward-free MDP $\mathcal{M}$, expert demonstrations D, and feature matrix F, we also know some high-level task requirements encoded as a co-safe LTL formula φ_{cs}. This side information is utilized in two steps: we first extend the original MDP $\mathcal{M}$ into a product automaton incorporating the task structure, and then augment the optimization objective by explicitly evaluating the policy.

2.4.1. Extending the State Space

An implicit assumption in all MDP-based IRL methods is that the expert's policy is memoryless, i.e., the distribution of the next action is decided by the current state and independent on trajectory history. The assumption breaks if the task has some hierarchical structure and can be easily decomposed into several sub-tasks, which is a common case in practice. Side information as high-level task requirements can be used to generate memory states automatically, which enables us to construct a product automaton $M_{\varphi_{\text{cs}}}$ with the original environment $\mathcal{M}$ and a DFA $\mathcal{A}_{\varphi_{\text{cs}}}$. Then we learn a memoryless policy over the extended state space of $M_{\varphi_{\text{cs}}}$.

Given $\mathcal{A}_{\varphi_{\text{cs}}}$ and $\mathcal{M}$, define the *product automaton* $M_{\varphi_{\text{cs}}} = \langle \bar{S}, \bar{S}_I, \bar{S}_F, A, \bar{T}, \gamma \rangle$, where $\bar{S} = S \times Q$ is a finite state space; $\bar{S}_I = S_I \times q_I$ is the set of initial states; $\bar{S}_F = S \times Q_F$ is the set of final states; $\bar{T} : \bar{S} \times A \times \bar{S} \to [0, 1]$ is a transition function such that for any $(s, q), (s', q') \in \bar{S}, a \in A$, $\bar{T}\big((s, q), a, (s', q')\big) = T(s, a, s')$ if $\delta\big(q, \mathcal{L}(s')\big) = q'$ and 0 otherwise. Policies in $M_{\varphi_{\text{cs}}}$ can be defined analogously to those in $\mathcal{M}$. Similar to the evaluation of $\mathcal{A}_{\varphi_{\text{cs}}}$, a finite path $\tau_M = (s_0, q_0), a_0, (s_1, q_1), a_1 \cdots (s_l, q_l), a_l \in (\bar{S} \times A)^{l+1}$ of $M_{\varphi_{\text{cs}}}$ satisfies φ_{cs} if and only if $(s_l, q_l) \in \bar{S}_F$, or equivalently $q_l \in Q_F$.

Any finite (resp. infinite) trajectory in $\mathcal{M}$ can be uniquely mapped to a trajectory of equal length in the product automaton $M_{\varphi_{\text{cs}}}$. We define an operator $h(\cdot \mid \mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}}) : (S \times A)^* \to (\bar{S} \times A)^*$ to translate finite trajectories in $\mathcal{M}$ into the corresponding trajectories in the product automaton $M_{\varphi_{\text{cs}}}$. The operator h enables us to interpret the demonstrations D in $M_{\varphi_{\text{cs}}}$. For any $\tau_l \in D$, define

$$h(\tau_l \mid \mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}}) = \bar{s}_{l,0}, a_{l,0}, \bar{s}_{l,1}, a_{l,1}, \cdots, \bar{s}_{l,t_l}, a_{l,t_l},$$

such that $\bar{s}_{l,0} := (s_{l,0}, q_I)$ and for $j = 1, \cdots, t_l$, $\bar{s}_{l,j} := \Big(s_{l,j}, \delta\big(q_{l,j-1}, \mathcal{L}(s_{l,j})\big)\Big)$. Any trajectory in $M_{\varphi_{\text{cs}}}$ can be uniquely projected to a trajectory in $\mathcal{M}$, simply by dropping the second component of each state in $\bar{S}$. In the following, we assume that the learning procedure occurs in the product automaton in order to take advantage of the side information. For simplicity we use $h(D \mid \mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}}) := \{h(\tau_l \mid \mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}}) : \tau_l \in D\}$ to represent the set of projected trajectories of D

in $M_{\varphi_{\text{cs}}}$.

The construction of $\mathcal{A}_{\varphi_{\text{cs}}}$ and $M_{\varphi_{\text{cs}}}$ is internal to the learning algorithm and may not be accessible by the expert. Correspondingly the agent has no access to the expert policy. The only shared inputs between the agent and expert are the high-level task specification φ_{cs}, the environment dynamics $\mathcal{M}$, and the set D of demonstrated trajectories in $\mathcal{M}$. Any equivalent DFA for φ_{cs} works in principle, except with varying computation time due to possibly different sizes of $M_{\varphi_{\text{cs}}}$.

2.4.2. Augmenting the Objective with Side Information

In order to guarantee the performance of the learned policy, we explicitly compute the probability of satisfying φ_{cs} from all valid initial states. This is done by computing a function $\bar{y}(\cdot \mid \bar{\pi}) : \bar{S} \to [0, 1]$ such that $\bar{y}(\bar{s} \mid \bar{\pi})$ is the probability of satisfying φ_{cs} by taking policy $\bar{\pi}$ from initial state $\bar{s}$. By a result in model checking [13],

$$\bar{y}(\bar{s} \mid \bar{\pi}) = \begin{cases} 1, & \text{if } \bar{s} \in \bar{S}_F, \\ 0, & \text{if } \bar{s} \notin \mathit{Reach}(\bar{S}_F), \\ \sum\limits_{a \in A} \bar{\pi}(\bar{s}, a) \sum\limits_{\bar{s}' \in \bar{S}} T(\bar{s}, a, \bar{s}') \bar{y}(\bar{s}' \mid \bar{\pi}), & \text{otherwise.} \end{cases} \tag{2.5}$$

There is a unique $\bar{y}(\cdot \mid \bar{\pi})$ for any given $\bar{\pi}$; it can be obtained either by linear programming, or by computing the least fixed point of the operator

$$\Gamma_{\bar{\pi}}(\bar{y})(\bar{s}) = \begin{cases} 1, & \text{if } \bar{s} \in \bar{S}_F, \\ 0, & \text{if } \bar{s} \notin \mathit{Reach}(\bar{S}_F), \\ \sum\limits_{a \in A} \bar{\pi}(\bar{s}, a) \sum\limits_{\bar{s}' \in \bar{S}} T(\bar{s}, a, \bar{s}') \bar{y}(\bar{s}' \mid \bar{\pi}), & \text{otherwise.} \end{cases}$$

Assume $\bar{y}^{(0)}(s) = 0$ for all $\bar{s} \in \bar{S} \backslash \bar{S}_F$ and $\bar{y}^{(0)}(s) = 1$ for all $s \in \bar{S}_F$, and $\bar{y}^{(k)}$ is updated as $\bar{y}^{(k+1)} = \Gamma_{\bar{\pi}}(\bar{y}^{(k)})$ for all $k \in \mathbb{N}$, then it can be shown that $\lim_{k \to +\infty} \bar{y}^{(k)}$ exists and is the unique solution to (2.5) [13]. Note that since $\pi_\theta(\bar{s}, a) > 0$ for all θ and $(\bar{s}, a)$ such that there exists $\bar{s}' \in \bar{S}$

with $\bar{T}(\bar{s}, a, \bar{s}') > 0$ by definition of softmax policy, $Reach(\bar{S}_F)$ is independent on θ.

We can augment the MLIRL objective (2.4) by adding a non-decreasing differentiable function $g : \mathbb{R}^{|\bar{S}|} \to 1$ of $\bar{y}(\cdot \mid \pi_\theta)$ to explicitly consider the performance of π_θ with respect to the task specification. The new objective is to minimize

$$J^{\text{side}}\big(\theta \mid M_{\varphi_{\text{cs}}}, h(D|\mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}})\big) = J^{\text{mle}}\big(\theta \mid M_{\varphi_{\text{cs}}}, h(D|\mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}})\big) - \mu \cdot g(\bar{y}), \tag{2.6}$$

where $\mu > 0$ is a trade-off parameter adjusting the weight between the objective and the task performance objective. The optimization is subject to constraints (2.2), (2.3) and (2.5).

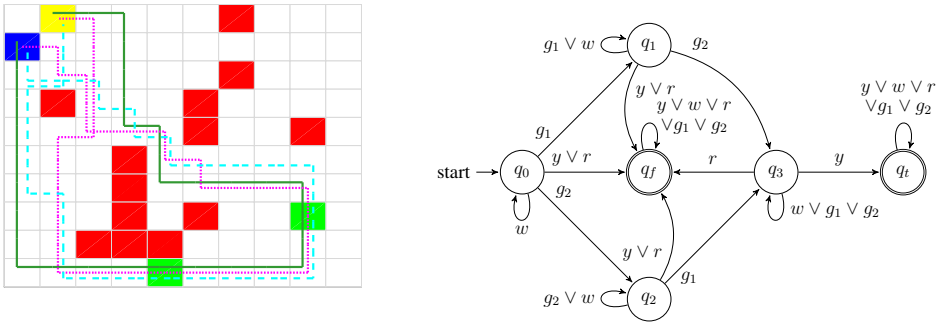
We solve the optimization problem by gradient descent, in which the key is to compute the derivative of Q_θ, π_θ and $\bar{y}$ with respect to θ. For any matrix B, we denote its component at row i and column j as $B(i, j)$. If we assume that π_θ does not change much in the neighborhood of θ, we can estimate $\frac{\partial Q_\theta}{\partial \theta}$ while considering π_θ as constant. Then for any $i = 1, \cdots k$ and $(\bar{s}, a) \in \bar{S} \times A$,

$$\frac{\partial Q_\theta(\bar{s}, a)}{\partial \theta_i} = F_i(\bar{s}, a) + \gamma \sum_{\bar{s}' \in \bar{S}} \sum_{a' \in A} T(\bar{s}, a, \bar{s}') \pi_\theta(\bar{s}', a') \frac{\partial Q_\theta(\bar{s}', a')}{\partial \theta_i}. \tag{2.7}$$

$$\frac{\partial \pi_\theta(\bar{s}, a)}{\partial \theta_i} = \pi_\theta(\bar{s}, a)\Big(\frac{\partial Q_\theta(\bar{s}, a)}{\partial \theta_i} - \sum_{\tilde{a}} \pi_\theta(\bar{s}, \tilde{a}) \frac{\partial Q_\theta(\bar{s}, \tilde{a})}{\partial \theta_i}\Big). \tag{2.8}$$

$$\begin{aligned} &\frac{\partial}{\partial \theta_i} \bar{y}(\bar{s}) = \sum_{a \in A} \pi_\theta(\bar{s}, a) \sum_{\bar{s}' \in \bar{S}} T(\bar{s}, a, \bar{s}') \left(\frac{\partial \bar{y}(\bar{s}')}{\partial \theta_i} + \Big(\frac{\partial Q_\theta(\bar{s}, a)}{\partial \theta_i} - \sum_{\tilde{a}} \frac{\partial Q_\theta(\bar{s}, \tilde{a})}{\partial \theta_i} \Big) \bar{y}(\bar{s}') \right), \\ &\frac{\partial}{\partial \theta_i} \bar{y}(\bar{s}) = 0, \text{ if } \bar{s} \in \bar{S}_F \bigcup \big(\bar{S} \backslash Reach(\bar{S}_F)\big). \end{aligned} \tag{2.9}$$

The derivatives of Q_θ, π_θ and $\bar{y}$ with respect to θ are unique solutions of (2.7)–(2.9), given π_θ and $\bar{y}$. The uniqueness of the solution $\frac{\partial Q_\theta}{\partial \theta_i}$ in (2.7) holds for any stationary (i.e., time-invariant) policy π_θ given that F is bounded [21], which trivially holds as F is fixed. The uniqueness of the solution $\frac{\partial \bar{y}}{\partial \theta_i}$ in (2.9) holds for any stationary policy π_θ, $\bar{y}$ and $\frac{\partial Q_\theta}{\partial \theta}$, which can be proved by contradiction: assume that there exist two different functions $y_1, y_2 : \bar{S} \to \mathbb{R}$ that are both solutions to (2.9). Then for any



(a) (b)

Figure 1: Illustration of the grid world example. Left: grid world map and demonstration trajectories. Right: an equivalent DFA for φ_{cs}.

$\bar{s} \in \bar{S}$,

$$y_1(\bar{s}) - y_2(\bar{s}) = \sum_{a \in A} \pi_\theta(\bar{s}, a) \sum_{\bar{s}' \in \bar{S}} T(\bar{s}, a, \bar{s}') \big(y_1(\bar{s}') - y_2(\bar{s}')\big).$$

As (2.5) is known to have a unique solution, $y_1(\bar{s}) - y_2(\bar{s})$ has to be zero for all $\bar{s} \in \bar{S}$, which leads to a contradiction to the assumption. Therefore (2.9) has a unique solution $\frac{\partial \bar{y}}{\partial \theta_i}$.

2.5. Experimental Results

We illustrate our approach on a path planning task in a 10-by-10 grid world map, as shown in Figure 1a. Each cell represents a state in $\mathcal{M}$, from which the agent has 4 available actions: up, down, left, and right. States are labeled by their colors: *r* (red), *w* (white), *y* (yellow) and *b* (blue). The two green states are labeled as g_1 (green$_1$) and g_2 (green$_2$). The yellow state is an absorbing state, i.e., it has no outgoing transitions.

The specification task is to visit both green cells (intermediate goals) in any order, and end at the yellow cell (final goal), while avoiding red cells (obstacles). These requirements are encoded as the co-safe LTL formula

$$\varphi_{\text{cs}} = \varphi_{\text{init}} \rightarrow (\varphi_{\text{safe}} \wedge \varphi_{\text{goal}}),$$

where

$$\varphi_{\text{init}} = \neg r \wedge \neg y \qquad \text{(Initial state)},$$

$$\varphi_{\text{safe}} = \neg r \,\mathtt{U}\, y \qquad \text{(Obstacle avoidance)},$$

$$\varphi_{\text{goal}} = \big((\neg y)\,\mathtt{U}\,(\,\mathtt{F}\, g_1 \wedge\, \mathtt{F}\, g_2)\big) \wedge (\,\mathtt{F}\, y) \qquad \text{(Goal reaching)}.$$

An equivalent DFA is shown in Figure 1b. Each state in the DFA corresponds to some task status (see Table 2), and each transition represents some progress toward task completion. These transitions can be automatically encoded into features to facilitate learning. In fact, three features were constructed this way (see $\mathbf{f}_2, \mathbf{f}_3, \mathbf{f}_4$ in Table 1). The other two features come from transitions observed in demonstrations ($\mathbf{f}_1$), and a penalty for each transition ($\mathbf{f}_5$). Note that the final state has no outgoing transitions, and outgoing loops have zero reward.

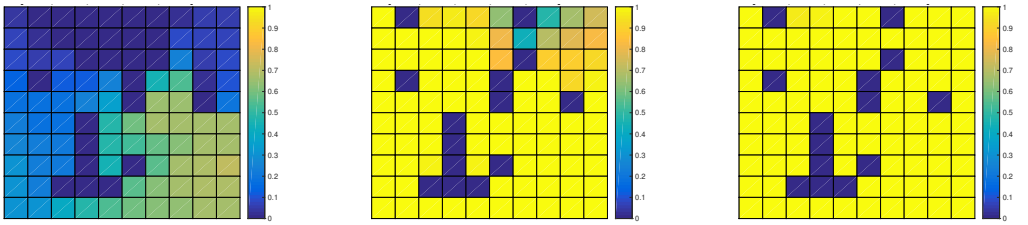
The agent is given a set of demonstrated trajectories that successfully implemented the task, as shown in Figure 1a. In this example all demonstrated trajectories start from the blue cell, pass both green cells (in arbitrary order), avoid all red cells, and eventually end at the yellow cell. States in the upper right corner are never observed in demonstration.

We now discuss the learned policy in three different cases, where the agent is provided with a different amount of side information, and the policies are learned with or without high-level task information.

Case 1 (MLIRL in $\mathcal{M}$). The agent only knows about the MDP $\mathcal{M}$, the labeling function $\mathcal{L}$, and the demonstrations $\mathcal{D}$, and learns a policy with MLIRL, i.e., by minimizing $J^{\text{mle}}(\theta \mid \mathcal{M}, D)$ in Eq. (2.4) while satisfying the constraints (2.2) and (2.3). Since the agent does not know φ_{cs} or the DFA, we cannot use all features from Table 1. Instead, we replace $\mathbf{f}_2(\bar{s}, a)$ and $\mathbf{f}_3(\bar{s}, a)$ by the

Table 1: Design of features in Case 2 and Case 3.

Feature	Explanation
$\mathbf{f}_1$	$f_1(s,a) = 1$ if (s,a) appeared in demonstration; otherwise $f_1(s,a) = 0$.
$\mathbf{f}_2$	$f_2(s,a)$ is the probability to reach q_t for the first time by taking a at state s.
$\mathbf{f}_3$	$f_3(s,a)$ is the probability to reach q_2, q_3 for the first time by taking a at state s.
$\mathbf{f}_4$	$f_4(s,a)$ is the negative probability to reach red states.
$\mathbf{f}_5$	$f_5(s,a) = -1$ if $s \notin \bar{S}_F$.



(a) MLIRL policy in $\mathcal{M}$ (Case 1). Min prob: 6.90×10^{-9}; average prob: 0.304.

(b) MLIRL policy in $M_{\varphi_{\text{cs}}}$ (Case 2). Min prob: 0.430; average prob: 0.954.

(c) Policy with augmented objective (Case 3). Min prob: 0.962; average prob: 0.999.

Figure 2: Probability of satisfying φ_{cs} when following the corresponding policy from each initial state.

probability of reaching the yellow state or a green state from $\bar{s}$ by taking a. All other features have the same interpretation as in Table 1. The learned feature weight vector is

$$\hat{\theta}^{(1)} = [8.5176, 4.2678, -0.0442, -0.8208, 3.7336]^{\intercal}.$$

The sign of the learned weights is instructive: they define a policy that seeks to follow demonstrations ($\mathbf{f}_1$) and tries to reach the yellow state ($\mathbf{f}_2$) in a timely manner ($\mathbf{f}_5$). As the weights of $\mathbf{f}_3$ and $\mathbf{f}_4$ are negative, the agent fails to realize the importance of visiting green states and avoiding red states. There are at least two reasons for such behavior. First, as there is a feature ($\mathbf{f}_1$) marking the state-action pairs observed in demonstrations, the agent may simply try to follow the demonstrations whenever possible to minimize $J^{\text{mle}}(\theta \mid \mathcal{M}, D)$, without further reasoning about the demonstrations, which results in overfitting. Second, there is no side information for the agent to evaluate its policy or identify important features. As shown in Figure 2a, the agent behaves best in the lower right region, where it can follow some expert expert demonstration easily; it behaves the worst in the upper middle region, where the expert demonstrations are lacking.

Case 2 (MLIRL in $M_{\varphi_{\text{cs}}}$). The agent has all inputs in Case 1 and the DFA, and learns a policy with MLIRL within the product automaton. Compared with Case 1, the agent can now construct a product automaton. With the extended state space, the agent may behave differently based on the current status with respect to intermediate goals and potentially learn the importance of avoiding red

Table 2: Interpretation of DFA states.

DFA State	Interpretation
q_0	None of g_1, g_2, y or r visited.
q_1	Visited g_1, never visited g_2, y, r.
q_2	Visited g_2, never visited g_1, y, r.
q_3	Visited g_1 and g_2, never visited y, r.
q_t	Visited g_1, g_2, y without visiting r (success).
q_f	Visited r, or visited y before visiting both g_1 and g_2 (failure).

cells. A simple check of the structure of the product automaton reveals that any visit to a red cell will lead to a transition to q_f in the DFA, which makes it impossible to reach $\bar{S}_F$ later. Therefore in order to reach a final state, it is necessary to add some penalty on visiting red states. The learned feature weight vector is

$$\hat{\theta}^{(2)} = [9.6090, 2.3128, 2.7393, -0.0121, 2.3221]^\intercal.$$

As shown in Figure 2b, the probability of satisfying φ_{cs} has been greatly improved. The weight for $\mathbf{f}_3$ is away from zero as expected, but the weight for $\mathbf{f}_4$ is still small, which suggests that the agent still has not learned to always avoid red cells. As a result, the probability of task success is the lowest in the upper right region, which is not covered by demonstrations. The two sources of problems explained in Case 1 still exist here, which calls for the augmentation of objective function using LTL side information.

Case 3 (Policy with augmented objective). The agent has the same input as in Case 2, but now the policy is learned with the augmented objective function J^{side} in (2.6), where we set $g(\bar{y}) = \sum_{\bar{s}\in\bar{S}} \bar{y}(\bar{s} \mid \pi_\theta)$, i.e., the sum of probabilities of satisfying φ_{cs} from all initial states. With $\mu = 0.01$, the learned feature weight vector is

$$\hat{\theta}^{(3)} = [10.2010, 1.8908, 3.9550, 8.1854, 1.8855]^\intercal.$$

Compared with $\hat{\theta}^{(2)}$, the most significant change in $\hat{\theta}^{(3)}$ is that the weight on $\mathbf{f}_4$ is almost as large as $\mathbf{f}_1$, and much larger than the weights on other features. The agent now learns the importance of avoiding red states, and the performance with respect to task implementation has been significantly

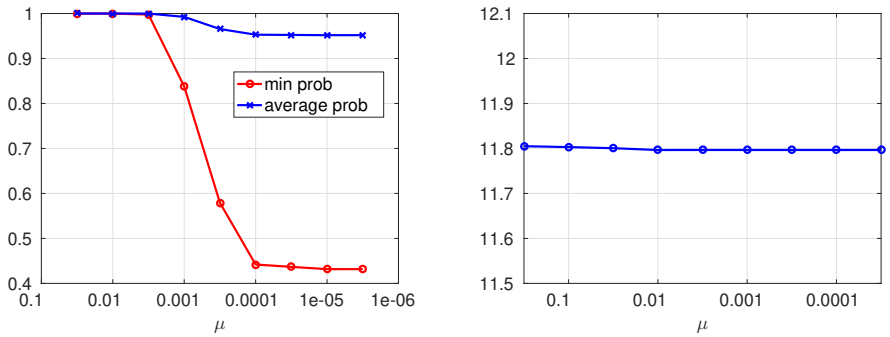


Figure 3: Probability of satisfying φ_{cs} (left) and negative log-likelihood of the state-action pairs in demonstration (right), as function of μ.

improved, especially from states that demonstrations fail to cover, as shown in Figure 2c. It shows that, by evaluating policies with side information φ_{cs}, the agent manages to get rid of the overfitting problem and the induced policy can now be generalized well into regions not previously seen in demonstrations.

To check the effect of the weight μ, we solved Case 3 with a series of μ and plotted the corresponding minimum and average probabilities of satisfying specification from all possible initial states, and corresponding negative log-likelihood of demonstrated trajectories (see Figure 3). Each experiment is repeated three times. Note that the value of the original objective $J^{\text{mle}}(\theta \mid M_{\varphi_{\text{cs}}}, D)$ is only slightly affected by μ, while the average probability of satisfying φ_{cs} is very sensitive to μ. This confirms that the augmentation of the objective function with LTL side information is necessary.

If the given high-level task description is incomplete or inaccurate, as is often the case in practice, demonstrations can compensate for an imperfect specification as long as the features are expressive enough, and no extra memory is needed to implement the missing part of the task.

To illustrate this point, we learned a policy where the side requirement to avoid red cells φ_{safe} is missing, and evaluated it with respect to the *true* task encoded by φ_{cs}. The result is shown in Figure 4. We observe the overall performance is only slightly worse than the case with accurate high-level task information, which suggests that the agent manages to learn from expert demonstration to avoid visiting red states at most initial states. Still, performance can be significantly worse at states that are

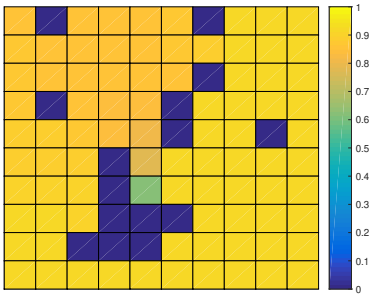


Figure 4: Probability of satisfying φ_{cs} for policies learned without the obstacle avoidance requirement.

close to obstacles, such as the cell in row 7, column 5. This example illustrates the agent's ability to learn actions preferences from demonstrations. However if the inaccurate high-level task information leads to insufficient memory states, the performance of the learned policies can be poor, as it is impossible to recover enough missing memory states from demonstration purely by learning the rewards.

Chapter 3: Task-Oriented Deep Inverse Reinforcement Learning

3.1. Introduction

The topic of teaching robots to implement tasks via demonstrations, also called *learning from demonstrations* (LfD) [10], has been studied for many years. Given a set of demonstration trajectories, the goal of LfD is to learn a policy, which is a mapping from states to distributions over actions, that imitates the demonstrations in a particular way. Intuitively, it is easier to show robots how to implement a task in a specific scenario than to design a general controller, which may require significant human effort and specialized knowledge. There are two major approaches to LfD [133]: One is behavioral cloning (BC) or imitation learning, which treats policy learning as a supervised learning problem. It is possible to do inference about the task structure and learn several sub-policies for the overall task [97, 103, 131], but the goal is still to directly learn a policy by doing statistical analysis of the demonstrations. The other approach is inverse reinforcement learning (IRL) [1, 202], which directly outputs reward functions and can only generate policies by interacting with a specific environment.

One key concern of LfD is how to generalize the demonstrations to new scenarios, which is almost always necessary in practice. Ideally, we expect the robot to not only learn a policy to implement the task in exactly the same environment where it has seen demonstrations, but also be adaptive to reasonably similar environments. Compared with BC, IRL is better suited for this purpose due to the following reasons. By learning a reward function, IRL essentially aims at learning a representation of the task [129] and inferring the demonstrator's intent [12]. Moreover, the policies learned by IRL in new environments are computed specifically for the new environments using the learned reward function.

Most existing work on IRL learns a reward function merely by observing a set of expert demonstrations in a given environment. Unfortunately, this common setting makes it prohibitively difficult to achieve good performance in new environments. The problem of IRL is ill-posed as many different

reward functions can lead to the same policy. As a result, the learned reward function may not be a reliable task performance criterion even in the training environment, let alone new environments in which no demonstration has been observed.

In contrast, human rarely needs to learn new skills only from state-level observations. In many cases, we have some high-level side information of the task that is implemented by the demonstrations, which greatly reduces the learning effort. For example, the whole task may be partitioned into several subtasks, where each subtask should be implemented using an independent policy. The subtasks may be implemented in any order, or have to be implemented in some specific order. Some subtasks may only be required if the robot observes some certain condition, otherwise it is fine skip them.

Although high-level task information can be easily acquired for humans, it is difficult, if not impossible, to be inferred accurately from demonstration trajectories. Assume that you are to learn how to repair cars (the task) but have zero knowledge about the components (features) or they should be examined and repaired (the task structure). You get the chance to watch several videos (demonstrations) that show how an experienced auto mechanic have repaired several cars, but there is no explanation on the goal for each step or why these steps are necessary. It is not hard to imagine that the auto mechanic's demonstrations will be very ambiguous and confusing: There are too many possible interpretations of the demonstrations and it is not clear how to select one. In general, the demonstrator's policy, e.g., the mechanic's car-repairing policy, is affected by both the task and the specific environment, e.g., the specific condition of the car to be repaired. In order to reconstruct the task information, the robot has to identify the tasks and infer the conditions to take each one of them, which generally requires both positive and negative examples in expert demonstrations over many different environments.

Considering the difficulty of task inference and the easy access to high-level task information, we propose to incorporate such high-level task information directly as part of the input to IRL. In this chapter, we encode the high-level task information as a deterministic finite automaton (DFA), which tracks the progress in task implementation. It is well-known that DFA can be used to represent all regular expressions [164], which is commonly used to represent search patterns. In practice, the

input DFA can either be manually designed or be transformed from formal language specifications using off-the-shelf tools.

3.2. Preliminaries

Generally, the problem of inverse reinforcement learning (IRL) can be described as follows: Given an environment model $\mathcal{M}$ and a set of demonstration trajectories D, infer a reward function R that can optimally interpret the demonstrations in some pre-specified way. Different works on IRL can be distinguished from each other from the following three aspects: First, the reward parameterization; second, the way to generate a policy with a given reward function; third, the interpretation of the demonstrations using the output policy. In this section, we describe two existing IRL algorithms: maximum entropy inverse reinforcement learning (MaxEnt IRL) algorithm [202] and one of its deep variant, MaxEnt Deep IRL Algorithm [192, 195]. For each algorithm, we show their limitations for the purpose of generalizing to new environments, which motivate our algorithm.

We adopt the following setting on environment model and demonstrations that are commonly used in IRL works. The environment is modeled as a reward-free Markov decision process $\mathcal{M} = \langle S, A, T, \rho, \gamma \rangle$ where S is a state space; A is an action space; $T : S \times A \to \mathcal{D}(S)$ (where $\mathcal{D}(S)$ is the set of all probability distributions over S) is the transition distribution; $\rho \in \mathcal{D}(S)$ is an initial distribution over S and $\gamma \in (0, 1)$ is a discount factor. Let $D = \{\tau_1, \ldots, \tau_N\}$ be a set of demonstration trajectories, where $\tau_i = \{(s_{i,0}, a_{i,0}), \ldots, (s_{i,n_i}, a_{i,n_i})\}$ for all $i = 0, \ldots, n_i$.

3.2.1. Maximum Entropy IRL

In the original MaxEnt IRL algorithm [202], the reward function is parameterized as a linear combination of a given set of feature functions. In other words, given a set of features $\{f_1, \ldots, f_K\}$ where $f_k : S \times A \to \mathbb{R}$ for $k = 1, \ldots, K$, the reward function R_θ is parameterized by $\theta = [\theta_1, \ldots, \theta_K]^\intercal$ such that

$$R_\theta(s, a) = \sum_{k=1}^{K} \theta_k f_k(s, a).$$

The basic assumption is that the expected total reward over the distribution of trajectories is the same as the empirical average reward over demonstration trajectories. With the principle of maximum entropy, it can be derived that the probability of generating any (finite-length) trajectory $\tau = s_0, a_0, \ldots, s_{|\tau|}, a_{|\tau|}$ is proportional to the exponent of the total reward of τ:

$$Pr(\tau|R_\theta) \propto \exp \frac{1}{|\tau|} \sum_{i=0}^{|\tau|} R_\theta(s_i, a_i). \tag{3.1}$$

While linear parameterization is commonly used in IRL literature [1, 88, 127, 147], it suffers from several drawbacks. On the one hand, it requires human knowledge to provide properly designed reward features, which can be labor-intensive; on the other hand, if the given features fail to encode all the essential requirements to generate the demonstrations, there is no way to recover this flaw by learning from demonstrations. One way to deal with this problem is to use nonlinear reward models such as Gaussian process [105], decision trees [104] or neural network to automatically construct reward features from expert demonstrations.

3.2.2. Maximum Entropy Deep IRL

An IRL algorithm is generally referred to as a deep IRL algorithm if the reward is modeled as a neural network. There are several existing works on deep IRL [56, 192, 194, 195] that originate from MaxEnt IRL, largely due to the properties that Q_θ is independent from π_θ and implicitly derives π_θ by $\pi_\theta(a|s) \propto \exp Q_\theta(s, a)$. We take the MaxEnt deep IRL algorithm (MEDIRL) [195] as an example. Unlike the previous case where the reward function is modeled as a linear combination of pre-specified features, the reward is modeled as a convolutional neural network in MEDIRL. The reward parameter θ is the weight vector of the reward network. The objective is to maximize the posterior probability of the demonstration trajectories given a prior distribution $P(\theta)$ of θ:

$$L(\theta) := \log Pr(D, \theta) = \underbrace{\log Pr(D|R_\theta)}_{L_D} + \underbrace{\log P(\theta)}_{L_\theta}. \tag{3.2}$$

L_D is the log likelihood of the demonstration trajectories in D given the reward function R_θ. Let π_θ be the policy corresponding to R_θ, then L_D can be expressed as

$$L_D = \sum_{\tau_i \in D} \sum_{j=0}^{n_i - 1} \log \pi_\theta(a_{i,j}|s_{i,j}) + C, \tag{3.3}$$

where C is a constant that is dependent on D and the transition distribution T. L_θ can be interpreted as either the logarithm of the prior distribution $P(\cdot)$ at θ or as a differentiable regularization term on θ. The original MEDIRL algorithm was also designed for finite-horizon problems, but has been extended to the infinite-horizon case [23]. In this chapter, we adopt the infinite-horizon setting, but the same algorithm can be easily adapted to solve finite-horizon prolems.

For MaxEnt IRL, the computation of π_θ given R_θ is essentially a maximum entropy reinforcement learning problem. It can be proved [201] that for any R_θ, there exists a unique Q which is the (unique) fixed point of (3.4).

$$Q_\theta(s,a) = \quad R_\theta(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \log \sum_{a'} \exp(Q_\theta(s',a')). \tag{3.4}$$

The policy π_θ can be represented as the explicit expression of Q_θ in (3.5).

$$\pi_\theta(a|s) = \frac{Q_\theta(s,a)}{\sum_{a'} \exp(Q_\theta(s',a'))}. \tag{3.5}$$

The gradient of π_θ and Q_θ can be computed as in (3.6).

$$\begin{aligned} \frac{\partial Q_\theta(s,a)}{\partial \theta} =& \frac{\partial R_\theta(s,a)}{\partial \theta} + \gamma \sum_{s'} T(s'|s,a) \sum_{a'} \pi_\theta(a'|s') \frac{\partial Q_\theta(s',a')}{\partial \theta}, \\ \frac{\partial \pi_\theta(a|s)}{\partial \theta} =& z_\theta(s,a) - \pi_\theta(a|s) \sum_{a'} z_\theta(s,a'), \end{aligned} \tag{3.6}$$

where

$$z_\theta(s,a) := \pi_\theta(a|s) \frac{\partial Q_\theta(s,a)}{\partial \theta}$$

for any $s \in S, a \in A$.

Since $\gamma \in (0,1)$, it can be shown that for any θ, there exists a unique solution $\frac{\partial Q_\theta(s,a)}{\partial \theta}$ to (3.6). Therefore, there is also a unique solution $\frac{\partial \pi_\theta(a|s)}{\partial \theta}$ to (3.6). With (3.6), we can write the gradient of L_D with respect to θ as

$$\frac{\partial L_D}{\partial \theta} = \frac{\partial}{\partial \theta} \sum_{i=1}^{N} \sum_{l=0}^{n_i} \log \pi_\theta(s_{i,l}, a_{i,l}) = \sum_{i=1}^{N} \sum_{l=0}^{n_i} \Big(\frac{\partial Q_\theta(s_{i,l}, a_{i,l})}{\partial \theta} - \sum_{a'} z_\theta(s_{i,l}, a') \Big). \tag{3.7}$$

For problems with finite states, finite actions and known transition functions, we can use dynamic programming to solve the Q function given any policy π. For problems with continuous state spaces, there are approximate algorithms to estimate Q using neural networks, such as the soft Q-learning algorithm [67] or the soft actor-critic algorithm [68].

Although MEDIRL can construct reward features automatically from demonstrations, it suffers from a fundamental limitation that the learned reward function is Markovian. As a result, the learned policy has to be independent from the history, which does not suffice for tasks that are composed of multiple subtasks. Moreover, given some a priori knowledge of the task structure, MEDIRL cannot effectively take advantage of such information.

3.3. Task-Oriented Deep Inverse Reinforcement Learning

In this section, we introduce a new IRL algorithm called task-oriented deep IRL (TODIRL), which explicitly incorporates high-level task information and thus leads to more reliable generalization performance to new environments.

3.3.1. Extending the State Space Using Task Information

In this work, we represent the high-level task information as a deterministic finite automaton (DFA). A DFA $\mathcal{A}$ is defined as a tuple $\langle Q_\mathcal{A}, \Sigma, \delta, q_0, F \rangle$ where $Q_\mathcal{A}$ is a set of states; Σ is a set of input symbols (also called the alphabet); $\delta : Q_\mathcal{A} \times \Sigma \to Q_\mathcal{A}$ is a deterministic transition function; $q_0 \in Q_\mathcal{A}$ is the initial state; $F \subseteq Q_\mathcal{A}$ is a set of final states (also called *accepting* states). Given a finite sequence of input symbols $w = \sigma_0, \sigma_1, \ldots, \sigma_{k-1}$ in Σ^k for some $k \in \mathbb{N}^+$, the DFA $\mathcal{A}$ generates a unique sequence of $k+1$ states $\tau_\mathcal{A} = q_0, q_1, \ldots, q_k$ in $Q_\mathcal{A}^{k+1}$ such that for each $t = 1, \ldots, k$,

$q_t = \delta(q_{t-1}, \sigma_{t-1})$. We denote the last state q_k by taking the sequence w of inputs from q_0 as $\underline{\delta}(q_0, w)$. $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if and only if $\underline{\delta}(q_0, w) \in F$. Let $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^*$ be the set of finite sequences of input symbols that are accepted by $\mathcal{A}$, which is also referred to as the *language* of $\mathcal{A}$.

To effectively take advantage of the known task information, we introduce a *task DFA* $\mathcal{A}$ and a labeling function η to encode the known task information. The labeling function $\eta : S \to \Sigma$ is a mapping from the states of the MDP to the input symbols of the DFA, so the transitions in MDP will automatically trigger transitions in the DFA. The DFA state is always initialized to q_0. Assume that the agent takes a sequence of actions $a_0, a_1, \ldots, a_k \in A^{k+1}$ from a given initial state $s_0 \in S$ in $\mathcal{M}$. At each step $t \in \{0, \ldots, k\}$, the agent takes an action a_t from MDP state s_t and DFA state q_t. Then the MDP state transits to s_{t+1} with probability $T(s_{t+1} \mid s_t, a_t)$ and the DFA state simultaneously transits to $q_{t+1} = \delta(q_t, \eta(s_t))$. The derived trajectory $\tau = s_0, a_0, \ldots, s_t, a_t, s_{t+1}$ implements the task successfully if and only if $q_{t+1} \in F$. In essence, the states in $Q_{\mathcal{A}}$ are the memory states that track the current progress in task implementation. The input symbols in Σ are the task-critical signals triggered by the states in S.

We propose an algorithm called *task-oriented deep IRL* which is shown in Algorithm 1. The key idea is to first build a task DFA $\mathcal{A}$ using the known task information and then learn a reward function over the *extended* state space $S \times Q_{\mathcal{A}}$, rather than the original state space S of the MDP $\mathcal{M}$. As a result, the reward depends on both the current state s in $\mathcal{M}$ and the memory state in $\mathcal{A}$, as well as the action $a \in A$. Memory states can be considered as different stages in task implementation. The agent learns a different reward function and thus derives a different policy at each stage.

3.3.2. Evaluating Task Performance Using the Task DFA

Besides constructing the memory space for rewards and policies, the task DFA can also be used to evaluate the learned policies with respect to their task performance. The task performance of a policy π is evaluated via a function $y : S \times Q_{\mathcal{A}} \to [0, 1]$, where for each $(s, q) \in S \times Q_{\mathcal{A}}$, $y(s, q)$ is the probability to reach $S \times F$ from (s, q). In other words, states in $S \times F$ are treated as absorbing states. For problems with finite states spaces and action spaces, y can be represented as a vector of length

Algorithm 1 Task-oriented Inverse Reinforcement Learning

Input: A reward-free MDP $\mathcal{M} = \langle S, A, T, \rho, \gamma \rangle$, a labeling function $\eta : S \to \Sigma$, a DFA $\mathcal{A} = \langle Q_{\mathcal{A}}, \Sigma, \delta, q_0, F \rangle$, a set of demonstrations $D = \{\tau_1, \ldots, \tau_N\}$.
Output: A reward network $R_\theta : S \times Q_{\mathcal{A}} \to \mathbb{R}$ and a policy $\pi_\theta : S \times Q_{\mathcal{A}} \to \mathcal{D}(A)$.
Initialize the reward network parameter θ_0.
for each iteration t **do**
    Compute the Q function Q_{θ_t} and the policy π_{θ_t} for the current θ_t via (3.4) and (3.5).
    Compute $\frac{\partial Q_\theta}{\partial \theta} \mid_{\theta=\theta_t}$ and $\frac{\partial \pi_\theta}{\partial \theta} \mid_{\theta=\theta_t}$ via (3.6).
    Compute $\frac{\partial L_D(\theta_t)}{\partial \theta_t}$ via (3.7) and then compute $\frac{\partial L(\theta)}{\partial \theta} \mid_{\theta=\theta_t}$.
    Update θ: $\theta_{t+1} \leftarrow \theta_t + \alpha_t \frac{\partial L(\theta)}{\partial \theta} \mid_{\theta=\theta_t}$.
end for

$|S||Q_{\mathcal{A}}|$ which satisfies the following linear equation:

$$y_\pi(s,q) = \begin{cases} 1 & \text{if } q \in F, \\ 0 & \text{if } S \times F \text{ is not reachable from } (s,q), \\ \sum_{a \in A} \pi(a|(s,q)) T(s'|s,a) y_\pi(s', \delta(q, \eta(s'))) & \text{otherwise,} \end{cases} \tag{3.8}$$

where we use $y(s,q)$ to denote the component of y that is corresponding to (s,q). It has been shown (see Theorem 10.19 in [13]) that there always exists a unique solution y to (3.8). For ease of visualization, we define a scalar task performance criterion

$$L_\varphi = \sum_{s \in S} \rho(s) y_\pi(s, q_0), \tag{3.9}$$

which is the average probability to implement the task in $\mathcal{A}$ over all initial states by taking policy π. L_φ is used to evaluate the task performance of the learned policy in new environments in Section 4.5.

3.4. Related Work

Recently, there has been interesting works on LfD with task information. The first attempt to incorporate task evaluation into IRL was to augment the demonstrations with evaluation of their task performance. Lee et al. [102] proposed an IRL algorithm that learns from both successful

(positive) demonstrations and failed (negative) demonstrations. El Asri et al. [51] and Burchfiel et al. [30] augmented each demonstration trajectory with a score rated by experts. The boolean labels and the continuous scores can be used to train a classification or a regression model to evaluate policies. Their experiment results showed that such data augmentation help reduce the number of demonstration. But since the task is not explicitly defined, the learned policy evaluation model may be neither reliable nor interpretable. Pan and Shen [134] assumed that the experts provide with a set of subgoal states for each demonstration. However, the learning agent does not understand how or why the demonstrator picks this set of critical subgoal states, especially if the number of subgoals are inconsistent over different demonstrations. Though the robot may recognize some similar states using the learned reward features in a new environment, it cannot tell if all of previous subgoals are still necessary or if they should be executed in the same order. With our method, the agent can search for a sequence of subgoals in the extended state space $S \times Q_{\mathcal{A}}$ that implements the task, which may not be necessarily the same as shown in training environments.

Several work has been done on policy learning with assumptions about the task structure. Niekum et al.[130] and Michini et al. [120] use Bayesian inference to segment unstructured demonstration trajectories. Shiarlis et al. [158] assumed that the expert performs a given sequence of (symbolic) subtasks in each demonstration. They solve the problem of temporal alignment for the demonstrations and policy learning for each subtask simultaneously. Kipf et al. [87] solved a similar problem using recurrent neural networks, where each latent node corresponds to a subtask. The outputs are policies for each subtask, which are not expected to generalize to new environments with constraints (for example, obstacle states). For example, a policy that successfully navigates to a target object in one environment may lead to collision in another environment, as the location of obstacles have changed. Re-planning is usually necessary to deal with this problem. With our method, we can easily adapt to new environments by solving a new policy with the learned reward function.

Perhaps the most closely related work to ours is that of Wen et al. [186], which also discussed about the idea of using high-level task information in IRL. However, their method as is restricted to linearly parameterized rewards and there was no discussion about the reward generalization performance in

new environments.

3.5. Experimental Results

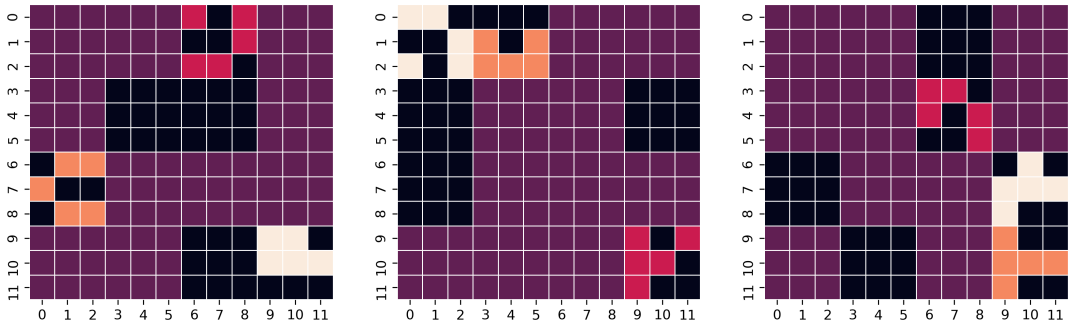
Model. We use the game "temporal grid-world" for numerical experiments. Temporal grid-world is an MDP in which the underlying task has a temporal structure, i.e., knowing the current state of the MDP is not enough to determine the next best action. The transitions in the MDP are deterministic, and the set of actions is {"up", "down", "left", "right"}.

Each cell in the MDP has one of the colors: {red, yellow, white, purple, black} and belongs to one of the following categories: {important object, obstacle, distractor object}. The category of each cell is defined based on the color of all the cells in the 3×3 neighborhood of that cell. We define the categories as: *Important object 1 (O1)*: 5 red cells and 4 black cells, *Important object 2 (O2)*: 5 yellow cells and 4 black cells, *Important object 3 (O3)*: 5 white cells and 4 black cells, *Obstacle (OB)*: 9 black cells and *Distractor object (DB)*: any cell that does not belong to one of the other categories.

Fig. 5a shows the training grid-world used for all experiments. To refer to a cell we use a tuple with the following structure: (vertical index, horizontal index). As an example, in Fig. 5a, *O1* is located at (1,7). At test time, the placement of the objects and obstacles is randomized. Fig. 5b and Fig. 5c depict two test grid-worlds.

Task Specification and DFA. The task specification is as follows: {Reach *O1*, *O2* and *O3* in this order and never reach *OB*}. To encode the task specification, a DFA is constructed with 5 states as described in Table 3. The DFA is visualized in Fig. 4 in the appendix. The states of the DFA act as memory states and capture the agent's progress toward task completion.

Reward network. When performing TODIRL, the reward network is modeled as a multilayer perceptron (MLP) with 2 hidden layers; each layer has 80 neurons and is followed by ReLU non-linearity. The input to the network at each state is composed of the colors of the 3×3 neighborhood of the agent in the MDP, the current DFA state and the current action. The network outputs a single



(a) Training grid-world (b) Test grid-world with ID 1 (c) Test grid-world with ID 2

Figure 5: Training and test grid-worlds. Indexing convention: (vertical axis index, horizontal axis index). In (a) $O1 = (1, 7)$, $O2 = (7, 1)$, $O3 = (10, 10)$ and $OB = \{(4, 4), (4, 5), (4, 6), (4, 7), (10, 7)\}$. All other cells correspond to distractor objects.

Table 3: DFA states

State	Interpretation
q_0	None of *O1, O2, O3 or OB* has been reached.
q_1	*O1* has been reached. *O2* or *O3* has never been reached.
q_2	*O1* and *O2* have been reached in this order. *O3* has never been reached.
q_3	Winning state. *O1* and *O2* and *O3* have been reached in this order. This state is absorbing.
q_f	Failure state. This state is absorbing.

number as the reward. We use full gradient descent for training the reward network, i.e, at each iteration, we use all the demonstrations to calculate $\frac{\partial L(\theta)}{\partial \theta}$ according to Eq. 3.7.

Demonstration trajectories. To produce the demonstration trajectories, we manually designed a ground-truth reward function over the extended state space $S \times Q_{\mathcal{A}}$ and run MaxEnt RL [202] on the extended state space. The MaxEnt RL algorithm yields a softmax policy which we use to sample demonstration trajectories.

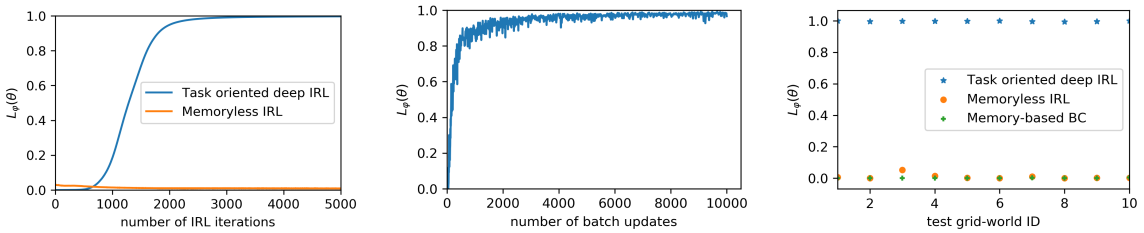
Baselines. We implemented two baseline models to compare with TODIRL; *"memoryless IRL agent"* and *"memory-based behavioural cloning (BC) agent"*. The memoryless IRL agent is a basic MaxEnt IRL agent [193] that does not benefit from the extended state space and relies solely on the states of the MDP. The memory-based BC agent operates on the extended state space and uses the behavioural cloning [143] method to learn a policy that mimics the demonstrations. To train this agent, we created a training dataset with elements (X, Y), where X denotes the set of all pairs

$(s, q) \in S \times Q_{\mathcal{A}}$ observed in the demonstrations and Y denotes the corresponding demonstration actions. We modeled the policy using a deep convolutional neural network. The network has two convolutional layers, the first layer has 12 kernels and the second layer has 24 kernels, all kernels are of size (2×2) with stride of 1. The convolutinal layers are followed by 2 fully connected layers with 100 neurons each and the output layer has 4 neurons corresponding to the score of each action. After each hidden layer, ReLU non-linearity is used. The input to the policy network is $x = (s, q)$, and the output would be a probability distribution over actions. Let P_x be the probability distribution predicted over actions for x, C corresponds to the index of the action in demonstrations and A is the set of all action indexes. Then the objective function that is minimized is defined as

$$L_{BC}(\theta) = -\sum_{x \in X} \log \left(\frac{\exp\left(P_x[C]\right)}{\sum_{i \in A} \exp\left(P_x[i]\right)} \right). \tag{3.10}$$

We train the network using ADAM optimization [86] with mini batches of size 128. In all the experiments we use a learning rate of 0.003.

TODIRL vs baselines. The primary criterion we use for evaluating the performance of a trained agent is the value of L_{φ}. All models are trained using the same single training grid-world (Fig. 5a) and same demonstrations. The TODIRL agent and the memory-based BC agent perform well at training time (Fig. 6a and Fig. 6b) with TODIRL agent still outperforming the BC agent. The memoryless IRL agent, however, performs poorly even during training (Fig. 6a). The main difference between TODIRL and memory-based BC lies in their generalization ability. Fig. 6c shows how the three agents compare in terms of their generalizability to the test grid-worlds. As evident from this figure, the memory-based IRL agent performs very well on test cases with $L_{\varphi} \approx 1$, whereas the other two methods generalize poorly. The reason for this difference in generalizability is that the memory-based IRL agent learns a local reward function which generalizes significantly better to test grid-worlds where the same objects and obstacles are placed randomly. The behavioral cloning agent, on the other hand, learns to shallowly imitate the demonstrations by directly learning a policy. The memoryless IRL agent performs poorly at both training and test time as it essentially learns a single reward function based on MDP states and ignores the temporal structure of the underlying task. For



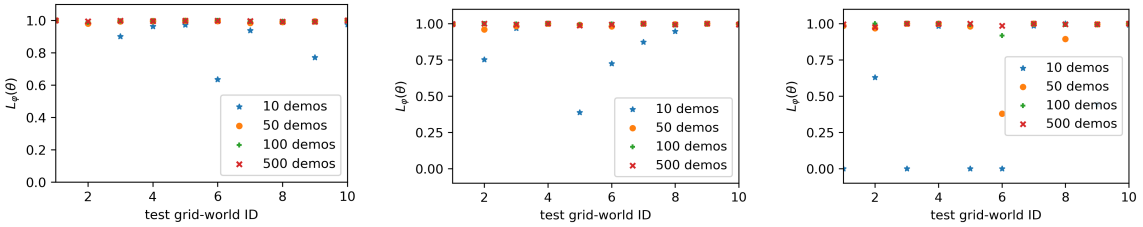
(a) Training IRL agents (b) Training memory-based BC agent (c) Test performance comparison

Figure 6: Training and testing performance of TODIRL and the baselines with 100 demonstrations.

further visualization and analysis of test performance of TODIRL, refer to Sec. 7.2 in the appendix.

Incomplete DFA. We also trained an agent that has access to incomplete task specification. What is missing from the specification is the statement "never reach an *OB*". We equipped this agent with a corresponding DFA (Fig. 5 in the appendix) that does not transition when an *OB* is reached. Fig. 7b shows the performance of such an agent at test time as a function of the number of demonstrations. Fig. 7a corresponds to the case where a complete DFA is used. A comparison between these two cases shows that the performance of the agent with incomplete DFA declines, specially with decreasing number of demonstrations. Note that this specific choice of incomplete DFA does not remove any of the temporal information, the only information that is missing is the fact that reaching obstacles leads to failure, but since this fact is true no matter what the DFA state is, the agent can still learn the task from demonstrations without help from the DFA, however, only if the number of demonstrations is large enough.

Reward input size. Fig 7c shows the generalization of the reward function when we assume the input is a neighborhood of size 5×5, while the ground truth reward is a function of the 3×3 neighborhood. For large number of demonstrations (> 100), both cases generalize well. For smaller number of demonstrations, however, the case corresponding to 3×3 input generalizes better. This observation verifies our hypothesis that the proposed method benefits from modelling the reward locally. The 5×5 neighborhood includes all the information available in the 3×3 neighborhood but the extra information is redundant for our problem and hence leads to weaker generalization



(a) Complete DFA

(b) Incomplete DFA

(c) Complete DFA, 5×5 input to reward network

Figure 7: Testing performance with 100 demonstrations for different design choices.

with limited amount of training as it interferes with learning the most relevant features; this is an interesting observation, and this result could be extended to other problems where the reward function is a function of local observations rather than the function of the global state.

Chapter 4: Constrained Cross-Entropy Method for Safe Reinforcement Learning

4.1. Introduction

We study the following constrained optimal control problem in this chapter: Given a dynamical system model with continuous states and actions, a objective function and a constraint function, find a controller that maximizes the objective function while satisfying the constraint. Although this topic has been studied for decades within the control community [20], it is still challenging for practical problems. To illustrate some major difficulties, consider the synthesis of a policy for a nonholonomic mobile robot to reach a goal while avoiding obstacles (which introduces constraints) in a cost-efficient way (which induces an objective). The obstacle-free state space is usually nonconvex. The equations of the dynamical system model are typically highly nonlinear. Constraint functions and cost functions may not be convex or differentiable in the state and action variables. There may even be hidden variables that are not observable and make transitions and costs non-Markovian. Given all these difficulties, we still need to compute a policy that is at least feasible and improve the cost objective as much as possible.

Reinforcement learning (RL) methods have been widely used to learn optimal policies for agents with complicated or even unknown dynamics. For problems with continuous state and action spaces, the agent's policy is usually modeled as a parameterized function of states such as deep neural networks and later trained using policy gradient methods [65, 123, 153, 154, 155, 159, 188]. By encoding control tasks as reward or cost functions, RL has successfully solved a wide range of tasks such as Atari games [121, 122], the game of Go [160, 161], controlling simulated robots [144, 197] and real robots [106, 125, 198].

Most of the existing methods for RL solve only unconstrained problems. However, it is generally non-trivial to transform a constrained optimal control problem into an unconstrained one, due to the asymmetry between the goals of objective optimization and constraint satisfaction. On the one

hand, it is usually acceptable to output a policy that is only locally optimal with respect to the optimization objective. On the other hand, in many application scenarios where constraints encode safety requirements or the amount of available resources, violating the constraint even by a small amount may have significant consequences.

Existing methods for safe reinforcement learning that are based on policy gradient methods cannot guarantee strict feasibility of the policies they output, even when initialized with feasible initial policies. When initialized with an infeasible policy, they usually are not be able to find even a single feasible policy until their convergence (with an example in Section 4.5). These limitations motivate the following question: Can we develop a reinforcement learning algorithm that explicitly addresses the priority of constraint satisfaction? Rather than assuming that the initial policy is feasible and that one can always find a feasible policy in the estimated gradient direction, we need to deal with cases in which the initial policy is not feasible, or we have never seen a feasible policy before.

Inspired by stochastic optimization methods based on the cross-entropy (CE) concept [75], we propose a new safe reinforcement learning algorithm, which we call the *constrained cross-entropy (CCE)* method. The basic framework is the same with standard CE methods: In each iteration, we sample from a distribution of policies, select a set of elite sample policies and use them to update the policy distribution. Rather than treating the constraints as an extra term in the objective function as what policy gradient method do, we use constraint values to sort sample policies. If there are not enough feasible sample policies, we select only those with the best constraint performance as elite sample policies. If a given proportion of the sample policies are feasible, we select the feasible sample policies with the best objective values as elite sample policies. Instead of initializing the optimization with a feasible policy, the method improves both the objective function and the constraint function with the constraint as a prioritized concern.

Our algorithm can be used as a black-box optimizer. It does not even assume that there is an underlying reward or cost function encoding the optimization objective and constraint functions. In fact, the algorithm can be applied to any finite-horizon problem (say, with horizon N) whose objective and constraint functions are defined as the average performance over some distribution of

trajectories. For example, a constraint function can be the probability that the agent satisfies a given task specification (which may be Markovian or non-Markovian) with policy π_θ, if the satisfaction of the given task can be decided with any N-step trajectory. An optimization objective may be the expected number of steps before the agent reaches a goal state, or the expected maximum distance the agent has left from its origin, or the expected minimum distance between the agent and any obstacle over the whole trajectory.

Our contributions are as follows. First, we present a model-free constrained RL algorithm that works with continuous state and action spaces. Second, we prove that the asymptotic behavior of our algorithm can be almost-surely described by that of an ordinary differential equation (ODE), which is easily interpretable with respect to the objectives. Third, we give sufficient conditions on the properties of this ODE to guarantee the convergence of our algorithm. At last, we empirically show that our algorithm converges to the global optimum with high probability in a convex problem, and effectively find feasible policies in a 2D navigation example while other policy-gradient-based algorithms fail to find strictly feasible solutions.

4.2. Related Work

Safety has long been concerned in RL literature and is formulated as various criteria [62]. We choose to take the so-called *constrained criterion* [62] to encode our safety requirement, which is the same as in the literature of constrained Markov decision processes (CMDP) [8]. Approaches are still limited for safe RL with continuous state and action spaces. Uchibe and Doya [177] proposed a constrained policy gradient reinforcement learning algorithm, which relies on projected gradients to maintain feasibility. The computation of projection restricts the types of constraints it can deal with, and there is no known guarantee on convergence. Chow et al. [42] came up with a trajectory-based primal-dual subgradient algorithm for a risk-constrained RL problem with finite state and action spaces. The algorithm is proved to converge almost-surely to a local saddle point. However, the constraints are just implicitly considered by updating dual variables and the output policy may not actually satisfy the constraints. Recently, Achiam et al. [4] proposed a trust region method for CMDP called constrained policy optimization (CPO), which can deal with high-dimensional policy classes

such as neural networks and claim to maintain feasibility if started with a feasible solution. However, we found in Section 4.5 that feasibility is rarely guaranteed during learning in practice, possibly due to errors in gradient and Hessian matrix estimation.

Cross-entropy-based stochastic optimization techniques have been applied to a series of RL and optimal control problems. Mannor, Rubinstein and Gat [117] used cross-entropy methods to solve a stochastic shortest-path problem on finite Markov decision processes, which is essentially an unconstrained problem. Szita and Lörincz [171] took a noisy variant to learn how to play Tetris. Kobilarov [90] introduced a similar technique to motion planning in constrained continuous-state environments by considering distributions over collision-free trajectories. Livingston, Wolff and Murray [113] generalized this method to deal with a broader class of trajectory-based constraints called linear temporal logic specifications. Both methods simply discard all sample trajectories that violate the given constraints, and thus their work can be considered as a special case of our work when the constraint function has binary outputs. Similar applications in approximate optimal control with constraints can be found in [60, 107, 135].

4.3. Preliminaries

We first introduce some notations that are used throughout this chapter. For a set B, let $\mathcal{D}(B)$ be the set of all probability distributions over B, $int(B)$ be the interior of B and $\mathbf{1}_B$ be the indicator function of B. For any $k \in \mathbb{N}^+$, define

$$B^k = \{s_0, s_1, \ldots, s_{k-1} \mid s_t \in B,\ \forall t = 0, \ldots, k-1\}$$

as the set of all sequences composed by elements in B of length k. We further define

$$B^* \bigcup_{1 \leq k < \infty} B^k$$

as the set of all (non-empty) finite sequences generated by elements in B. Given two integers $i, j \in \mathbb{N}$ such that $i \leq j$, we use $i : j$ to denote the sequence $i, i+1, \ldots, j-1, j$.

A (reward-free) *Markov decision process (MDP)* is defined as a tuple $M = \langle S, A, T, P_0 \rangle$, where S is a set of states, A is a set of actions, $T : S \times A \to \mathcal{D}(S)$ is a transition distribution function and $P_0 \in \mathcal{D}(S)$ is an initial state distribution. Without loss of generality, we assume that the set of available actions are the same at all states. S and A can either be continuous or discrete.

Given an MDP M, a *policy* $\pi : S^* \to \mathcal{D}(A)$ is a mapping from a sequence of history states to a distribution over actions. π is called *stationary* or *memoryless* if its output is decided by the last state in history, that is, $\pi(\zeta) = \pi(\zeta_k)$ holds for any $\zeta = \zeta_0, \zeta_1, \ldots, \zeta_k \in S^*$. π is called *deterministic* if the support of its output distribution is always a singleton. For notational simplicity, we use $\pi(\zeta)$ to represent the unique action $a \in A$ such that $\pi(a|\zeta) > 0$ for any $\zeta \in S^*$. If π is not deterministic, we call it a *randomized* policy. Let Π, Π_S, Π_D, Π_{SD} be the set of all policies, stationary policies, deterministic policies and stationary deterministic policies for M. It is clear that $\Pi_{SD} = \Pi_S \bigcap \Pi_D \subset \Pi$.

Given a finite horizon $N \in \mathbb{N}^+$, an *N-step trajectory* τ is a sequence of N state-action pairs: $\tau = s_0, a_0, \ldots, s_{N-1}, a_{N-1} \in (S \times A)^N$. Each policy $\pi \in \Pi$ decides a distribution $P_{\pi,N}$ over N-step trajectories such that for any $\tau = s_0, a_0, \ldots, s_{N-1}, a_{N-1}$,

$$P_{\pi,N}(\tau) = P_0(s_0) \prod_{t=0}^{N-2} T(s_{t+1}|s_t, a_t) \prod_{t=0}^{N-1} \pi(a_t|s_{0:t}).$$

Without loss of generality, we assume that N is fixed and use P_π to represent $P_{\pi,N}$.

To solve an N-step planning problem, we can generally define a trajectory-based *objective* function $J : (S \times A)^N \to \mathbb{R}$ as a mapping from each N-step trajectory to a scalar value. For each $\pi \in \Pi$, let

$$G_J(\pi) = \mathbb{E}_{\tau \sim P_\pi}[J(\tau)]$$

be the expected value of J with the N-step trajectory distribution decided by π. Many commonly used objectives for finite-horizon planning problems can be represented as G_J, such as

- Expected N-step total reward. Given a reward function $R : S \times A \to \mathbb{R}$, define

$$J(\tau) = \sum_{t=0}^{N-1} R(s_t, a_t),\ \tau = s_0, a_0, \ldots, s_{N-1}, a_{N-1}.$$

  $G_J(\pi) = \mathbb{E}_{\tau \sim \rho_\pi}[J(\tau)]$ is the expected N-step total reward while running π.

- Probability. Given a set of N-step trajectories $B \subseteq (S \times A)^N$, define $J(\tau) = \mathbf{1}_B(\tau)$ and $G_J(\pi)$ will be the probability to induce a trajectory in B while running π. For example, $G_J(\pi)$ can be used to represent the probability to reach a set of target states or the probability to remain in a safe region for N steps.

A policy $\pi^* \in \Pi$ is *optimal* with respect to J if

$$G_J(\pi^*) = \max_{\pi' \in \Pi} G_J(\pi').$$

Generally, π^* is not stationary if the horizon N is finite. But since the transition distribution $\mathcal{T}$ is Markovian, there always exists a (non-stationary) deterministic optimal policy, which is formally stated in Lemma 1.

Lemma 1. *Given $N \in \mathbb{N}^+$ be a finite horizon and an MDP M, let $J : (S \times A)^N \to \mathbb{R}$ be any trajectory-based functional. There always exists a deterministic (yet possibly non-stationary) optimal policy π^*. In other words, there exists $\pi_d \in \Pi_D$ such that*

$$G_J(\pi_d) = \max_{\pi \in \Pi} G_J(\pi).$$

Proof. Let $\pi : S^* \to \mathcal{D}(A)$ be a policy for M. Then it generates a distribution over N-step trajectories which is P_π. For any $t = 0, \ldots, N-1$, the probability that $s_0, a_0, \ldots, s_t$ (denoted as

$s_{0:t}, a_{0:t-1}$) is a prefix of a generated trajectory is

$$\begin{aligned} p_t^{pre} =& P_\pi(s_{0:t}, a_{0:t-1}) = \sum_{a'_{t:N-1}, s'_{t+1:N-1}} P_\pi(s_{0:t}, a_{0:t-1}, a'_{t:N-1}, s'_{t+1:N-1}) \\ =& P_0(s_t) \prod_{t'=0}^{t-1} \Big(\pi(a_{t'} \mid s_{0:t'}) T(s_{t'+1} \mid s_{t'}, a_{t'}) \Big). \end{aligned}$$

Given a prefix $s_{0:t}, a_{0:t}$, the probability that the next $(N-t-1)$ state-action pairs are $s_{t+1}, a_{t+1}, \ldots, s_{N-1}, a_{N-1}$ (denoted as $s_{t+1:N-1}, a_{t+1:N-1}$) is

$$p_{t+1}^{suf} = P_\pi^{suf}\big(s_{t+1:N-1}, a_{t+1:N-1} \mid s_{0:t}, a_{0:t}\big) = \prod_{t'=t}^{N-2} T(s_{t'+1} \mid s_{t'}, a_{t'}) \prod_{t'=t+1}^{N-1} \pi(a_{t'} \mid s_{0:t'}).$$

Define $J_N = J(s_{0:N-1}, a_{0:N-1})$. We can rewrite $G_J(\pi)$ as

$$\begin{aligned} G_J(\pi) =& \sum_{s_{0:N-1}, a_{0:N-1}} \pi(a_t \mid s_{0:t}) p_t^{pre} p_{t+1}^{suf} J_N \\ =& \sum_{s_{0:t}} \pi(a_t \mid s_{0:t}) \sum_{a_{0:t-1}} p_t^{pre} \sum_{s_{t+1:N-1}, a_{t+1:N-1}} p_{t+1}^{suf} J_N. \end{aligned}$$

Define

$$Q_\pi(a_t \mid s_{0:t}) = \sum_{a_{0:t-1}} p_t^{pre} \sum_{s_{t+1:N-1}, a_{t+1:N-1}} p_{t+1}^{suf} J_N,$$

then

$$G_J(\pi) = \sum_{s_{0:t}} \pi(a_t \mid s_{0:t}) Q_\pi(a_t \mid s_{0:t}).$$

Note that p_t^{pre}, p_{t+1}^{suf} and J_N are independent of $\pi(a_t|s_{0:t})$; $\pi(a_t \mid s_{0:t})$ is also independent for different t and prefix $s_{0:t}$. Therefore $Q_\pi(a \mid s_{0:t})$ is independent of $\pi(a \mid s_{0:t})$. For any prefix except $s_{0:t}$, it holds for any optimal policy $\pi'(\cdot \mid s_{0:t})$ that maximizes G_J that

$$\{a \in A \mid \pi'(a \mid s_{0:t}) > 0\} \subseteq \arg\max_{a \in A} Q_\pi(a \mid s_{0:t})$$

which always incorporates a deterministic choice. In other words, randomized policies cannot reach

higher G_J than deterministic policies. $\square$

Similarly, we can define a trajectory-based *cost* function $Z : (S \times A)^N \to \mathbb{R}$ and define

$$H_Z(\pi) = \mathbb{E}_{\tau \sim P_\pi}[Z(\tau)]$$

as the expected cost over trajectory distribution P_π. A policy $\pi \in \Pi$ is *feasible* for a constrained optimization problem with cost function Z and *constraint upper bound* d if $H_Z(\pi) \leq d$. Let $\Pi_{Z,d}$ be the set of all feasible policies.

For notational simplicity, we omit J and Z in G_J and H_Z whenever there is no ambiguity. For any policy $\pi \in \Pi$, we refer to $G(\pi)$ and $H(\pi)$ as the G-*value* and H-*value* of π.

4.4. Constrained Cross-Entropy Framework

In this section, we first state the constrained policy optimization given a trajectory-based objective function J and a trajectory-based cost function Z, then we describe how to transform the constrained problem into an unconstrained one with a surrogate objective function. We propose an algorithm called constrained cross-entropy method to optimize the surrogate objective and show that the algorithm converges almost surely with some given assumptions.

4.4.1. Problem Formulation

We consider a finite-horizon RL problem with a strictly positive objective function $J : (S \times A)^N \to \mathbb{R}^+$, a cost function $Z : (S \times A)^N \to \mathbb{R}$ and a constraint upper bound d. For MDPs with continuous state and action spaces, it is usually intractable to exactly solve an optimal stationary policy due to the curse of dimensionality. An alternative is to use function approximators, such as neural networks, to parameterize a subset of policies. Given a parameterized class of policies Π_Θ with a parameter space $\Theta \subseteq \mathbb{R}^{d_\theta}$, we aim to solve the following problem:

$$\pi^* = \underset{\pi \in \Pi_\Theta \bigcap \Pi_{Z,d}}{\arg\max} [G_J(\pi)]. \tag{4.1}$$

The proposed algorithm, which we call the *constrained cross-entropy* method, generalizes the well-known cross-entropy method [117] for unconstrained optimization. The basic idea is to generate a sequence of policy distributions that eventually concentrates on a feasible (locally) optimal policy. Given a distribution over Π_Θ, we randomly generate a set of sample policies, sort them with a ranking function that depends on their G-values and H-values and then update the policy distribution with a subset of highly ranked sample policies. The set of sample policies that are selected to update the current policy distribution are also referred to as *elite samples* or *elite set* in the literature (for example, [90, 113, 117]).

Given the policy parameterization Π_Θ, we use distributions over the parameter space Θ to represent distributions over the policy space Π_Θ. Let $f : \mathcal{V} \to \mathcal{D}(\Theta)$ be a family of distributions over Θ with parameter space $\mathcal{V}$. For each $\boldsymbol{v} \in \mathcal{V}$, $f_{\boldsymbol{v}}(\cdot)$ is a distribution over policies in Π_Θ. We assume that for any $\theta \in \Theta$, there exists $\boldsymbol{v}_\theta \in \mathcal{V}$ such that $f_{\boldsymbol{v}_\theta}(\theta') = \mathbf{1}_{\{\theta\}}(\theta')$. In other words, $f_{\boldsymbol{v}_\theta}$ is a discrete distribution that is concentrated at θ. Given $\mathcal{V}$ and f, we rewrite the original problem (4.1) where we search over policies into the following problem which searches over policy distributions:

$$\boldsymbol{v}^* = \arg\max_{\boldsymbol{v} \in \mathcal{V}} \mathbb{E}_{\theta \sim f_{\boldsymbol{v}}}[G_J(\pi_\theta) \mid \pi_\theta \in \Pi_{Z,d}]. \tag{4.2}$$

We show the connection between (4.1) and (4.2) with Lemma 2.

Lemma 2. *Let π_{θ^*} and v^* be any solution to* (4.1) *and* (4.2) *respectively. Then*

$$G_J(\pi_{\theta^*}) = \mathbb{E}_{\theta \sim f_{\boldsymbol{v}^*}}[G_J(\pi_\theta) \mid \pi_\theta \in \Pi_{Z,d}].$$

Proof. If π_{θ^*} is a solution to (4.1), then $\pi_{\theta^*} \in \Pi_{Z,d}$ and $G_J(\pi_{\theta^*}) \geq G_J(\pi_\theta)$ for all $\pi_\theta \in \Pi_{Z,d}$. Therefore,

$$\begin{aligned} G_J(\pi_{\theta^*}) &= \mathbb{E}_{\theta \sim f_{\boldsymbol{v}_{\theta^*}}}[G_J(\pi_\theta) \mid \pi_\theta \in \Pi_{Z,d}] \\ &\leq \mathbb{E}_{\theta \sim f_{\boldsymbol{v}^*}}[G_J(\pi_\theta) \mid \pi_\theta \in \Pi_{Z,d}] \leq \quad \mathbb{E}_{\theta \sim f_{\boldsymbol{v}^*}}[G_J(\pi_{\theta^*})] = G_J(\pi_{\theta^*}), \end{aligned}$$

where the first inequality holds since $\boldsymbol{v}^*$ is a solution to (4.2). □

4.4.2. Surrogate Objective

As with other CE-based algorithms, we replace the objective in (4.2) with a surrogate function. For the unconstrained CE method, the surrogate function is the conditional expectation of G_J over the elite sample policies with the current sampling distribution $f_{\boldsymbol{v}}$. The ranking function for unconstrained CE is defined using the concept of ρ-quantiles for random variables, which is formally defined as below.

Definition 1. *[74] Given a distribution $P \in \mathcal{D}(\mathbb{R})$, $\rho \in (0,1)$ and a random variable $X \sim P$, the ρ-quantile of X is defined as a scalar γ such that $Pr(X \leq \gamma) \geq \rho$ and $Pr(X \geq \gamma) \geq 1-\rho$.*

For $\rho \in (0,1)$, $\boldsymbol{v} \in \mathcal{V}$ and any function $X : \Theta \to \mathbb{R}$, we denote the ρ-quantile of X for $\theta \sim f_{\boldsymbol{v}}$ by $\xi_X(\rho, \boldsymbol{v})$. Let

$$\delta : \mathbb{R} \times \{\geq, \leq, >, <, =\} \times \mathbb{R} \to \{0,1\}$$

be an indicator function such that for $\circ \in \{\geq, \leq, >, <, =\}$, $\delta(x \circ y) = 1$ if and only if $x \circ y$ holds. Usually, we interpret ρ as the proportion of highly ranked policies. For the unconstrained CE method, a policy π_θ is considered as highly ranked if $G(\pi_\theta) \geq \xi_G(1-\rho, \boldsymbol{v})$, that is, if the G-value of π_θ is greater than at least $(1-\rho)$ of all policies in Π_Θ with sampling distribution $f_{\boldsymbol{v}}$. The surrogate objective function for the unconstrained CE method is

$$\mathbb{E}_{\theta \sim f_{\boldsymbol{v}}}[G(\pi_\theta)\delta(G(\pi_\theta) \geq \xi_G(1-\rho, \boldsymbol{v}))]. \tag{4.3}$$

When there is a constraint $H(\pi) \leq d$, we also need to take the H-value of π_θ into consideration while designing ranking functions. As in the unconstrained case, we will have a ρ proportion of all policies as highly ranked policies. Let $p_{\boldsymbol{v}}$ be the probability of sampling feasible policies with $f_{\boldsymbol{v}}$. The definition of highly-ranked policies with respect to $f_{\boldsymbol{v}}$ can be split into two cases, depending whether $p_{\boldsymbol{v}} \geq \rho$ or not.

Case 1. If $p_{\boldsymbol{v}} < \rho$, the ρ-quantile of H with distribution $f_{\boldsymbol{v}}$ will be greater than the constraint threshold d. In this case, we rank policies in the decreasing order of their H-values. The indicator function of highly ranked policies is $\delta(H(\pi_\theta) \leq \xi_H(\rho, \boldsymbol{v}))$. As a result, all feasible policies and a small proportion (to be specific, $(\rho - p_{\boldsymbol{v}})\backslash(1 - p_{\boldsymbol{v}})$) of infeasible policies with the least H-values will be highly ranked.

Case 2. If $p_{\boldsymbol{v}} \geq \rho$, the probability of sampling feasible policies with $f_{\boldsymbol{v}}$ is at least ρ. In this case, we rank *feasible* policies in the increasing order of their G-values. Define $U : \Pi_\Theta \to \mathbb{R}$ such that

$$U(\pi_\theta) = G(\pi_\theta)\delta(H(\pi_\theta) \leq d).$$

The indicator function of highly ranked policies is $\delta(U(\pi_\theta) \geq \xi_U(1 - \rho, \boldsymbol{v}))$. Since G_J is strictly positive, $U(\pi) > U(\pi')$ holds for any feasible π and infeasible π'. As $p_{\boldsymbol{v}} \geq \rho$, any policy π_θ such that $U(\pi_\theta) \geq \xi_U(1 - \rho, \boldsymbol{v}))$ will be feasible. As a result, a fraction of $\rho\backslash p_{\boldsymbol{v}}$ feasible policies with the highest G-values will be highly ranked.

We can combine the two cases and write down a single indicator function of highly ranked policies with distribution $f_{\boldsymbol{v}}$. Define $S : \Pi_\Theta \times \mathcal{V} \times (0, 1) \to \{0, 1\}$ such that

$$\begin{aligned} S(\pi_\theta, \boldsymbol{v}, \rho) =& \delta(\xi_H(\rho, \boldsymbol{v}) > d)\delta(H(\pi_\theta) \leq \xi_H(\rho, \boldsymbol{v}))+ \\ & \delta(\xi_H(\rho, \boldsymbol{v}) \leq d)\delta(U(\pi_\theta) \geq \xi_U(1 - \rho, \boldsymbol{v})). \end{aligned}$$

Then the surrogate function for CCE can be expressed as follows:

$$L(\boldsymbol{v}; \rho) = \mathbb{E}_{\theta \sim f_{\boldsymbol{v}}}[G(\pi_\theta)S(\pi_\theta, \boldsymbol{v}, \rho)]. \tag{4.4}$$

Note that the surrogate function (4.4) for the constrained problem has the same structure as that for the unconstrained problem in (4.3). Intuitively, the highly-ranked policies are selected to update the current policy distribution $f_{\boldsymbol{v}}$. If $p_{\boldsymbol{v}} < \rho$, it suggests that the distribution update should be focused on increasing the probability to sample feasible policies; if $p_{\boldsymbol{v}} \geq \rho$, we can pay more attention to increasing the expected G-value over feasible policies.

Remark 1. *For the unconstrained problem, $p_{\boldsymbol{v}} = 1 > \rho$ and $U(\pi_\theta) = G(\pi_\theta)$, then*

$$\begin{aligned}&\mathbb{E}_{\theta\sim f_{\boldsymbol{v}}}[\delta(G(\pi_\theta) \geq \xi_G(1-\rho, \boldsymbol{v}))]\\ =\ &\mathbb{E}_{\theta\sim f_{\boldsymbol{v}}}[G(\pi_\theta)\delta(U(\pi_\theta) \geq \xi_U(1-\rho, \boldsymbol{v}))] = L(\boldsymbol{v};\rho).\end{aligned}$$

Therefore (4.3) *is a special case of* (4.4)*.*

Remark 2. *If $\xi_H(\rho, \boldsymbol{v}) \leq d$, then*

$$\begin{aligned}G(\pi_\theta)\delta\big(G(\pi_\theta) \geq \xi_G(1-\rho, \boldsymbol{v})\big) \geq &U(\pi_\theta)\delta\big(U(\pi_\theta) \geq \xi_U(1-\rho, \boldsymbol{v})\big)\\ \geq &G(\pi_\theta)\delta\big(H(\pi_\theta) \leq \xi_H(\rho, \boldsymbol{v})\big).\end{aligned}$$

Intuitively, if at least $100\rho\%$ of all policies are feasible, $L(\boldsymbol{v};\rho)$ is less than the objective value for the unconstrained CE method and greater than the expected G-value over the $100\rho\%$ policies of the lowest H-values.

Remark 3. *For ease of analysis, we may approximate δ by a continuous function $\tilde{\delta}_\varepsilon$ where $\varepsilon > 0$, such that for any $x, y \in \mathbb{R}$ and $\circ \in \{\geq, >\}$:*

$$\tilde{\delta}_\varepsilon(x \circ y) = \begin{cases}\delta(x \circ y) & \text{if } y \circ x \text{ or } y < x - \varepsilon\\ (y-x)/\varepsilon + 1 & \text{otherwise.}\end{cases}$$

$$\tilde{\delta}_\varepsilon(x < y) = 1 - \tilde{\delta}_\varepsilon(x \geq y),\ \tilde{\delta}_\varepsilon(x \leq y) = 1 - \tilde{\delta}_\varepsilon(x > y).$$

The main problem we solve in this chapter can be then stated as follows.

Problem 1. *Given a set $\Pi = \{\pi_\theta : \theta \in \Theta\}$ of policies with parameter space Θ, a set $F_{\mathcal{V}} = \{f_{\boldsymbol{v}} \in \mathcal{D}(\Theta) : \boldsymbol{v} \in \mathcal{V}\}$ of distributions over Θ, two functions $G : \Pi \to \mathbb{R}^+$ and $H : \Pi \to \mathbb{R}$, a constraint upper bound d and $\rho \in (0,1)$, compute $\boldsymbol{v}^* \in \mathcal{V}$ such that*

$$\boldsymbol{v}^* = \arg\max_{\boldsymbol{v}\in\mathcal{V}} L(\boldsymbol{v};\rho),$$

where $L : \mathcal{V} \times (0,1) \to \mathbb{R}$ is defined in (4.4)*.*

4.4.3. The Constrained Cross-Entropy Algorithm

In this section, we focus on how to solve Problem 1 and propose the CCE algorithm. We first describe the key idea behind the (idealized) CE-based stochastic optimization method as in [76]. For notational simplicity, we use $\mathbb{E}_{\boldsymbol{v}}[\cdot]$ to represent $\mathbb{E}_{\theta\sim f_{\boldsymbol{v}}}[\cdot]$ in the rest of this chapter.

As explained in the previous section, we aim at finding a policy distribution $f_{\boldsymbol{v}^*}$ to maximize $L(\boldsymbol{v};\rho)$. By definition of ρ-quantiles, it is a rare event to sample the highly ranked policies for small ρ. The idea behind CE is to treat this optimization problem as an estimation problem of rare-event probabilities. With importance sampling, we may estimate $L(\boldsymbol{v};\rho)$ using any sampling distribution g that shares the same support Θ as $f_{\boldsymbol{v}}$, then

$$L(\boldsymbol{v};\rho) = \mathbb{E}_g[G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)\frac{f_{\boldsymbol{v}}(\theta)}{g(\theta)}].$$

It is well-known that the optimal distribution $g^*_{\boldsymbol{v}}$ [152] with minimal variance is

$$g^*_{\boldsymbol{v}}(\theta) = \frac{G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)f_{\boldsymbol{v}}(\theta)}{L(\boldsymbol{v};\rho)}. \tag{4.5}$$

In practice we smoothen the updates by including a learning rate $\alpha \in (0,1)$ so the goal distribution is $\tilde{g}_{\boldsymbol{v}} = \alpha g^*_{\boldsymbol{v}} + (1-\alpha)f_{\boldsymbol{v}}$. Since neither $g^*_{\boldsymbol{v}}$ nor $\tilde{g}_{\boldsymbol{v}}$ are necessarily in $F_{\mathcal{V}}$, we project $\tilde{g}_{\boldsymbol{v}}$ to $f_{\boldsymbol{v}'} \in F_{\mathcal{V}}$ by minimizing the Kullback-Leibler (KL) divergence between $f_{\boldsymbol{v}''} \in F_{\mathcal{V}}$ and $\tilde{g}_{\boldsymbol{v}}$, which is also equivalent to minimizing the cross entropy between $\tilde{g}_{\boldsymbol{v}}$ and $f_{\boldsymbol{v}''}$.

$$\begin{aligned}
\boldsymbol{v}' &= \arg\min_{\boldsymbol{v}''\in\mathcal{V}} D_{KL}(\tilde{g}_{\boldsymbol{v}} \,||\, f_{\boldsymbol{v}''}) \\
&= \arg\max_{\boldsymbol{v}''\in\mathcal{V}} \mathbb{E}_{\theta\sim\tilde{g}_{\boldsymbol{v}}}[\log f_{\boldsymbol{v}''}(\theta)] \\
&= \arg\max_{\boldsymbol{v}''\in\mathcal{V}} \Big(\alpha\mathbb{E}_{\boldsymbol{v}}\big[\frac{G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)}{L(\boldsymbol{v};\rho)}\log f_{\boldsymbol{v}''}(\theta)\big] + (1-\alpha)\mathbb{E}_{\boldsymbol{v}}\big[\log f_{\boldsymbol{v}''}(\theta)\big]\Big).
\end{aligned} \tag{4.6}$$

We focus ourselves on a specific family of distributions over Θ called *natural exponential family* (NEF), which includes many useful distributions such as Gaussian distribution and Gamma

distribution. A formal definition of NEF is as follows.

Definition 2. *A parameterized family* $F_{\mathcal{V}} = \{f_{\boldsymbol{v}} \in \mathcal{D}(\Theta), \boldsymbol{v} \in \mathcal{V} \subseteq \mathbb{R}^{d_{\boldsymbol{v}}}\}$ *is called a* natural exponential family *if there exist continuous mappings* $\Gamma : \mathbb{R}^{d_\theta} \to \mathbb{R}^{d_{\boldsymbol{v}}}$ *and* $K : \mathbb{R}^{d_\theta} \to \mathbb{R}$ *such that* $f_{\boldsymbol{v}}(\theta) = \exp\big(\boldsymbol{v}^\intercal \Gamma(\theta) - K(\boldsymbol{v})\big)$*, where* $\mathcal{V} \subseteq \{\boldsymbol{v} \in \mathbb{R}^{d_{\boldsymbol{v}}} : |K(\boldsymbol{v})| < \infty\}$ *is the natural parameter space and* $K(\boldsymbol{v}) = \log \int_\Theta \exp\big(\boldsymbol{v}^\intercal \Gamma(\theta)\big) d\theta$*.*

Define $m(\boldsymbol{v}) = \mathbb{E}_{\boldsymbol{v}}[\Gamma(\theta)] \in \mathbb{R}^{d_{\boldsymbol{v}}}$ for $\boldsymbol{v} \in \mathcal{V}$, which is continuously differentiable in $\boldsymbol{v}$. It can be verified that

$$\begin{aligned} m(\boldsymbol{v}) =& \frac{\partial}{\partial \boldsymbol{v}} K(\boldsymbol{v}) \\ \frac{\partial}{\partial \boldsymbol{v}} m(\boldsymbol{v}) =& \mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)], \end{aligned}$$

where $\mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)]$ denotes the covariance matrix of $\Gamma(\theta)$ with $\theta \sim f_{\boldsymbol{v}}$. We take Assumption 1 to guarantee that m^{-1} exists and is continuously differentiable over $\{\eta : \exists\ \boldsymbol{v} \in int(\mathcal{V})\ s.t.\ \eta = m(\boldsymbol{v})\}$. The proof can be done by directly applying the inverse function theorem to m on $int(\mathcal{V})$.

Assumption 1. $\mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)]$ *is positive definite for any* $\boldsymbol{v} \in \mathcal{V} \subseteq int(\{\boldsymbol{v} \in \mathbb{R}^{d_{\boldsymbol{v}}} : |K(\boldsymbol{v})| < \infty\})$.

With Assumption 1, $\nabla^2 K(\boldsymbol{v}) = \mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)] \succcurlyeq 0$ and thus $K(\boldsymbol{v})$ is convex in $\boldsymbol{v}$. Thus $\log f_{\boldsymbol{v}''}(\theta) = (\boldsymbol{v}'')^\intercal \Gamma(\theta) - K(\boldsymbol{v}'')$ is concave in $\boldsymbol{v}''$. As a result, $\boldsymbol{v}'$ in (4.6) can be found by setting

$$\frac{\partial}{\partial \boldsymbol{v}''}\Big(- \int_\Theta \tilde{g}_{\boldsymbol{v}}(\theta) \log f_{\boldsymbol{v}''}(\theta) d\theta \Big) = \mathbf{0},$$

which induces

$$m(\boldsymbol{v}') - m(\boldsymbol{v}) = \alpha\big(\mathbb{E}_{g^*_{\boldsymbol{v}}}[\Gamma(\theta)] - m(\boldsymbol{v})\big). \tag{4.7}$$

As a property of NEF, the KL-divergence of $f_{\boldsymbol{v}}$ from g satisfies $\frac{\partial}{\partial \boldsymbol{v}} D_{KL}(g \,||\, f_{\boldsymbol{v}}) = -\mathbb{E}_g[\Gamma(\theta)] + m(\boldsymbol{v})$. Therefore

$$m(\boldsymbol{v}') - m(\boldsymbol{v}) = -\alpha\Big(\frac{\partial}{\partial \boldsymbol{v}''} D_{KL}(g^*_{\boldsymbol{v}} \,||\, f_{\boldsymbol{v}''})\Big)\Big|_{\boldsymbol{v}''=\boldsymbol{v}}, \tag{4.8}$$

which shows that if $\boldsymbol{v}$ is updated to $\boldsymbol{v}'$ by solving (4.6), $m(\boldsymbol{v})$ will be updated in the negative gradient direction of the objective function $D_{KL}(g^*_{\boldsymbol{v}} \,||\, f_{\boldsymbol{v}})$ where $g^*_{\boldsymbol{v}}$ is the optimal sampling distribution from importance sampling.

Define $\tilde{L}(\boldsymbol{v};\rho) = \mathbb{E}_{g^*_{\boldsymbol{v}}}[\Gamma(\theta)] - m(\boldsymbol{v})$. If G is bounded with a strictly positive lower bound, then

$$\begin{aligned}
\tilde{L}(\boldsymbol{v};\rho) &= \frac{\mathbb{E}_{\boldsymbol{v}}[G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)\Gamma(\theta)]}{L(\boldsymbol{v};\rho)} - m(\boldsymbol{v}) \\
&= \int_\Theta \frac{G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)}{L(\boldsymbol{v};\rho)} f_{\boldsymbol{v}}(\theta)(\Gamma(\theta) - m(\boldsymbol{v}))d\theta \\
&\overset{(*)}{=} \int_\Theta \frac{G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)}{L(\boldsymbol{v};\rho)} \big(\frac{\partial}{\partial \boldsymbol{v}} f_{\boldsymbol{v}}(\theta)\big) d\theta \\
&\overset{(**)}{=} \frac{\partial}{\partial \boldsymbol{v}''} \frac{\mathbb{E}_{\boldsymbol{v}''}[G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)]}{L(\boldsymbol{v};\rho)}\Big|_{\boldsymbol{v}''=\boldsymbol{v}} = \frac{\partial}{\partial \boldsymbol{v}''} \log \mathbb{E}_{\boldsymbol{v}''}[G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)]\Big|_{\boldsymbol{v}''=\boldsymbol{v}},
\end{aligned} \tag{4.9}$$

where the $(*)$ step holds by noticing

$$\frac{\partial}{\partial \boldsymbol{v}} f_{\boldsymbol{v}}(\theta) = f_{\boldsymbol{v}}(\theta)(\Gamma(\theta) - m(\boldsymbol{v}))$$

and the $(**)$ step holds by the dominated convergence theorem. Combining (4.7) and (4.9), we get

$$m(\boldsymbol{v}') - m(\boldsymbol{v}) = \alpha\tilde{L}(\boldsymbol{v};\rho) = \alpha \frac{\partial}{\partial \boldsymbol{v}''} \log \mathbb{E}_{\boldsymbol{v}''}[G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)]\Big|_{\boldsymbol{v}''=\boldsymbol{v}}, \tag{4.10}$$

which leads to the second interpretation of the updates: The update from $\boldsymbol{v}$ to $\boldsymbol{v}'$ approximately follows the gradient direction of $\log L(\boldsymbol{v}'';\rho)$, while the quantiles are estimated using the previous distribution $f_{\boldsymbol{v}}$.

If we apply $\log f_{\boldsymbol{v}''}(\theta) = (\boldsymbol{v}'')^\intercal \Gamma(\theta) - K(\boldsymbol{v}'')$ to (4.6), we can simplify the right-hand side (RHS) of (4.6) as

$$\big(\alpha\mathbb{E}_{\boldsymbol{v}}\big[\frac{G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)\Gamma(\theta)}{L(\boldsymbol{v};\rho)}\big] + (1-\alpha)m(\boldsymbol{v})\big)^\intercal \boldsymbol{v}'' - K(\boldsymbol{v}''),$$

which is concave in $\boldsymbol{v}''$. By setting the derivative with respect to $\boldsymbol{v}''$ as zero, we get an explicit expression of $m(\boldsymbol{v}')$ as in (4.11).

$$m(\boldsymbol{v}') = \alpha\mathbb{E}_{\boldsymbol{v}}\big[\frac{G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)\Gamma(\theta)}{L(\boldsymbol{v};\rho)}\big] + (1-\alpha)m(\boldsymbol{v}). \tag{4.11}$$

Algorithm 2 Constrained Cross-Entropy Method

Require: An objective function G, a constraint function H, a constraint upper bound d, a class of parameterized policies Π_Θ, an NEF family $F_\mathcal{V}$.

1: $l \leftarrow 1$. Initialize $n_l, \boldsymbol{v}_l, \rho, \lambda_l, \alpha_l$. $k_l \leftarrow \lceil \rho n_l \rceil$. $\hat{\eta}_l \leftarrow \mathbf{0}$.
2: **repeat**
3: Sample $\theta_1, \ldots, \theta_{n_l} \sim f_{\boldsymbol{v}_l}$ i.i.d..
4: **for** $i = 1, \ldots, n_l$ **do**
5: Simulate π_{θ_i} and estimate $G(\pi_{\theta_i})$, $H(\pi_{\theta_i})$.
6: **end for**
7: Sort $\{\theta_i\}_{i=1}^{n_l}$ in ascending order of H. Let Λ_l be the first k_l elements.
8: **if** $H(\pi_{\theta_{k_l}}) \leq d$ **then**
9: Sort $\{\theta_i \mid H(\pi_{\theta_i}) \leq d\}$ in descending order of G. Let Λ_l be the first k_l elements.
10: **end if**
11: $\hat{\eta}_{l+1} \leftarrow \alpha_l \sum_{\theta \in \Lambda_l} \frac{G(\pi_\theta)}{\sum_{\theta \in \Lambda_l} G(\pi_\theta)} \Gamma(\theta) + (1 - \alpha_l)\big(\frac{\lambda_l}{n_l} \sum_{i=1}^{n_l} \Gamma(\theta_i) + (1 - \lambda_l)\hat{\eta}_l\big)$.
12: $\boldsymbol{v}_{l+1} \leftarrow m^{-1}(\hat{\eta}_{l+1})$.
13: Update n_l, λ_l, α_l. $l \leftarrow l + 1$. $k_l \leftarrow \lceil \rho n_l \rceil$.
14: **until** Stopping rule is satisfied.

The pseudocode of the CCE algorithm is given in Algorithm 2, which approximately takes the updates in (4.11) in each iteration, with all expectations and quantiles estimated by Monte Carlo simulation. Given $f_{\boldsymbol{v}_l} \in \mathcal{D}(\Theta)$ in the l^{th} iteration, we sample over policies (Step 3), evaluate their G-values and H-values (Step 5), estimate $S(\cdot, \boldsymbol{v}, \rho)$ (Step 7 to 10) and estimate $m(\boldsymbol{v}_{l+1})$ with $\hat{\eta}_{l+1}$ (Step 11) and finally update the sampling distribution to $\boldsymbol{v}_{l+1}$ (Step 12).

4.4.4. Convergence Analysis

We prove the convergence of Algorithm 2 by comparing the asymptotic behavior of $\{\hat{\eta}_l\}_{l \geq 0}$ with the flow induced by the following ordinary differential equation (ODE):

$$\frac{\partial \eta(t)}{\partial t} = \tilde{L}(m^{-1}(\eta(t)); \rho), \tag{4.12}$$

where we define $\eta = m(\boldsymbol{v})$ or equivalently, $\boldsymbol{v} = m^{-1}(\eta)$. The main result that connects the asymptotic behavior of Algorithm 2 with that of an ODE is stated in Theorem 1.

Theorem 1. *If Assumptions 1 and 2 hold, the sequence $\{\hat{\eta}_l\}_{l \geq 0}$ in Step 11 of Algorithm 2 converges*

to a connected internally chain recurrent set of (4.12) *as* $l \to \infty$ *with probability 1.*

By definition of η in (4.12), we know

$$\frac{\partial \eta(t)}{\partial t} = \frac{\partial \boldsymbol{v}}{\partial t} \cdot \mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)].$$

Since $\mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)]$ is invertible by Assumption 1, (4.12) can be rewritten with variable $\boldsymbol{v}$

$$\frac{\partial \boldsymbol{v}}{\partial t} = \Big(\tilde{L}(\boldsymbol{v};\rho)\Big)^{\intercal}\big(\mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)]\big)^{-1}. \tag{4.13}$$

The conclusion of Theorem 1 can be equivalently stated in terms of the variable $\boldsymbol{v}$: the sequence $\{\boldsymbol{v}_l\}_{l\geq 0}$ of Algorithm 2 converges to a connected internally chain recurrent set of (4.13) as $l \to \infty$ with probability 1.

Intuitively, a point $\boldsymbol{v}_0 \in \mathcal{V}$ is *chain recurrent* for (4.13) if the solution $\boldsymbol{v}(t)$ of (4.13) with initial condition $\boldsymbol{v}(0) = \boldsymbol{v}_0$ can return to $\boldsymbol{v}_0$ within some finite time $t' > 0$ itself or just with finitely many arbitrarily small perturbations. An *internally chain recurrent set* is a nonempty compact *invariant* set of chain-recurrent points. In other words, $\boldsymbol{v}$ can never leave an internally chain recurrent set if $\boldsymbol{v}_0$ belongs to it.

Theorem 1 implies that with probability 1, the set of points that occur infinitely often in $\{\boldsymbol{v}_l\}_{l\geq 0}$ are internally chain recurrent for (4.13). Since $f_{\boldsymbol{v}}$ belongs to NEF, $\mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)]$ is the Fisher information matrix at $\boldsymbol{v}$ and the right hand side of (4.13) is an estimate of the natural gradient of $\log L(\boldsymbol{v}; p)$ with a fixed indicator function S. This suggests that $\boldsymbol{v}$ evolves to increase $L(\boldsymbol{v};\rho)$, which is consistent with the optimization problem (4.4) and our motivation to solve a constrained RL problem. Note that internally chain-recurrent sets are generally not unique and our algorithm can still converge to a local optimum.

We need a series of assumptions for technical reasons.

Assumption 2. *(2a)* $\tilde{L}(\boldsymbol{v};\rho)$ *is continuous in* $\boldsymbol{v} \in int(\mathcal{V})$ *and* (4.12) *has a unique integral curve for any given initial condition.*

(2b) The number of samples in the l^{th} iteration is $n_l = \Theta(l^\beta)$, $\beta > 0$. The gain sequence $\{\alpha_l\}$ is positive and decreasing with $\lim_{l\to\infty} \alpha_l = 0$, $\sum_{l=1}^{\infty} \alpha_l = \infty$. $\{\lambda_l\}$ satisfies $\lambda_l = O(\frac{1}{l^\lambda})$ for some $\lambda > 0$ such that $\beta + 2\lambda > 1$.

(2c) For any $\rho \in (0, 1)$ and $f_{\boldsymbol{v}}$ for any $\boldsymbol{v} \in \mathcal{V}$, the ρ-quantile of $\{H(\pi_\theta) : \theta \sim f_{\boldsymbol{v}}\}$ and the $(1-\rho)$-quantile of $\{U(\pi_\theta) : \theta \sim f_{\boldsymbol{v}}\}$ are both unique.

(2d) Both Θ and $\mathcal{V}$ are compact.

(2e) The function G defined in Problem 1 is bounded and has a positive lower bound: $\inf_{\pi\in\Pi} G(\pi) > 0$. The function H in Problem 1 is bounded.

(2f) $\boldsymbol{v}_l \in int(\mathcal{V})$ for any iteration l.

Assumption (2a) ensures that (4.12) is well-posed and has a unique solution. Assumption (2b) addresses some requirements on the number of sampled policies in each iteration and other hyper-parameters in Algorithm 2. Assumptions (2c) to (2e) are used in the proof of the convergence of Algorithm 2. Assumption (2c) is required to show that $\frac{1}{n_l}\sum_{\theta\in\Lambda_l} G(\pi_\theta)$ in Step 11 of Algorithm 2 is an unbiased estimate of $\mathbb{E}_{\boldsymbol{v}_l}[G(\pi_\theta)S(\pi_\theta, \boldsymbol{v}_l, \rho)]$. Assumption (2d) and (2e) are compactness and boundedness constraints for the sets and functions involved in Algorithm 2, which are unlikely to be restrictive in practice. Assumption (2f) states that $\mathcal{V}$ is large enough such that the learned $\boldsymbol{v}$ lies within its interior.

The main idea behind the proof of Theorem 1 is similar to that of Theorem 3.1 in [76], although the details are tailored to our problem. There are two major parts in the convergence proof: The first part shows that all the sampling-based estimates converge to the true values almost surely, including sample quantiles and sample estimates of G, H and L. The second part shows that the asymptotic behavior of the idealized updates in (4.7) can be described by the ODE (4.12).

In practice we can only estimate the expectations and quantiles in (4.11) using finite samples. Let $\mathcal{Y}_l = \{\theta_1, \ldots, \theta_{n_l}\}$ be the set of samples in the l^{th} iteration with sampling distribution $f_{\boldsymbol{v}_l}$. We denote the sample estimate of $S(\pi_\theta, \boldsymbol{v}, \rho)$ as $\hat{S}(\pi_\theta, \boldsymbol{v}, \rho)$.

Consider the equation in the Step 11 of Algorithm 2:

$$\hat{\eta}_{l+1} = \alpha_l \frac{\sum_{i=1}^{n_l} G(\pi_{\theta_i}) \hat{S}(\pi_{\theta_i}, \boldsymbol{v}_l, \rho) \Gamma(\theta_i)}{\sum_{i=1}^{n_l} G(\pi_{\theta_i}) \hat{S}(\pi_{\theta_i}, \boldsymbol{v}_l, \rho)} + (1 - \alpha_l) \Big(\frac{\lambda_l}{n_l} \sum_{i=1}^{n_l} \Gamma(\theta_i) + (1 - \lambda_l) \hat{\eta}_l \Big), \tag{4.14}$$

where $\boldsymbol{v}_l = m^{-1}(\hat{\eta}_l)$ and $\boldsymbol{v}_{l+1} = m^{-1}(\hat{\eta}_{l+1})$. We can rewrite (4.14) as

$$m(\boldsymbol{v}_{l+1}) - m(\boldsymbol{v}_l) = \hat{\eta}_{l+1} - \hat{\eta}_l = \alpha_l \Big(\tilde{L}(\boldsymbol{v}_l; \rho) + b_l + w_l \Big), \tag{4.15}$$

where

$$\begin{aligned} b_l =& \frac{\sum_{i=1}^{n_l} G(\pi_{\theta_i}) \hat{S}(\pi_{\theta_i}, \boldsymbol{v}_l, \rho) \Gamma(\theta_i)}{\sum_{i=1}^{n_l} G(\pi_{\theta_i}) \hat{S}(\pi_{\theta_i}, \boldsymbol{v}_l, \rho)} - \frac{\mathbb{E}_{\boldsymbol{v}_l}[G(\pi_\theta) S(\pi_\theta, \boldsymbol{v}_l, \rho) \Gamma(\theta)]}{\mathbb{E}_{\boldsymbol{v}_l}[G(\pi_\theta) S(\pi_\theta, \boldsymbol{v}_l, \rho)]}, \\ w_l =& \frac{1 - \alpha_l}{\alpha_l} \Big(\frac{\lambda_l}{n_l} \sum_{i=1}^{n_l} \Gamma(\theta_i) - \lambda_l \hat{\eta}_l \Big). \end{aligned} \tag{4.16}$$

Comparing (4.15) and (4.10), we see that the error is sampling-based estimation of all expectations and quantiles is captured by b_l and w_l.

We aim to show the connection between $\{\hat{\eta}_l\}_{l \geq 0}$ and the ODE (4.12) using the following conclusion in stochastic approximation.

Theorem 2. *(Theorem 1.2, [17], with modified notation) Let $Y : \mathbb{R}^m \to \mathbb{R}^m$ be a continuous vectorfield with unique integral curves. Let $\{v_n\}_{n \geq 0}$ be the solution to $v_{n+1} - v_n = \gamma_n(Y(v_n) + u_n + b_n)$, where $\{\gamma_n\}_{n \geq 0}$ is a decreasing gain sequence. Assume that*

- *$\{\gamma_n\}_{n \geq 0}$ is bounded.*
- *$\lim_{n \to +\infty} b_n = 0$.*
- *For any $N > 0$,*

$$\lim_{n \to \infty} \Big(\sup_{k : 0 \leq \tau_k - \tau_n \leq N} || \sum_{i=n}^{k-1} \gamma_i u_i || \Big) = 0,$$

  where $\{\tau_n\}_{n \in \mathbb{N}}$ is defined as: $\tau_0 = 0$, $\tau_n = \sum_{i=0}^{n-1} \gamma_i$.

Then the limit set of $\{v_n\}_{n \geq 0}$ is a connected set internally chain-recurrent for the flow induced by Y.

We first show that $\lim_{l\to\infty} b_l = 0$ where b_l is defined in (4.16), which is stated in Lemma 3.

Lemma 3. *With Assumption (2b), (2c), (2d), (2e),* $\lim_{l\to\infty} b_l = 0$*, with probability 1.*

In order to prove Lemma 3, we first show that the sample quantile is an unbiased estimate of the true quantile, which is stated in Lemma 4. Although we only show the result for the ρ-quantile of H, similar results apply for the $(1-\rho)$-quantile of U.

Lemma 4. *Given $\rho \in (0,1)$, let $\xi(\rho, \boldsymbol{v}_l)$ be the true ρ-quantile of $H(\pi_\theta)$ with $\theta \sim f_{\boldsymbol{v}_l}$ and $\hat{\xi}_l$ be a sample ρ-quantile acquired from n_l i.i.d. samples. With Assumption (2b), (2c), (2d), (2e), $\hat{\xi}_l - \xi(\rho, \boldsymbol{v}_l) \to 0$ as $l \to \infty$ with probability 1.*

Proof. By Assumption (2e), $H(\pi_\theta) \in \mathcal{H} = [H_{min}, H_{max}]$ for all $\pi_\theta \in \Pi_\Theta$ for some $H_{min}, H_{max} \in \mathbb{R}$. It can be verified that any ρ-quantile $\xi(\rho, \boldsymbol{v}_l)$ with $\theta \sim f_{\boldsymbol{v}_l}(\cdot)$ can be represented as an optimal solution of the following optimization problem [74]:

$$\min_{\gamma\in\mathcal{H}} J_l(\gamma) = \mathbb{E}_{\boldsymbol{v}_l}[h(H(\pi_\theta), \gamma)]$$

$$s.t.\ h(H(\pi_\theta), \gamma) = \begin{cases} \rho(H(\pi_\theta) - \gamma), & \text{if } H(\pi_\theta) \geq \gamma, \\ (1-\rho)(\gamma - H(\pi_\theta)), & \text{if } H(\pi_\theta) < \gamma. \end{cases}$$

Similarly the sample ρ-quantile $\hat{\xi}_l$ can be computed by minimizing

$$\hat{J}_l(\gamma) = \frac{1}{n_l} \sum_{i=1}^{n_l} h(H(\pi_{\theta_i}), \gamma),$$

where $\{\theta_1, \ldots, \theta_{n_l}\}$ are i.i.d. samples with distribution $f_{\boldsymbol{v}_l}$.

We first show that $J_l(\gamma)$ uniformly converges to $\hat{J}_l(\gamma)$ over $\mathcal{H}$ with probability 1, i.e. $\sup_{\gamma\in\mathcal{H}} |J_l(\gamma) - \hat{J}_l(\gamma)| \to 0$ as $l \to \infty$ with probability 1.

Let δ and r be two arbitrary scalars such that $\delta > 0$ and $r \leq \frac{\delta}{3\max(\rho, 1-\rho)}$. Let $B(\gamma, r) = \{\gamma' \in \mathcal{H} : ||\gamma - \gamma'|| \leq r\}$ be the r-neighborhood of $\gamma \in \mathcal{H}$ within $\mathcal{H}$. Since $\mathcal{H}$ is compact, there exists a finite set $\mathcal{U} = \{h_1, \ldots, h_k\} \subset \mathcal{H}$ such that $\mathcal{H} \subseteq \bigcup_{i=1}^{k} B(h_i, r)$. For each $\gamma \in \mathcal{H}$, let $u(\gamma) \in \mathcal{U}$ be the

closest component in $\mathcal{U}$. By definition, $\sup_{\gamma\in\mathcal{H}} ||\gamma - u(\gamma)|| \leq r$. For any $\gamma \in \mathcal{H}$,

$$\begin{aligned}
|J_l(\gamma) - J_l(u(\gamma))| &= |\mathbb{E}_{\boldsymbol{v}_l}[h(H(\pi_\theta), \gamma)] - \mathbb{E}_{\boldsymbol{v}_l}[h(H(\pi_\theta), u(\gamma))]\ | \\
&\leq \max(\rho, 1-\rho) \sup_{\gamma\in\mathcal{H}} ||\gamma - h(\gamma)|| \leq \frac{\delta}{3}, \text{ and} \\
|\hat{J}_l(\gamma) - \hat{J}_l(u(\gamma))| &= \frac{1}{n_l}\ |\sum_{i=1}^{n_l} \Big(h(H(\pi_{\theta_i}), \gamma) - h(H(\pi_{\theta_i}), u(\gamma))\Big)\ | \\
&\leq \max(\rho, 1-\rho) \sup_{\gamma\in\mathcal{H}} ||\gamma - u(\gamma)|| \leq \frac{\delta}{3}.
\end{aligned}$$

As $H(\cdot) \subseteq [H_{min}, H_{max}]$, we can bound the probability that $|J_l(u(\gamma)) - \hat{J}_l(u(\gamma))| > \delta/3$ for any $\delta \geq 0$ by Hoeffding's bound:

$$Pr\big(|J_l(u(\gamma)) - \hat{J}_l(u(\gamma))| \geq \frac{\delta}{3}\big) \leq 2e^{-\frac{2n_l\delta^2}{9|H_{max}-H_{min}|^2}}.$$

As card$(\mathcal{U}) = k < \infty$, we can bound the probability that $|J_l(h_i) - \hat{J}_l(h_i)| < \frac{\delta}{3}$ holds for all $h_i \in \mathcal{U}$ with the union bound:

$$Pr\big(\max_{h_i\in\mathcal{U}} |J_l(h_i) - \hat{J}_l(h_i)| \geq \frac{\delta}{3}\big) \leq \sum_{i=1}^{k} Pr\big(|J_l(h_i) - \hat{J}_l(h_i)| \geq \frac{\delta}{3}\big) \leq 2ke^{-\frac{2n_l\delta^2}{9|H_{max}-H_{min}|^2}}.$$

Therefore with probability at least $\Big(1 - 2ke^{-\frac{2n_l\delta^2}{9|H_{max}-H_{min}|^2}}\Big)$,

$$|J_l(\gamma) - \hat{J}_l(\gamma)| \leq \frac{\delta}{3} + \frac{\delta}{3} + \frac{\delta}{3} = \delta$$

holds uniformly for all $\gamma \in \mathcal{H}$. Therefore

$$\sum_{l=1}^{\infty} Pr(\sup_{\gamma\in\mathcal{H}} |J_l(\gamma) - \hat{J}_l(\gamma)| > \delta) \leq \sum_{l=1}^{\infty} 2ke^{-\frac{2n_l\delta^2}{9|H_{max}-H_{min}|^2}} < \infty.$$

The last inequality holds as $n_l = \Theta(l^\beta)$ and $\beta > 0$ by Assumption (2b). By Borel-Cantelli Lemma, $Pr(\sup_{\gamma\in\mathcal{H}} |J_l(\gamma) - \hat{J}_l(\gamma)| > \delta\ i.o.) = 0$. As the above proof holds for any $\delta > 0$, $\sup_{\gamma\in\mathcal{H}} |J_l(\gamma) - \hat{J}_l(\gamma)| \to 0$ as $l \to \infty$ with probability 1. In other words, $\hat{J}_l(\cdot)$ converges

uniformly to $J_l(\cdot)$ as $l \to \infty$ with probability 1. Note that this uniform convergence holds whenever Assumption (2b) and (2e) hold.

Then we prove that $\lim_{l\to+\infty} |\hat{\xi}_l - \xi(\rho, \boldsymbol{v}_l)| = 0$, with probability 1.

Since $\sup_{\gamma\in\mathcal{H}} |J_l(\gamma) - \hat{J}_l(\gamma)| \to 0$ as $l \to \infty$ with probability 1, for any $\varepsilon > 0$, there exists some $L(\varepsilon) > 0$ such that $\sup_{\gamma\in\mathcal{H}} |J_l(\gamma) - \hat{J}_l(\gamma)| < \varepsilon$ holds for all $l > L(\varepsilon)$, with probability 1. Therefore with probability 1 and $l > L(\varepsilon)$,

$$J_l(\hat{\xi}_l) - \varepsilon < \hat{J}_l(\hat{\xi}_l), \quad \hat{J}_l(\xi(\rho, \boldsymbol{v}_l)) < J_l(\xi(\rho, \boldsymbol{v}_l)) + \varepsilon.$$

As $\xi(\rho, \boldsymbol{v}_l)$ minimizes $J_l(\cdot)$ and $\hat{\xi}_l$ minimizes $\hat{J}_l(\cdot)$, we have

$$J_l(\xi(\rho, \boldsymbol{v}_l)) \leq J_l(\hat{\xi}_l), \quad \hat{J}_l(\hat{\xi}_l) \leq \hat{J}_l(\xi(\rho, \boldsymbol{v}_l)).$$

Combining the above two equalities, we get

$$J_l(\xi(\rho, \boldsymbol{v}_l)) - \varepsilon \leq J_l(\hat{\xi}_l) - \varepsilon < \hat{J}_l(\hat{\xi}_l) \leq \hat{J}_l(\xi(\rho, \boldsymbol{v}_l)) < J_l(\xi(\rho, \boldsymbol{v}_l)) + \varepsilon.$$

Therefore for any $\varepsilon > 0$ and $l > L(\varepsilon)$,

$$J_l(\xi(\rho, \boldsymbol{v}_l)) - \varepsilon < J_l(\hat{\xi}_l) < J_l(\xi(\rho, \boldsymbol{v}_l)) + \varepsilon$$

with probability 1. Equivalently, $J_l(\hat{\xi}_l) - J_l(\xi(\rho, \boldsymbol{v}_l)) \to 0$ as $l \to +\infty$ with probability 1.

We define $J_{\boldsymbol{v}}$ in the same way as we defined J_l, namely,

$$J_{\boldsymbol{v}}(\gamma) = \mathbb{E}_{\boldsymbol{v}}[h(H(\pi_\theta), \gamma)]$$

for all $\boldsymbol{v} \in \mathcal{V}$ and $\gamma \in \mathcal{H}$. By Assumption (2c), the ρ-quantile of $\{H(\pi_\theta) : \theta \sim f_{\boldsymbol{v}}(\cdot)\}$ is unique for all $\boldsymbol{v} \in \mathcal{V}$, i.e., $J_{\boldsymbol{v}}(\gamma)$ is minimized with a unique $\xi(\rho, \boldsymbol{v})$ for all $\boldsymbol{v} \in \mathcal{V}$. We can verify from the

definition of $J_{\boldsymbol{v}}(\cdot)$ such that if $\gamma \leq \xi(\rho, \boldsymbol{v})$,

$$\begin{aligned} & J_{\boldsymbol{v}}(\gamma) - J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v})) \\ = \; & \mathbb{E}_{\boldsymbol{v}}\big[(H(\pi_\theta) - \gamma)\mathbf{1}_{[\gamma, \xi(\rho, \boldsymbol{v}))}(H(\pi_\theta))\big] + (\xi(\rho, \boldsymbol{v}) - \gamma)\big(Pr_{\boldsymbol{v}}(H(\pi_\theta) \geq \xi(\rho, \boldsymbol{v})) - (1 - \rho)\big). \end{aligned}$$

If $\gamma > \xi(\rho, \boldsymbol{v})$,

$$\begin{aligned} & J_{\boldsymbol{v}}(\gamma) - J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v})) \\ = \; & \mathbb{E}_{\boldsymbol{v}}\big[(\gamma - H(\pi_\theta))\mathbf{1}_{(\xi(\rho, \boldsymbol{v}), \gamma)}(H(\pi_\theta))\big] + (\gamma - \xi(\rho, \boldsymbol{v}))(Pr_{\boldsymbol{v}}(H(\pi_\theta) \leq \xi(\rho, \boldsymbol{v})) - \rho). \end{aligned}$$

By definition of $\xi(\rho, \boldsymbol{v})$, it holds that

$$\begin{aligned} Pr_{\boldsymbol{v}}\big(H(\pi_\theta) \geq \xi(\rho, \boldsymbol{v})\big) - (1 - \rho) \geq 0, \\ Pr_{\boldsymbol{v}}\big(H(\pi_\theta) \leq \xi(\rho, \boldsymbol{v})\big) - \rho \geq 0. \end{aligned}$$

Therefore for any $\boldsymbol{v} \in \mathcal{V}$, $J_{\boldsymbol{v}}(\gamma)$ decreases monotonically if $\gamma < \xi(\rho, \boldsymbol{v})$ and increases monotonically if $\gamma > \xi(\rho, \boldsymbol{v})$. Since the global minimizer is always unique, $J_{\boldsymbol{v}}(\gamma) > J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}))$ for any $\gamma \neq \xi(\rho, \boldsymbol{v})$. For any fixed $\delta' > 0$ and any $\boldsymbol{v} \in \mathcal{V}$, $\rho \in (0, 1)$, if $|\gamma - \xi(\rho, \boldsymbol{v})| \geq \delta'$, it holds that

$$|J_{\boldsymbol{v}}(\gamma) - J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}))| \geq \min\big(J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}) + \delta') - J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v})), J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}) - \delta') - J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}))\big).$$

For any fixed $\delta' > 0$ and all $\boldsymbol{v} \in \mathcal{V}$, it holds that

$$J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}) + \delta') - J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v})) > 0 \text{ and } J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}) - \delta') - J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v})) > 0.$$

Since $\mathcal{V}$ is compact, we get

$$\inf_{\boldsymbol{v} \in \mathcal{V}} \big(J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}) + \delta') - J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}))\big) > 0 \text{ and } \inf_{\boldsymbol{v} \in \mathcal{V}} \big(J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}) - \delta') - J_{\boldsymbol{v}}(\xi(\rho, \boldsymbol{v}))\big) > 0$$

for any $\delta' > 0$.

Assume that $\hat{\xi}_l - \xi(\rho, \boldsymbol{v}_l)$ does not converge to 0 with probability 1. Then there exists $\bar{\delta} > 0$ such

that $Pr(\{|\hat{\xi}_l - \xi(\rho, \boldsymbol{v}_l)| \geq \bar{\delta}\ i.o.\}) > 0$. Since $J_l(\hat{\xi}_l) - J_l(\xi(\rho, \boldsymbol{v}_l)) \to 0$, we know that with positive probability, there exists a subsequence $\{l_k\}_{k\geq 0} \in \mathbb{N}^\infty$ such that $|\hat{\xi}_{l_k} - \xi(\rho, \boldsymbol{v}_{l_k})| \geq \bar{\delta}$ for each $k \in \mathbb{N}$ and $\lim_{k\to\infty}(J_{l_k}(\hat{\xi}_{l_k}) - J_{l_k}(\xi(\rho, \boldsymbol{v}_{l_k}))) = 0$. However,

$$
\begin{aligned}
&|J_{l_{k_j}}(\hat{\xi}_{l_k}) - J_{l_k}(\xi(\rho, \boldsymbol{v}_{l_k}))| \\
\geq &\min\Big(\inf_{\boldsymbol{v}\in\mathcal{V}} \big(J(\xi(\rho, \boldsymbol{v}) - \bar{\delta}) - J(\xi(\rho, \boldsymbol{v}))\big), \inf_{\boldsymbol{v}\in\mathcal{V}} \big(J(\xi(\rho, \boldsymbol{v}) + \bar{\delta}) - J(\xi(\rho, \boldsymbol{v}))\big)\Big) > 0,
\end{aligned}
$$

which contradicts our assumption that $\lim_{k\to\infty}(J_{l_k}(\hat{\xi}_{l_k}) - J_{l_k}(\xi(\rho, \boldsymbol{v}_{l_k}))) = 0$. Therefore the assumption is wrong and $\lim_{l\to+\infty} |\hat{\xi}_l - \xi(\rho, \boldsymbol{v}_l)| = 0$ with probability 1. □

We can now give a proof to Lemma 3.

Proof. By Assumption (2e), $\inf_{\pi\in\Pi} G(\pi) > 0$. By definition of $(1-\rho)$-quantile, it holds for any $\boldsymbol{v} \in \mathcal{V}$ that

$$
\mathbb{E}_{\boldsymbol{v}}[G(\pi_\theta)S(\pi_\theta, \boldsymbol{v}, \rho)] \geq \inf_{\pi\in\Pi} G(\pi)\rho > 0.
$$

Similarly we can show

$$
\sum_{i=1}^{n_l} G(\pi_{\theta_i})\hat{S}(\pi_{\theta_i}, \boldsymbol{v}, \rho) \geq \inf_{\pi\in\Pi} G(\pi) > 0.
$$

There are two types of approximation involved in b_l: the first is to approximate $\xi_H(\rho, \boldsymbol{v}_l)$ and $\xi_U(1-\rho, \boldsymbol{v}_l)$ by $\hat{\xi}_{H,l}$ and $\hat{\xi}_{U,l}$. The second is to approximate the expectations with sample means, for example, to approximate $\mathbb{E}_{\boldsymbol{v}_l}[G(\pi_\theta)\hat{S}(\pi_{\theta_i}, \boldsymbol{v}_l, \rho)\Gamma(\theta)]$ with $\frac{1}{n_l}\sum_{i=1}^{n_l} G(\pi_{\theta_i})\hat{S}(\pi_{\theta_i}, \boldsymbol{v}_l, \rho)\Gamma(\theta_i)$.

We have shown that $\lim_{l\to\infty} |\xi_H(\rho, \boldsymbol{v}_l) - \hat{\xi}_{H,l}| = 0$ with probability 1 and $\lim_{l\to\infty} |\xi_U(1-\rho, \boldsymbol{v}_l) - \hat{\xi}_{U,l}| = 0$ with probability 1 by Lemma 4. With the continuous approximation of δ as explained in Remark 3, we can show $\lim_{l\to\infty} |S(\pi_\theta, \boldsymbol{v}_l, \rho) - \hat{S}(\pi_\theta, \boldsymbol{v}_l, \rho)| = 0$ with probability 1. using the continuous mapping theorem. We only need to consider the second part in this proof.

$\Gamma(\cdot)$ is bounded as it is a continuous function defined over a compact set (by Assumption (2d)). By

Assumption (2e), both G and H are bounded over Π. Therefore

$$\lim_{l\to\infty}\Big|\frac{1}{n_l}\sum_{i=1}^{n_l}G(\pi_{\theta_i})\hat{S}(\pi_{\theta_i},\boldsymbol{v}_l,\rho)\Gamma(\theta_i)-\frac{1}{n_l}\sum_{i=1}^{n_l}G(\pi_{\theta_i})S(\pi_{\theta_i},\boldsymbol{v}_l,\rho)\Gamma(\theta_i)\Big|=0$$

holds with probability 1.

As $G(\pi_\theta)$, $S(\pi_\theta,\boldsymbol{v}_l,\rho)$, $\Gamma(\theta)$ are all bounded for any θ and ρ, there exist finite a,b such that $a\le G(\pi_\theta)S(\pi_\theta,\boldsymbol{v}_l,\rho)\Gamma(\theta)\le b$ for any $\theta\in\Theta$. By Hoeffding's inequality, for any $\varepsilon>0$

$$Pr(\Big|\frac{1}{n_l}\sum_{i=1}^{n_l}G(\pi_{\theta_i})S(\pi_{\theta_i},\boldsymbol{v}_l,\rho)\Gamma(\theta_i)-\mathbb{E}_{\boldsymbol{v}_l}[G(\pi_\theta)S(\pi_\theta,\boldsymbol{v}_l,\rho)\Gamma(\theta)]\Big|\ge\varepsilon)\le 2e^{\frac{-2n_l\varepsilon^2}{(b-a)^2}}.$$

By Assumption (2b), $n_l=\Theta(l^\beta)$ and $\beta>0$. Therefore for any $\varepsilon>0$,

$$\sum_{l=1}^{\infty}Pr(\Big|\frac{1}{n_l}\sum_{i=1}^{n_l}G(\pi_{\theta_i})S(\pi_{\theta_i},\boldsymbol{v}_l,\rho)\Gamma(\theta_i)-\mathbb{E}_{\boldsymbol{v}_l}[G(\pi_\theta)S(\pi_\theta,\boldsymbol{v}_l,\rho)\Gamma(\theta)]\Big|\ge\varepsilon)\le\sum_{l=1}^{\infty}2e^{\frac{-2n_l\varepsilon^2}{(b-a)^2}}<\infty.$$

Then by Borel-Cantelli Lemma, with probability 1,

$$\lim_{l\to\infty}\Big|\frac{1}{n_l}\sum_{i=1}^{n_l}G(\pi_{\theta_i})S(\pi_{\theta_i},\boldsymbol{v}_l,\rho)\Gamma(\theta_i)-\mathbb{E}_{\boldsymbol{v}_l}[G(\pi_\theta)S(\pi_\theta,\boldsymbol{v}_l,\rho)\Gamma(\theta)]\Big|=0.$$

Similarly, we can show that with probability 1,

$$\lim_{l\to\infty}\Big|\frac{1}{n_l}\sum_{i=1}^{n_l}G(\pi_{\theta_i})S(\pi_{\theta_i},\boldsymbol{v}_l,\rho)-\mathbb{E}_{\boldsymbol{v}_l}[G(\pi_\theta)S(\pi_\theta,\boldsymbol{v}_l,\rho)]\Big|=0.$$

Then $\lim_{l\to\infty}b_l=0$ holds with probability 1 by continuous mapping theorem. □

Now we provide a proof for Theorem 1.

Proof. We connect the sequence $\{\hat{\eta}_l\}_{l\ge0}$ to the ODE (4.12) by applying Theorem 2. We need to verify that all sufficient conditions in 2 hold properly. By (4.15), $\hat{\eta}_{l+1}-\hat{\eta}_l=\alpha_l\Big(\tilde{L}(\boldsymbol{v}_l;\rho)+b_l+w_l\Big)$.

- By Assumption (2a), $\tilde{L}(\boldsymbol{v};\rho)$ is continuous in $\boldsymbol{v}\in int(\mathcal{V})$. Since $m^{-1}(\eta)$ is continuous in η,

$\tilde{L}(\boldsymbol{v};\rho)\Big|_{\boldsymbol{v}=m^{-1}(\eta)}$ is continuous in η. (4.12) has a unique integral curve by Assumption (2a).

- By Assumption (2b), $\{\alpha_l\}_{l\geq 0}$ is bounded and decreasing.
- By Lemma 3, $\lim_{l\to\infty} b_l = 0$ with probability 1 with Assumption (2b), (2c), (2d), (2e).
- Then we show that for any $N \in \mathbb{N}^+$,

$$\lim_{l\to\infty}\Big(\sup_{k:\sum_{i=n}^{k}\alpha_i<N}||\sum_{i=n}^{k}\alpha_i w_i||\Big)=0.$$

Define $M_l = \sum_{i=1}^{l}\alpha_i w_i$. Then $M_l = M_{l-1} + \alpha_n w_n$. As the set $\{\theta_i\}_{i=1}^{n_l}$ is generated i.i.d. with distribution $f_{m^{-1}(\hat{\eta}_l)}(\cdot)$ and $\hat{\eta}_l = \mathbb{E}_{m^{-1}(\hat{\eta}_l)}[\Gamma(\theta)]$, it holds that

$$\mathbb{E}[M_l|M_1,\ldots,M_{l-1}]-M_{l-1}=(1-\alpha_l)\lambda_l\Big(\mathbb{E}_{m^{-1}(\hat{\eta}_l)}[\frac{1}{n_l}\sum_{i=1}^{n_l}\Gamma(\theta_i)|M_{l-1}]-\hat{\eta}_l\Big)=0$$

regardless of the value of $\hat{\eta}_l$. To show that $\{M_n\}_{n\geq 0}$ is a martingale, we show that $\mathbb{E}[||M_n||] < \infty$. Note that w_i is independent of w_j if $i \neq j$, as all θ are independently generated. Therefore $\mathbb{E}[w_i^\intercal w_j] = \mathbb{E}[w_i]^\intercal\mathbb{E}[w_j] = 0$.

$$\begin{aligned}\mathbb{E}[||M_n||^2] =&\mathbb{E}[M_n^\intercal M_n] = \mathbb{E}[\big(\sum_{i=1}^{n}\alpha_i w_i\big)^\intercal\big(\sum_{i=1}^{n}\alpha_i w_i\big)]\\ =&\sum_{i=1}^{n}\alpha_i^2\mathbb{E}[w_i^\intercal w_i]+\sum_{i=1}^{n}\sum_{j\neq i}\alpha_i\alpha_j\mathbb{E}[w_i^\intercal w_j]\\ =&\sum_{i=1}^{n}\alpha_i^2\mathbb{E}[w_i^\intercal w_i]=\sum_{i=1}^{n}\frac{(1-\alpha_i)^2\lambda_i^2}{n_i}\mathrm{Cov}_{m^{-1}(\hat{\eta}_i)}[\Gamma(\theta)].\end{aligned}$$

As $\Gamma(\theta)$ is continuous and the domain Θ is compact, there exists $0 < C < \infty$ such that $\mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)] \leq C$ for any $\boldsymbol{v} \in \mathcal{V}$. Therefore by Assumption (2b),

$$\mathbb{E}[||M_n||^2]\leq\sum_{i=1}^{n}C\frac{(1-\alpha_i)^2\lambda_i^2}{n_i}=O\Big(\sum_{l=1}^{n}\frac{1}{l^{\beta+2\lambda}}\Big).$$

By Assumption (2b), $\beta + 2\lambda > 1$. Therefore $\lim_{n\to\infty}\mathbb{E}[||M_n||^2] < \infty$. As $\{||M_n||^2\}$

increases monotonically, we know

$$\sup_n \mathbb{E}[||M_n||^2] = \lim_{n\to\infty} \mathbb{E}[||M_n||^2] < \infty.$$

Since and $\mathbb{E}[||M_n||] \leq \sqrt{\mathbb{E}[||M_n||^2]}$, it holds that $\sup_n \mathbb{E}[||M_n||] < \infty$ and $\{M_n\}_{n\geq 0}$ is a martingale. Then by L_2 martingale convergence theorem, there exists M_∞ such that $M_n \to M_\infty$ with probability 1 and $\mathbb{E}[||M_\infty||^2] < \infty$.

$$\sup_{\{k:\sum_{i=n}^k \alpha_i < N\}} ||\sum_{i=n}^k \alpha_i w_i|| = \sup_{\{k:\sum_{i=n}^k \alpha_i < N\}} ||M_k - M_{n-1}|| \leq 2 \sup_{k\geq n} ||M_k||.$$

Therefore

$$0 \leq \lim_{n\to\infty} \Big(\sup_{\{k:\sum_{i=n}^k \alpha_i < N\}} ||\sum_{i=n}^k \alpha_i w_i|| \Big) \leq \lim_{n\to\infty} \Big(2 \sup_{k\geq n-1} ||M_k|| \Big) = 0$$

for any finite $N > 0$.

Since all conditions in Theorem 2 are satisfied, the limit set of sequence $\{\hat{\eta}_l\}_{l\geq 0}$ is a internally chain recurrent connected set for the flow induced by $\tilde{L}(m^{-1}(\eta); \rho)$ with probability 1. □

To further interpret Theorem 1, we first note that any equilibrium of (4.12) forms an internally chain recurrent set by itself. The following result shows a sufficient condition for an equilibrium point $\bar{\boldsymbol{v}}^*$ of (4.12) to be locally asymptotically stable, which means that there exists a small neighborhood of $\bar{\boldsymbol{v}}^*$ such that once entered, (4.13) will converge to $\bar{\boldsymbol{v}}^*$.

Theorem 3. *Let $\varphi : \mathcal{V} \to \mathbb{R}$ be any function such that $\frac{\partial}{\partial \boldsymbol{v}} \varphi(\boldsymbol{v}) = \tilde{L}(\boldsymbol{v}; \rho)$. Any equilibrium $\bar{\boldsymbol{v}}^* \in int(\mathcal{V})$ of* (4.13) *that is an isolated local maximum of $\varphi(\boldsymbol{v})$ is locally asympototically stable.*

Proof. The Lyapunov function we use is similar to that in [80]:

$$V(\boldsymbol{v}) = \varphi(\bar{\boldsymbol{v}}^*) - \varphi(\boldsymbol{v}),$$

where $\bar{\boldsymbol{v}}^*$ is an isolated local maximum of $\varphi(\boldsymbol{v})$ and $\boldsymbol{v}$ is in some neighborhood of $\bar{\boldsymbol{v}}^*$ such that

$\varphi(\bar{\boldsymbol{v}}^*) \geq \varphi(\boldsymbol{v})$ and $V(\boldsymbol{v}) \geq 0$. By previous analysis, $\log \varphi(\boldsymbol{v})$ and $V(\boldsymbol{v})$ are continuous in $\boldsymbol{v}$. For the derivative:

$$\frac{dV(\boldsymbol{v})}{dt} = -\frac{\partial \boldsymbol{v}}{\partial t}\frac{\partial \varphi(\boldsymbol{v})}{\partial \boldsymbol{v}} = -\Big(\tilde{L}(\boldsymbol{v};\rho)\Big)^{\intercal}(\mathrm{Cov}[\Gamma(\theta)])^{-1}\tilde{L}(\boldsymbol{v};\rho).$$

As $\mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)]$ is positive definite for $\boldsymbol{v} \in int(\mathcal{V})$, $\big(\mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)]\big)^{-1}$ is also positive definite. Therefore $\frac{\partial V(\boldsymbol{v})}{\partial t} \leq 0$ in a neighborhood of $\boldsymbol{v}^*$ and $\frac{\partial V(\boldsymbol{v})}{\partial t} = \mathbf{0}$ if and only if $\tilde{L}(\boldsymbol{v};\rho) = \mathbf{0}$, which guarantees that $\boldsymbol{v}$ is a stationary point of (4.13). As $\bar{\boldsymbol{v}}^*$ is an isolated local maximum of $\varphi(\boldsymbol{v})$, it is the only stationary point in some neighborhood of $\bar{\boldsymbol{v}}^*$. Therefore $\frac{\partial V(\boldsymbol{v})}{\partial \boldsymbol{v}} = \mathbf{0}$ if and only if $\boldsymbol{v} = \bar{\boldsymbol{v}}^*$ (if $\boldsymbol{v}$ is in the neighborhood of $\boldsymbol{v}^*$) and $\bar{\boldsymbol{v}}^*$ is locally asymptotically stable. □

The proof of Theorem 3 shows that $\varphi(\boldsymbol{v})$ always decreases in the interior of $\mathcal{V}$ unless it hits a stationary point of (4.13), which suggests a stronger property of our algorithm as stated in Theorem 4. In order to state the result we need to first introduce some definitions. By Assumption (2a),

$$Z = \Big(\tilde{L}(\boldsymbol{v};\rho)\Big)^{\intercal}(\mathrm{Cov}_{\boldsymbol{v}}[\Gamma(\theta)])^{-1}$$

is a continuous vector field defined on $\mathcal{V} \subset \mathbb{R}^{d_{\boldsymbol{v}}}$ with unique integral curves. The *flow* of Z is the family of mappings $\{\Phi_t(\cdot)\}_{t\in\mathbb{R}}$ defined on $\mathcal{V}$ by $\frac{\partial \Phi_t(\boldsymbol{v})}{\partial t} = Z(\Phi_t(\boldsymbol{v}))$ such that $\Phi_0(\boldsymbol{v}) \equiv \boldsymbol{v}$ and $\Phi_t(\Phi_s(\boldsymbol{v})) \equiv \Phi_{t+s}(\boldsymbol{v})$ for any $\boldsymbol{v} \in \mathcal{V}$, $t, s \in \mathbb{R}$. $\boldsymbol{v} \in \mathcal{V}$ is an *equilibrium* if $\Phi_t(\boldsymbol{v}) = \boldsymbol{v}$ for all t. A set $\mathcal{V}' \subset \mathcal{V}$ is *positively invariant* under the flow Φ if for all $t \geq 0$, $\Phi_t(\mathcal{V}') = \mathcal{V}'$.

Theorem 4. *If all equilibria of* (4.13) *are isolated, the sequence* $\{\boldsymbol{v}_l\}_{l\geq 0}$ *derived by Algorithm 2 converges toward an equilibrium of* (4.13) *as* $l \to \infty$ *with probability 1.*

Proof. Let φ be defined in the same way as in Theorem 3. We first show that φ is bounded over $\mathcal{V}$. By definition of $\tilde{L}(\boldsymbol{v};\rho)$ in (4.9),

$$\tilde{L}(\boldsymbol{v};\rho) = \frac{\mathbb{E}_{\boldsymbol{v}}[G(\pi_\theta)S(\pi_\theta,\boldsymbol{v},\rho)\Gamma(\theta)]}{L(\boldsymbol{v};\rho)} - m(\boldsymbol{v}).$$

Since G has a positive lower bound (by Assumption (2e)) and $\mathbb{E}_{\boldsymbol{v}}[S(\pi_\theta, \boldsymbol{v}', \rho)] \geq \rho$ for any $\boldsymbol{v} \in \mathcal{V}$,

$$L(\boldsymbol{v}; \rho) \geq \inf_{\pi \in \Pi} G(\pi)\rho > 0.$$

Since Γ is continuous over Θ, Θ and $\mathcal{V}$ are compact (by Assumption (2d)), $\Gamma(\theta)$ and $m(\boldsymbol{v}) = \mathbb{E}_{\boldsymbol{v}}[\Gamma(\theta)]$ are both bounded. Since G is also bounded (by Assumption (2e)), $\mathbb{E}_{\boldsymbol{v}}[G(\pi_\theta)S(\pi_\theta, \boldsymbol{v}, \rho)\Gamma(\theta)]$ is also bounded over $\mathcal{V}$ for any $\rho \in (0, 1)$. Therefore φ is also bounded over $\mathcal{V}$.

Let Φ be a flow induced by (4.13) and Λ be the set of all equilibria of (4.13). By definition, Λ is positively invariant under Φ. Define $V : \mathcal{V} \to \mathbb{R}^{\geq 0}$ as

$$V(\boldsymbol{v}) = \sup_{\boldsymbol{v}' \in \mathcal{V}} \varphi(\boldsymbol{v}') - \varphi(\boldsymbol{v}).$$

$\sup_{\boldsymbol{v}' \in \mathcal{V}} \varphi(\boldsymbol{v}') < \infty$ as φ is shown to be bounded in $\mathcal{V}$. By definition of Λ and the proof of Theorem 3, the mapping $t \mapsto V(\Phi_t(\boldsymbol{v}))$ is constant-valued for $\boldsymbol{v} \in \Lambda$ and strictly decreasing for $\boldsymbol{v} \in int(\mathcal{V}) \backslash \Lambda$. Since we also assume that (4.13) has only isolated equilibria and $\boldsymbol{v}$ is always in the interior of $\mathcal{V}$ (Assumption (2f)), $\{\boldsymbol{v}_l\}_{l \geq 0}$ converges to an equilibrium of (4.13) as $l \to \infty$ with probability 1 by Corollary 3.3 in [17]. □

4.5. Experimental Results

We show the performance of CCE in two numerical experiments: One is a discrete-time finite-horizon constrained linear quadratic regulator problem and the other is a 2D robot navigation problem with only local observations.

4.5.1. Constrained Linear Quadratic Regulator

We first run CCE on a simple finite-horizon constrained linear quadratic regulator (LQR) problem. The problem is convex and thus can be solved efficiently and accurately. The goal of this example is to check if CCE can converge to the globally optimal solution for a convex problem, as well as the effect of policy network structure on the performance of CCE.

Given an initial state $\boldsymbol{x}_0 \in \mathbb{R}^{n_{\boldsymbol{x}}}$, a finite horizon $N \in \mathbb{N}^+$, a lower bound $\boldsymbol{u}_{low} \in \mathbb{R}^{n_{\boldsymbol{u}}}$ and an upper bound $u_{upp} \in \mathbb{R}^{n_{\boldsymbol{u}}}$ of inputs, the optimization problem to be solved is

$$\begin{aligned} \min_{\substack{\boldsymbol{u}_0,\ldots,\boldsymbol{u}_{N-1} \\ \boldsymbol{x}_1,\ldots,\boldsymbol{x}_N}} \quad & \sum_{t=0}^{N-1} \left(\boldsymbol{x}_{t+1}^\intercal Q \boldsymbol{x}_{t+1} + \boldsymbol{u}_t^\intercal R \boldsymbol{u}_t\right) \\ s.t. \quad & \boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t,\ \forall t = 0,\ldots,N-1, \\ & \boldsymbol{u}_{low} \preccurlyeq \boldsymbol{u}_t \preccurlyeq \boldsymbol{u}_{upp},\ \forall t = 0,\ldots,N-1, \\ & \boldsymbol{x}_t \in \mathbb{R}^{n_{\boldsymbol{x}}},\ \boldsymbol{u}_t \in \mathbb{R}^{n_{\boldsymbol{u}}},\ \forall t = 0,\ldots,N-1. \end{aligned} \tag{4.17}$$

It is well known that if $Q \succeq 0$ and $R \succ 0$, an optimal solution $\boldsymbol{u}_t^*$ to (4.17) at each time $t = 0,\ldots,N-1$ is a continuous piecewise affine function of the state $\boldsymbol{x}_t$ [26]. At each time t, there exists a polyhedral partition $\{P_t^j\},\ j = 1,\ldots,k_t$ of $\mathbb{R}^{n_{\boldsymbol{x}}}$ such that $P_t^j = \{\boldsymbol{x} \in \mathbb{R}^{n_{\boldsymbol{x}}} | F_t^j \boldsymbol{x} \leq K_t^j\}$ and $\boldsymbol{u}_t^*(\boldsymbol{x}) = C_t^j \boldsymbol{x} + d_t^j$ for $\boldsymbol{x} \in P_t^j$. Since Problem (4.17) is convex, we can compute its globally optimal solution $\boldsymbol{x}_t^*$ and $\boldsymbol{u}_t^*$ via tools such as CVX [64].

The specific matrices we used are

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, R = \begin{bmatrix} 0.3 \end{bmatrix},$$
$$\boldsymbol{u}_{low} = -0.2, \boldsymbol{u}_{upp} = 0.2, \boldsymbol{x}_0 = \begin{bmatrix} 1 & -1 \end{bmatrix}^\intercal.$$

The horizon length is $N = 20$, which is long enough to for $\boldsymbol{u}^*$ to drive the states back to the origin. The state and input trajectories derived by a globally optimal policy π^* are shown in Figure 8.

We now solve (4.17) using CCE. We define J as the objective function in (4.17) and the constraint function Z as follows:

$$Z(\boldsymbol{x}_0, \boldsymbol{u}_0, \ldots, \boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}, \boldsymbol{x}_N) = 1 - \max_t \max(\boldsymbol{u}_{low} - \boldsymbol{u}_t, \boldsymbol{u}_t - \boldsymbol{u}_{upp}, 0).$$

Therefore, a trajectory $\tau = \boldsymbol{x}_0, \boldsymbol{u}_0, \ldots, \boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}, \boldsymbol{x}_N$ is feasible if and only if $Z(\tau) \geq 1$.

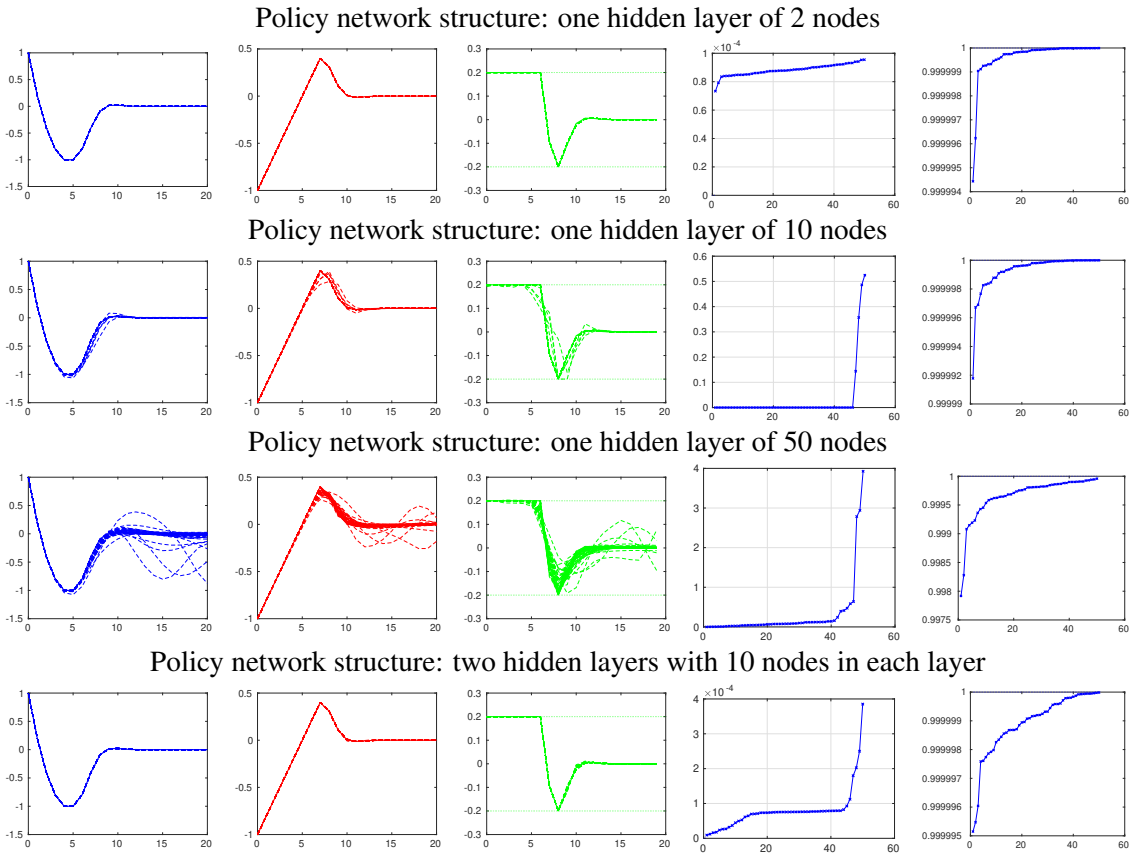


Figure 8: Comparison of the globally optimal policy π^* and the 50 learned policies for different policy network structures. Each row corresponds to a policy network structure. From left to right, the first three columns represent the trajectories of the two states $\boldsymbol{x}_t(1)$, $\boldsymbol{x}_t(2)$ and the input $\boldsymbol{u}_t$ over time t. The solid line in each figure is for π^* and the dashed lines are for the learned policies. In the fourth column, we show the gap between their G-values and $G(\pi^*)$ in ascending order. In the last column, we show the H-values of the learned policies in ascending order.

We use a fully-connected neural network to represent the policy or controller, which maps from the current position $\boldsymbol{x}_t = [\boldsymbol{x}_t(1), \boldsymbol{x}_t(2)]^\intercal \in \mathbb{R}^2$ to an input $\boldsymbol{u}_t \in \mathbb{R}$. We compare four different policy network structures: three networks with one hidden layer of 2, 10 or 50 nodes respectively and one network with two hidden layers of 10 nodes in each layer. The activation function for each hidden layer is the rectified linear unit (ReLU) and thus the learned controller is also a piecewise linear function of the states. There is no activation function for the output layer. Note that the policy represented by the neural network is time-invariant and thus it may be impossible to reach the globally optimal objective value.

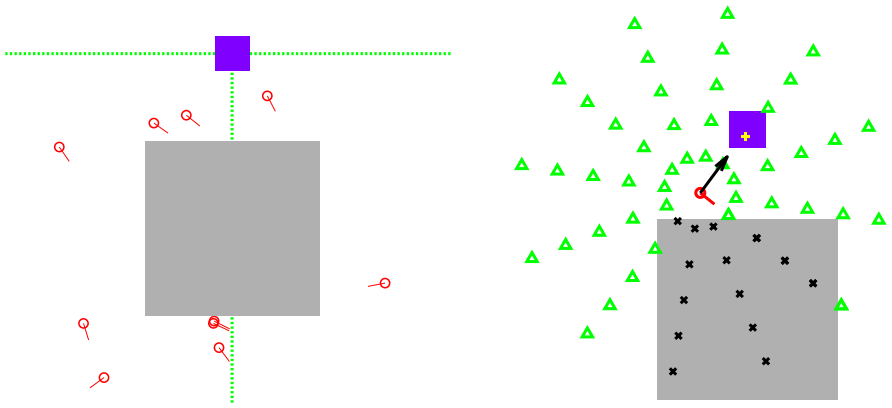
We assume that $F_{\mathcal{V}}$ is a family of Gaussian distributions with diagonal covariance matrices. Each sample policy is represented as a vector composed of all its network weights. For each policy network, we repeatedly run CCE for 50 times. At the beginning of each experiment, we randomly initialize the policy distribution parameter $\boldsymbol{v} \in \mathcal{V}$. In each iteration, we draw 100 sample policies from the current policy distribution. The results are shown in Figure 8, which includes the state and input trajectories of both the globally optimal policy π^* and each learned policy $\hat{\pi}$, the suboptimality gap $G(\hat{\pi}) - G(\pi^*)$ of the G-value and the H-value for each learned policy.

CCE converged in all experiments. The performance of the learned policy is largely dependent on the architecture of the policy network. As this example problem is simple, it turns out that a neural network with a single hidden layer of 2 nodes can approximate the globally optimal policy accurately and consistently. As we increase the number of nodes in the hidden layer, it becomes more difficult to find or converge to feasible solutions; the suboptimality gap of G-value also increases. Meanwhile, neither the number of hidden nodes nor the number of weight parameters is a reliable metric to evaluate the complexity of the model. As the policy network has 2 inputs and 1 output, a network with a single layer of 50 nodes has 150 weight parameters and a network with two hidden layers of 10 nodes has 130 weight parameters. However, Figure 8 shows that the performance of the latter network is much better than the previous one: all the trajectories led by the 50 learned policies are very close to that generated by π^* and the suboptimality gap of G-value is very small.

4.5.2. 2D Navigation

We also consider a mobile robot navigation task with only local observations. Unlike the previous example for which we can reliably compute the globally optimal solution, the optimization problem in this example is non-convex and we have to resort to approximate solutions. The goal is to compare the performance of CCE with that of the constrained policy optimization algorithm, which is a state-of-the-art constrained reinforcement learning algorithm [4].

The robot's state space is $S = \{(x, y, \zeta) \big| x_{min} \leq x \leq x_{max}, y_{min} \leq y \leq y_{max}, -\pi \leq \zeta < \pi\}$, which contains the robot's position and orientation in the global coordinate. The action space



(a) (b)

Figure 9: (9a) Map of the 2D navigation example. There are one obstacle region (grey rectangle), one goal region (blue rectangle) and 10 randomly selected initial states (red circles pointing to the forward direction). Dotted lines are added to show x and y axes. (9b) Illustration of the local features in the robot's local coordinate at one of the initial states, with $n_s = 5$. Obstacle nodes, goal nodes and free nodes are labeled by black crosses, yellow plus signs and green triangles respectively. The goal direction (black arrow) is also included in local features.

is 2-dimensional: $A = \{(v, \omega) \big| |v| \leq v_{max}, |\omega| \leq \omega_{max}\}$, which are linear and angular speed respectively. The environment map is shown in Figure 9a, where there is a compact goal region $\mathcal{G}$ and a disjoint compact obstacle region $\mathcal{B}$. The overall goal of the navigation task is to reach the goal region $\mathcal{G}$ without colliding with the obstacle region $\mathcal{B}$, while the objective and constraint are encoded in four different ways as shown in Table 4.

The policy is again modeled as a fully connected neural network with 2 hidden layers and 30 nodes in each layer. The activation function is ReLU for hidden layers and the hyperbolic tangent function (tanh) for the output layer. The policy network maps from local observations to actions. The local observations are interpreted as follows.

We assume that the robot cannot observe (x, y, ζ) directly and can only use local sensors (shown in Figure 9b) to observe if $\mathcal{B}$ or $\mathcal{G}$ is in its neighborhood and the direction of the center of $\mathcal{G}$ in its local coordinate. For a given parameter $n_s \in \mathbb{N}^+$ and sampling time Δt, we design a radial grid as n_s circles in the robot's local coordinate. The difference between the diameters of adjacent circles

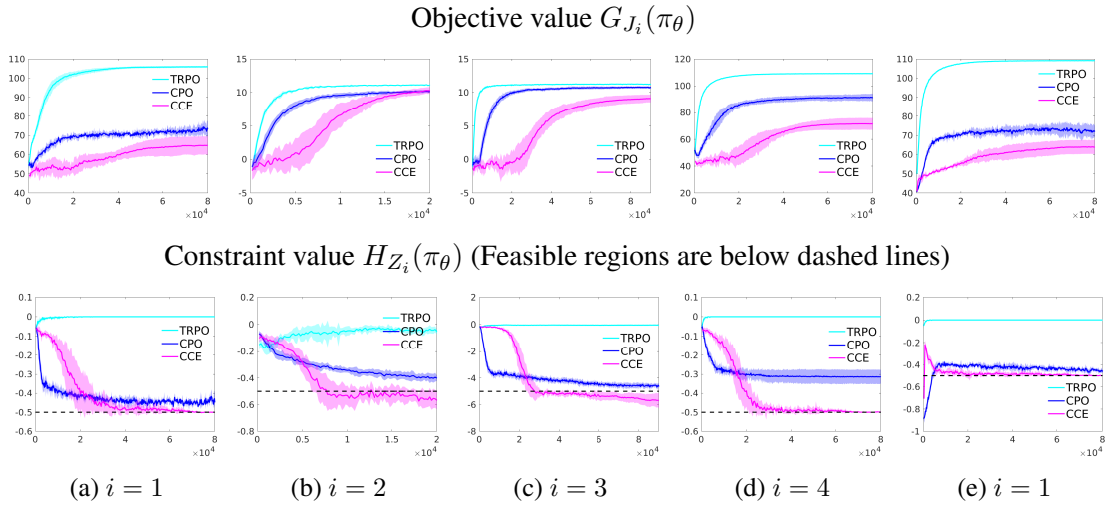
Table 4: $J_i(\tau)$, $Z_i(\tau)$ and constraint upper bound d_i for $i = 1, 2, 3, 4$, $\tau \in (S \times A)^N$.

i	$J_i(\tau)$	$Z_i(\tau)$	d_i	J_i Markovian	Z_i Markovian
1	1 for each state in $\mathcal{G}$; $2\|y\|$ for each state with $y \in [-2, -0.2]$; 0 otherwise.	-1 if the robot arrives $\mathcal{G}$ which is absorbing; 0 otherwise.	-0.5	Yes	Yes
2	30 times the minimum signed distance from any state in τ to $\mathcal{B}$.	-1 if the robot visited $\mathcal{G}$ in τ; 0 otherwise.	-0.5	No	No
3	Same as $J_2(\tau)$.	-1 for each state in $\mathcal{G}$; 0 otherwise.	-5	No	Yes
4	Same as $J_1(\tau)$.	-1 if the robot visits $\mathcal{G}$ and never visits $\mathcal{B}$; 0 otherwise.	-0.5	Yes	No

is $v_{max}\Delta t > 0$. There are $\lceil 2\pi/\omega_{max} \rceil$ uniformly distributed observation points on each circle and the robot can measure the label for each node. An observation point is labeled: 1, if it belongs to $\mathcal{G}$; -1, if it belongs to $\mathcal{B}$; and 0, otherwise. We also assume that the robot can sense the direction of the center of $\mathcal{G}$ in its local coordinate without knowing the distance. In total, there are a total of $(2 + n_s\lceil 2\pi/\omega_{max} \rceil)$ outputs of the local observation model. In our experiment, $\omega_{max} = \frac{\pi}{6}$, $n_s = 5$, so there are 62 local observations as the inputs to the policy network.

We compare the performance of CCE to trust region policy optimization (TRPO) [153], a state-of-the-art unconstrained RL algorithm, and its variant for constrained problems called constrained policy optimization (CPO). The policy space Π_Θ is a set of deterministic stationary policies. Trajectory length for all experiments is set to $N = 30$. Each sampled policy is evaluated using 10 sample trajectories. We set $\rho = 0.2$. For TRPO and CPO, we set the batch size as 6,000, the discount factor as 0.999, and the step size for trust region as 0.01. All other parameters are the default values in rllab [47].

We show the learning curves of CCE, TRPO and CPO for each experiment in Figure 10. For experiments in which J_i is not strictly positive, we use $\exp(J_i)$ instead of J_i to update the policy distributions in CCE. The vertical axes in Figure 10 show the *average* objective and constraint values



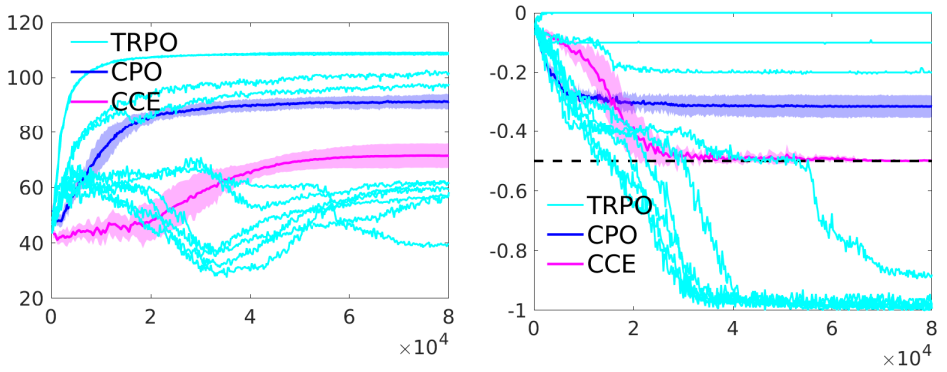
(a) $i = 1$ (b) $i = 2$ (c) $i = 3$ (d) $i = 4$ (e) $i = 1$

Figure 10: Learning curves of CCE, CPO and TRPO with different objectives G_{J_i} and constraints H_{Z_i}. The horizontal axes show the total number of sample trajectories for CCE and the total number of equivalent sample trajectories for TRPO and CPO. The vertical axes show the sample mean of the objective and constraint values of the learned policy (for TRPO and CPO) or the learned policy distribution (for CCE). The shade shows 1 standard deviation. The region below the dashed line in the second row is feasible. Each experiment is repeated for 5 times.

of the learned policy. For CCE, the average values are computed with all rollout trajectories that are simulated with *all* the policies sampled at the current iteration. For CPO and TRPO, we simulate the current policy from exactly the same set of initial states and compute the average objective and constraint values for all trajectories.

Results by TRPO show that the constraints cannot be satisfied by merely optimizing the corresponding objectives. However, CCE successfully outputs feasible policies in all experiments. On the other hand, CPO needs significantly more samples to find a single feasible policy, or simply converges to an infeasible policy especially if the constraint is non-Markovian.

One may argue that CPO is designed to work with feasible initial policies and Markovian objectives and constraints (specifically, both J and Z are discounted total rewards). Thus, we repeat the first experiment ($i = 1$) with feasible initial policies and obtain the result in the last column of Figure 10. In this case, CPO leaves the feasible region rapidly and then follows generally the same path as if it is initialized with an infeasible policy. This behavior suggests that its incapability to enforce constraint



(a) $G_{J_4}(\pi_\theta)$.

(b) $H_{Z_4}(\pi_\theta)$.

Figure 11: Average performance of CCE, CPO and TRPO for Experiment 4 with initial feasible policy.

satisfaction is not due to the lack of initial feasibility. Although CCE also leaves the feasible region at an early stage of iterations, it regains feasibility much faster than the previous case with infeasible initial polices. These results suggest that CCE is more reliable than CPO for applications where the strict constraint satisfaction is critical.

In Figure 11, we compare the performance of CPO and CCE in Experiment 4 to that of TRPO with objective $G_{J_4} - 100H_{Z_4}$. The fixed penalty coefficient 100 is chosen to be neither too large nor too small so it can show a large variety of locally optimal behaviors with very different G_{J_4}-values and H_{Z_4}-values. Figure 11 clearly shows the trade-off between G_{J_4}-values and H_{Z_4}-values, which partially explains the gap between the G_{J_4}-value outputs of CCE and CPO. With a fixed penalty coefficient, the policies learned by TRPO are either infeasible or with very small constraint values. The policy output by CCE has higher G_{J_4}-value than all the feasible policies found by TRPO and CPO.

Chapter 5: Correct-By-Synthesis Reinforcement Learning with Temporal Logic Constraints

5.1. Introduction

The goal of this paper is to synthesize optimal reactive strategies for systems with respect to some *unknown* performance criterion and in an adversarial environment such that given temporal logic specifications are satisfied. The consideration of unknown performance criterion may seem unreasonable at first sight, but it turns out to be an effective supplement to the specification as task description and suits the need in many applications. On the one hand, general requirements on system behaviors such as safety concerns and task rules may be known and expressed as specifications in temporal logic. On the other hand, quantitative performance criterion can help encode more subtle considerations, such as specific intentions for the current application scenario and personal preferences of human operators who work with the autonomous system. For a path planner of autonomous vehicles, specifications imply fixed nonnegotiable constraints like safety requirements, e.g., always drive on the correct lane, never jump the red light and eventually reach the destination. Quantitative performance criteria give preferences within the context constrained by the specifications, which may involve considerations that have not been taken into account during controller design and suggested by the human operators.

The two main topics most relevant to our work are reactive synthesis with temporal logic specifications and reinforcement learning with respect to unknown performance criteria. Neither solves the problem we consider in this paper.

On the synthesis side, early work focused on planning in static known environments [116, 178]. Reactivity to the changes in dynamic environments is a crucial functionality. For example, the environment of an autonomous vehicle involves the other vehicles and pedestrians moving nearby, and it is impractical to expect an autonomous vehicle to run on roads safely without reacting to its surrounding environment in real time. Recently, references [138, 140, 141] considered possibly

adversarial environments and reactive strategies (without any quantitative performance criteria).

Another concern in synthesis is optimality with respect to a given performance criterion. Optimal strategies have been studied with respect to given objectives while satisfying some temporal logic specifications, mostly in deterministic environments or stochastic environments with known transition distribution [46, 190]. Both qualitative objectives such as correctness guarantee with respect to an adversarial environment and quantitative objectives such as mean payoffs were studied in [36] though these results crucially rely on the quantitative measure being known a priori.

In order to deal with problems with a priori unknown performance criterion, it is intuitive to gain experience from direct interactions with the environment or with a human operator, which coincides with the motivation of many reinforcement learning methods [170]. Multiple learning methods have been studied and are available to problems with unknown reward functions and incomplete prior knowledge on system models [15, 156, 169, 181], and have been used in many applications, including the famous TD-Gammon example [173] and robot collision avoidance [77]. However, the learning process generally cannot guarantee the satisfaction of other independently imposed specifications while maximizing the expected rewards at the same time, though they can be modified to deal with some simple cases [59, 136].

To the best of our knowledge, the current paper is the first to deal with the problem of synthesizing a controller which optimizes some a priori unknown performance criterion while interacting with an uncontrolled environment in a way that satisfies the given temporal logic specifications. The approach we take is based on a decomposition of the problem into two subproblems. For the first part (Section 5.4.1), the intuition is to extract a strategy for the system, namely a permissive strategy [19], which encodes multiple (possibly all) ways in which the system can react to the adversarial environment and satisfy the specifications. Then in the second part (Section 5.4.2), we quantify the a priori unknown performance criterion as a (still unknown) reward function and apply the idea of reinforcement learning to choose an optimal strategy for the system within the operating envelope allowed by the permissive strategy. By decoupling the optimization problem with respect to the unknown cost from the synthesis problem, we manage to synthesize a strategy for the system that is

guaranteed to both satisfy the specifications and reach optimality over a set of winning strategies with respect to the a priori unknown performance criterion (Section 5.4.3).

5.2. Preliminaries

We now introduce some basic concepts.

5.2.1. Two-Player Games

First we model the setting as a two-player game. In this model we care about not only the controlled system, but also its external uncontrolled environment. Interactions between the controlled system and the uncontrolled environment play a critical role in guaranteeing the correctness of given specifications, as we will discuss later.

Definition 3. *A two-player game, or simply a game, is defined as a tuple* $\mathcal{G} = (S, S_s, S_e, I, A_c, A_{uc}, T, W)$*, where* S *is a finite set of states;* $\{S_s, S_e\}$ *is a partition of* S*, i.e.,* $S = S_s \bigcup S_e$*,* $S_s \bigcap S_e = \emptyset$*;* $I \subseteq S$ *is a set of initial states;* A_c *is a finite set of controlled actions of the system;* A_{uc} *is a finite set of uncontrolled actions for the environment and* $A_{uc} \bigcap A_c = \emptyset$*;* $T : S \times \{A_c \bigcup A_{uc}\} \to 2^S$ *is a transition function;* W *is the winning condition defined later.*

S_s and S_e are the sets of states from which it is the system's or the environment's turn to take actions, respectively. There are no available uncontrolled actions (environment actions) to any state $s \in S_s$, and correspondingly, states in S_e can not respond to any controlled action (system action). Let $A(s)$ be the set of actions available at state $s \in S$. Hence $A(s) \subseteq A_c$ if $s \in S_s$ and $A(s) \subseteq A_{uc}$ if $s \in S_e$.

If the transition function T of $\mathcal{G}$ satisfies $|T(s, a)| \leq 1$ for all $s \in S$ and $a \in A(s)$, the game is called *deterministic*; otherwise the game is called *non-deterministic*, highlighting the fact that multiple transitions are possible to some state-action pairs. We assume here that $\mathcal{G}$ is deterministic.

A *run* $\pi = s_0 s_1 s_2 \ldots$ of $\mathcal{G}$ is an infinite sequence of states such that $s_0 \in I$ and for $i \in \mathbb{N}$, there exists $a_i \in A(s_i)$ such that $s_{i+1} = T(s_i, a_i)$ ($\mathcal{G}$ is deterministic). Without loss of generality, assume that all states are reachable from I in $\mathcal{G}$.

5.2.2. Linear Temporal Logic

We use fragments of linear temporal logic (LTL) to specify the assumptions on environment behaviors and the requirements for the system. LTL can be regarded as a generalization of propositional logic. In addition to logical connectives such as conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$) and implication ($\rightarrow$), LTL also includes basic temporal operators such as next ($\bigcirc$), until ($\mathcal{U}$), derived temporal operators like always ($\Box$) and eventually ($\Diamond$), and any (nested) combination of them, like always eventually ($\Box\Diamond$).

An *atomic proposition* is a Boolean variable (or propositional variable). Suppose AP is a finite set of atomic propositions, then we can construct LTL formulas as follows: (i) Any atomic proposition $p \in AP$ is an LTL formula; (ii) given formulas φ_1 and φ_2, $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\bigcirc\varphi_1$ and $\varphi_1 \mathcal{U} \varphi_2$ are LTL formulas. A formula without any temporal operators is called a *Boolean formula* or *assertion*. A *linear time property* is a set of infinite sequences over 2^{AP}.

LTL formulas are evaluated over executions: An *execution* $\sigma = \sigma_0, \sigma_1, \sigma_2, \ldots$ is an infinite sequence of truth assignments to the variables in AP, where σ_i is the set of atomic propositions that are $True$ at position $i \in \mathbb{N}$. Let $P(\varphi)$ be the set of atomic propositions appearing in an LTL formula φ. Given φ and an execution σ, the condition that φ *holds at position* i *of* σ, denoted by $\sigma, i \models \varphi$, is constructed inductively as follows:

1. For any $p \in P(\varphi)$, $\sigma, i \models p$ iff $p \in \sigma_i$.
2. $\sigma, i \models \neg\varphi$ iff $\sigma, i \not\models \varphi$.
3. $\sigma, i \models \bigcirc\varphi$ iff $\sigma, i+1 \models \varphi$.
4. If $\varphi = \varphi_1 \wedge \varphi_2$, then $\sigma, i \models \varphi$ iff $\sigma, i \models \varphi_1$ and $\sigma, i \models \varphi_2$.
5. If $\varphi = \varphi_1 \vee \varphi_2$, then $\sigma, i \models \varphi$ iff $\sigma, i \models \varphi_1$ or $\sigma, i \models \varphi_2$.
6. If $\varphi = (\varphi_1 \rightarrow \varphi_2)$, then $\sigma, i \models \varphi$ iff $\sigma, i \models \varphi_1$ implies $\sigma, i \models \varphi_2$.

7. If $\varphi = \varphi_1 \mathcal{U} \varphi_2$, then $\sigma, i \models \varphi$ iff there exists $k \geq i$ such that $\sigma, j \models \varphi_1$ holds for all $i \leq j < k$ and $\sigma, k \models \varphi_2$.

8. $\Diamond\varphi = \mathit{True}\, \mathcal{U}\, \varphi$, $\Box\varphi = \neg\Diamond\neg\varphi$.

If $\sigma, 0 \models \varphi$, we say that φ holds on σ or σ satisfies φ, which can also be written as $\sigma \models \varphi$.

An LTL formula φ_1 is a *safety formula* if for every execution σ that violates φ_1, there exists an $i \in \mathbb{N}^+$ such that for every execution σ' that coincides with σ up to position i, σ' also violates φ_1. An LTL formula φ_2 is a *liveness formula* if for every prefix of any execution $\sigma_0, \ldots, \sigma_i$ $(i \geq 0)$, there exists an infinite execution σ' with prefix $\sigma_0, \ldots, \sigma_i$ such that $\sigma' \models \varphi_2$. Intuitively, safety formulas indicate that "something bad should never happen," and liveness formulas require that "good things will happen eventually."

Let AP be a set of atomic propositions, and define a *labeling function* $L : S \to 2^{AP}$ such that each state $s \in S$ is mapped to the set of atomic propositions that hold $True$ at state s. A *word* is an infinite sequence of labels $L(\pi) = L(s_0)L(s_1)L(s_2)\ldots$ where $\pi = s_0 s_1 s_2 \ldots$ is a run of $\mathcal{G}$. We say a run π satisfies φ if and only if $L(\pi) \models \varphi$.

To complete the definition of two-player games, define the winning condition $W = (L, \varphi)$ such that L is a labeling function and φ is an LTL formula, and a run π of $\mathcal{G}$ is *winning for the system* if and only if π satisfies φ. φ can be used to express the qualitative specifications such as system requirements and environment assumptions.

5.2.3. Control Strategies

Given the game $\mathcal{G}$, we would like to synthesize a control protocol such that the runs of $\mathcal{G}$ satisfy the specification φ.

A *(deterministic) memoryless strategy for the system* is a map $\mu : S_s \to A_c$, where $\mu(s) \in A(s)$ for all $s \in S_s$. A *(deterministic) finite-memory strategy for the system* is a tuple $\mu = (\mu_m, \rho_m, M)$ where $\mu_m : S_s \times M \to A_c$ such that $\mu_m(s, m) \in A(s)$ for all $s \in S_s$, $m \in M$, and $\rho_m : S \times M \to M$. The finite set M is called the *memory* and ρ_m is also called the *memory update function*. $\mu_m(s, m) \in A(s)$

for all $s \in S_s$ and $m \in M$. m is initialized to be $m_0 \in M$. Strategies can also be defined as *non-deterministic*, in which case μ will be defined as $\mu : S_s \to 2^{A_c}$ for memoryless strategies or $\mu = (\mu_m, \rho_m, M)$ with $\mu_m : S_s \times M \to 2^{A_c}$ for finite-memory strategies. Clearly deterministic strategies can be regarded as a special case of non-deterministic strategies when $|\mu_m(s, m)| = 1$ for all $s \in S_s, m \in M$. We require $|\rho_m(s, m)| = 1$ for all $s \in S$ and $m \in M$, no matter the strategy is deterministic or not. ρ_m will be evaluated each time after any player takes action. If we further specify the probability distribution P over $A(s)$ for each state $s \in S_s$, the corresponding strategies are called *randomized strategies*. We refer to deterministic strategies unless otherwise stated. By replacing S_s by S_e and A_c by A_{uc}, we can define memoryless and finite-memory strategy for the environment.

A run $\pi = s_0 s_1 s_2 \ldots$ is *induced by a strategy μ for the system* if for any $i \in \mathbb{N}$ such that $s_i \in S_s$, $s_{i+1} = T(s_i, \mu(s_i))$ (for memoryless strategies) or there exists an infinite sequence $m_0 m_1 m_2 \ldots$ over M such that $s_{i+1} = T(s_i, \mu_m(s_i, m_i))$ and for all $s_j \in S$, $m_{j+1} = \rho_m(s_{j+1}, m_j)$ (for finite-memory strategies). Let $R^\mu(s)$ be the set of runs of $\mathcal{G}$ induced by a strategy μ for the system and initialized with $s \in I$. $|R^\mu(s)| > 1$ when the strategies for the environment are not unique, even if μ is deterministic.

We say a strategy μ for the system *wins at state $s \in I$* if all runs $\pi \in R^\mu(s)$ are winning for the system. A strategy μ is called a *winning strategy* if it wins at all initial states of $\mathcal{G}$. A formula φ is *realizable* for $\mathcal{G}$ if there exists a winning strategy for the system with $W = (L, \varphi)$.

5.2.4. Reward Functions

Besides qualitative requirements which are encoded in the winning condition, we also consider quantitative evaluation from other sources such as the human operators. Such evaluation is modeled as a reward function which we want to maximize by choosing proper strategy for the system.

In order to evaluate the system strategy, we first map each system state-action pair to a nonnegative value by an *instantaneous reward function* $\mathcal{R} : S \times (A_c \bigcup A_{uc}) \to \mathbb{R}^+ \bigcup \{0\}$, and then consider the "accumulation" of such instantaneous rewards obtained over a run of a game $\mathcal{G}$. As runs are of infinite

length, we cannot simply add all the instantaneous reward acquired, which may approach infinity. Instead we define a *reward function* $J_{\mathcal{R}}^{\mathcal{G}} : S^{\omega} \to \mathbb{R}$ to compute reward for any run π of $\mathcal{G}$ given the instantaneous reward function $\mathcal{R}$. A common example of $J_{\mathcal{R}}^{\mathcal{G}}$ is the discounted reward

$$J_{\mathcal{R}}^{\mathcal{G}} = \sum_{k=0}^{\infty} \gamma^k r_{k+1}, \tag{5.1}$$

where γ is a discount factor satisfying $0 \leq \gamma < 1$, and r_{k+1} is the $(k+1)$th instantaneous reward obtained by the system. In this case, rewards acquired earlier are given more weight, while in other examples like the mean payoff function

$$\liminf_{k \to \infty} \frac{1}{k+1} \sum_{i=t}^{t+k} r_i,$$

weights on instantaneous reward are independent of the sequence.

Now we define a reward function $\bar{J}_{\mathcal{R}}^{\mathcal{G}} : \mathcal{P} \times I \to \mathbb{R}^{+} \bigcup \{0\}$ to evaluate each strategy for the system, where $\mathcal{P}$ is the set of system strategies. Usually $|R^{\mu}(s)| > 1$ as the uncontrolled environment has more than one strategies, and thus the definition of $\bar{J}_{\mathcal{R}}^{\mathcal{G}}(\mu, s)$ is not unique given $J_{\mathcal{R}}^{\mathcal{G}}(\pi)$ for all runs in $R^{\mu}(s)$. One commonly used choice for $\bar{J}_{\mathcal{R}}^{\mathcal{G}}$ is the expectation of $J_{\mathcal{R}}^{\mathcal{G}}(\pi)$ over all runs in $R^{\mu}(s)$ with some given distribution for the environment strategy, i.e., $\mathbb{E}_{\pi \in R^{\mu}(s)}\left[J_{\mathcal{R}}^{\mathcal{G}}(\pi)\right]$. The distribution is usually estimated from interaction experience with the environment. Another common way is to define $\bar{J}_{\mathcal{R}}^{\mathcal{G}}$ as the minimal possible reward acquired when the system strategy is μ, i.e.,

$$\bar{J}_{\mathcal{R}}^{\mathcal{G}}(\mu, s) = \inf_{\pi \in R^{\mu}(s)} J_{\mathcal{R}}^{\mathcal{G}}(\pi), \tag{5.2}$$

which we use as the reward function in our problem.

5.3. Problem Formulation

We have modeled the interaction between the uncontrolled environment and the controlled system as a two-player game whose winning condition is described by a given LTL formula. Moreover, we defined reward functions to evaluate the performance of different system strategies. Now we can go

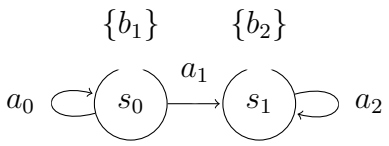


Figure 12: A game $\mathcal{G}_0$ without finite-memory optimal strategy.

on to formulate the main problem of the paper.

Problem 2. *A two-player deterministic game $\mathcal{G} = (S, S_s, S_e, I, A_c, A_{uc}, T, W)$ is given where $W = (L, \varphi)$ and φ is realizable for $\mathcal{G}$. Find a memoryless or finite-memory winning strategy μ for the system such that $\bar{J}^{\mathcal{G}}_{\mathcal{R}}(\mu, s)$ is maximized for all $s \in I$, where a reward function $\bar{J}^{\mathcal{G}}_{\mathcal{R}}$ is given with respect to an unknown instantaneous reward function $\mathcal{R} : S \times (A_c \bigcup A_{uc}) \to \mathbb{R}^+ \bigcup \{0\}$.*

Generally, there does not necessarily exist a memoryless or finite-memory winning strategy that maximizes the reward over all winning strategies, as it is possible that the instantaneous reward promotes the system to violate the specification. Take as an example $\mathcal{G}_0 = (S, S_s, S_e, I, A_c, A_{uc}, T, W)$, where $S_e = \emptyset$, $S = S_s = I = \{s_0, s_1\}, A_c = \{a_0, a_1, a_2\}, A_{uc} = \emptyset$ and $W = (L, \varphi)$. The transition function T and the labeling function L are shown in Figure 12, and the formula is $\varphi = \diamondsuit b_2$. The game $\mathcal{G}_0$ does have winning strategies for the system. For example, the strategy μ where $\mu(s_0) = a_1, \mu(s_1) = a_2$ is a memoryless winning strategy, and the strategy $\mu' = (\mu'_m, \rho'_m, \{0, 1\})$ where

$$\begin{aligned}
\mu'_m(s_1, 0) =& \mu'_m(s_1, 1) = \{a_2\},\\
\mu'_m(s_0, 0) =& \{a_0, a_1\},\\
\mu'_m(s_0, 1) =& \{a_1\},\\
\rho'_m(s_1, 0) =& 0,\\
\rho'_m(s_1, 1) =& \rho'_m(s_0, 0) = \rho'_m(s_0, 1) = 1
\end{aligned}$$

is a finite-memory winning strategy.

But $\mathcal{G}_0$ may not have memoryless or finite-memory optimal winning strategies for the system. Suppose the unknown instantaneous reward function is actually defined as $\mathcal{R}(s_1, a_2) = 0$, $\mathcal{R}(s_0, a_0) = \mathcal{R}(s_0, a_1) = 10$, and the reward function $J^{\mathcal{G}}_{\mathcal{R}}$ is the same as (5.1). In order to maximize $\bar{J}^{\mathcal{G}}_{\mathcal{R}}(\mu, \cdot)$, μ

should allow the system to stay at s_0 forever, which will violate φ. Thus optimal winning strategies need infinite memory.

Let us now move on to an overview of the two-stage solution approach we propose. Given a game $\mathcal{G}$ as in Problem 2, we first extract a non-deterministic winning strategy μ_p called a *permissive strategy* [19], which guarantees that $R^{\mu}(s) \subseteq R^{\mu_p}(s)$ for all memoryless winning strategies μ and $s \in I$. In some special cases (e.g. the conditions in Proposition 2), we are even able to compute a *maximally permissive strategy* μ_p^{max}, such that $R^{\mu}(s) \subseteq R^{\mu_p^{max}}(s)$ for all winning strategies μ and $s \in I$. Then in the second stage we restrict to the transitions allowed by μ_p (or μ_p^{max}), apply reinforcement learning methods to explore the a priori unknown instantaneous reward function $\mathcal{R}$ and compute an optimal strategy over all strategies of the new game obtained in the first stage. With this decomposition we managed to separate the problem of guaranteeing the correctness of specifications from that of seeking the optimal reward with a priori unknown instantaneous rewards.

5.4. Permissive Strategies, Learning and the Main Algorithm

This section is composed of three parts. We first introduce the idea of permissive strategies, then describe a reinforcement learning method which is used to learn an optimal strategy with respect to an unknown reward function without concern about any specification, and finally combine the two parts to apply the reinforcement learning method to explore for an optimal strategy out of those encoded in an appropriately constructed permissive strategy.

5.4.1. Extraction of Permissive Strategies

We first introduce an inclusion relation between strategies. Recall that we have defined the set of runs induced by a strategy μ for the system with initial state $s \in I$ as $R^{\mu}(s)$. For two non-deterministic strategies μ_1 and μ_2 for the system, we say that μ_1 *includes* μ_2 if $R^{\mu_2}(s) \subseteq R^{\mu_1}(s)$ holds for all $s \in I$. Furthermore, if μ_1 includes μ_2 and μ_2 includes μ_1, we call μ_1 and μ_2 *equivalent*. In other words, equivalent strategies induce the same set of runs. A game $\mathcal{G}$ has a *unique* winning strategy if all its winning strategies are equivalent. Now we can define permissive strategies based on this strategy inclusion relation.

Definition 4. *Given a two-player game $\mathcal{G}$, a non-deterministic strategy μ for the system is called permissive if (i) it is winning for the system and (ii) includes all memoryless winning strategies for the system. A permissive strategy is called maximally permissive if it includes all winning strategies for the system.*

All two-player games have permissive strategies. For games with finite states, there are only finitely many memoryless winning strategies. We can build a permissive strategy by adding a unique tag to each of them (as memory) and directly combining them together. In cases where there is no memoryless winning strategy, this fact is trivial as any winning strategy is permissive.

In general, permissive strategies are not necessarily unique. For example, the game $\mathcal{G}_0$ in Figure 12 has a unique memoryless winning strategy μ for the system where $\mu(s_0) = a_1$, $\mu(s_1) = a_2$. As a result, μ_p such that $\mu_p(s_0) = \{a_1\}$ and $\mu_p(s_1) = \{a_2\}$ is a deterministic memoryless permissive strategy. In the meantime, the finite-memory strategy $\mu' = (\mu'_m, \rho'_m, \{0, 1\})$ where

$$\begin{aligned}
\mu'_m(s_1, 0) =& \mu'_m(s_1, 1) = \{a_2\},\\
\mu'_m(s_0, 0) =& \{a_0, a_1\},\\
\mu'_m(s_0, 1) =& \{a_1\},\\
\rho'_m(s_0, 0) =& \rho'_m(s_0, 1) = \rho'_m(s_1, 1) = 0,\\
\rho'_m(s_1, 0) =& 0
\end{aligned}$$

includes μ and thus is also a permissive strategy. As μ_p does not include μ', they are different permissive strategies of $\mathcal{G}$.

On the other hand, maximally permissive strategies must be unique by definition, if they exist for a game $\mathcal{G}$. The specific representations of maximally permissive strategies may not be unique, just like a memoryless strategy can be rewritten as a finite-memory strategy in which the allowed actions are independent of the memory.

It is naturally desirable to extract maximally permissive strategies as they include all the other winning strategies. While they do not exist in general, the following proposition is a sufficient

condition of their existence.

Proposition 1 ([19])**.** *All games $\mathcal{G}$ with winning conditions $W = (L, \varphi)$ in which φ is a safety formula have memoryless or finite-memory maximally permissive strategies.*

This characterization can be extended to be both sufficient and necessary. It has been shown that maximally permissive strategies exist if and only if the winning conditions are *reactive safety properties* [49], which are equivalent to safety properties when the interaction between the environment and system is explicitly considered. Reactive safety characterizes precisely the properties whose satisfaction is checked by testing if the runs of $\mathcal{G}$ satisfy some safety formula.

There exists work on the construction of permissive strategies for games $\mathcal{G}$ with a general LTL formula φ in the winning condition [19, 166], so we only sketch the relevant results briefly here. The first step is to compute a *deterministic parity automaton* [174] from φ, which is taken into account by constructing a new game $\mathcal{G}'$ with a *parity winning condition*. Games with such a winning condition have permissive strategies and Bernet et al. [19] provided an algorithm to compute such strategies.

Additionally, Ehlers and Finkbeiner [49] offer a method for checking if the winning condition W is a reactive safety property for $\mathcal{G}$. The game $\mathcal{G}$ is first translated into a parity automaton as before, and is then used to construct a parity tree automaton [174]. Tree automata are commonly used to explicitly model inputs and outputs and the overall behavior of reactive systems. For reactive safety properties, trees get rejected if and only if some path in the tree visits some violating states, i.e., the states from which all trees are rejected. In other words, the set of accepted trees should be exactly those that never visit any violating state in all paths. The acceptance of trees can be decided by simply checking the set of states they can visit. The problem of checking if φ is a reactive safety property for $\mathscr{A}$ is reduced to checking the equivalence of two parity tree automata, which can be solved with existing approaches [82]. If we get a positive answer, we can construct another game with a safety formula in its winning condition which accepts exactly the same set of runs as $\mathscr{A}$. By Proposition 1, there exists a maximally permissive strategy for $\mathcal{G}$. The worst-case complexity of the resulting method is 2-EXPTIME.

Although Proposition 1 guarantees the existence of a maximally permissive strategy μ_p^{max} when φ is a safety formula, the computational time complexity is the same as that of synthesizing a strategy for a game with a general LTL formula [99]. The complexity can be significantly improved when φ is of the following special form. The proof is straightforward and is omitted due to the limited space.

Proposition 2. *For all games $\mathcal{G}$ with winning condition $W = (L, \varphi)$ and $\varphi = \varphi_0 \wedge \Box\varphi_1$, where φ_0 and φ_1 are Boolean formulas of p and $\bigcirc q$ for $p, q \in AP$, a memoryless maximally permissive strategy can be solved in linear time of the number of transitions of $\mathcal{G}$ and the size of φ.*

We use the software tool `slugs` [50] to extract permissive strategies when φ in $\mathcal{G}$ is in the form of generalized reactivity (1) (GR(1)) [138]. Under the condition of Proposition 2, `slugs` synthesizes a maximally permissive strategy.

The extraction and application of permissive strategies greatly simplify the solution of Problem 2, enabling us to focus on optimizing the performance over strategies known to be correct. By Definition 4, a permissive strategy μ_p is non-deterministic and thus its application to a game $\mathcal{G}$ is essentially encoding its memory update function into the game structure and removing all transitions that it does not allow. Hence any run π' of the resulting game $\mathcal{G}'$ has a unique counterpart π in the runs of $\mathcal{G}$ induced by μ_p, and vice versa. Moreover, such π and π' can only be winning for the system simultaneously. Since μ_p is winning for the system in $\mathcal{G}$, all runs it induces are winning for the system and so are their counterpart runs in $\mathcal{G}'$. As a result, any strategy μ' of $\mathcal{G}'$ is winning for the system. Let $J_{\mathcal{R}}^{\mathcal{G}'}(\pi')$ be the same as $J_{\mathcal{R}}^{\mathcal{G}}(\pi)$, and $\bar{J}_{\mathcal{R}}^{\mathcal{G}'}$ is defined similarly as $\bar{J}_{\mathcal{R}}^{\mathcal{G}}$.

5.4.2. Reinforcement Learning

Now that we have acquired a game $\mathcal{G}'$ whose runs are all guaranteed to be correct with respect to the underlying linear temporal logic specification, we can move on to learn an optimal strategy with respect to an a priori unknown instantaneous reward function $\mathcal{R}$. The reinforcement learning algorithm aim to maximize $\bar{J}_{\mathcal{R}}^{\mathcal{G}'}$ for the game $\mathcal{G}'$.

The choice of reinforcement learning algorithms depends on the choice of the reward function $\bar{J}_{\mathcal{R}}^{\mathcal{G}}$ in Problem 2, regardless of how the permissive strategy μ_p is generated. Here we focus on discounted

reward functions, but the pseudo-algorithm in Section 5.4.3 also works with other forms of $\bar{J}_{\mathcal{R}}^{\mathcal{G}'}$ so long as there exists an optimal deterministic winning strategy μ' which can be solved by the corresponding reinforcement learning method.

The discounted reward function for evaluating the rewards obtained by a run is shown in (5.1). We particularly focus on the minimal (worst-case) possible reward for each system strategy, as shown in (5.2). This definition concerns about the tight lower bound of the reward obtained by executing strategy μ' whatever strategy the uncontrolled environment implements. In other words, we assume that the environment acts adversarially and the game is equivalently a zero-sum game. It has been shown that in this case both the environment and the system have deterministic memoryless optimal strategies in $\mathcal{G}'$ [157]. As a result we can neglect all randomized strategies without loss of optimality. With proper assumptions on the game structure, such an optimal strategy can be computed by the maximin-Q algorithm, which is a simple variation of the minimax-Q learning algorithm [110], or by the generalized Q-learning algorithm for alternating Markov games [112]. Both methods guarantee that the learned greedy strategy, which always chooses an action with the best learned Q value, converges to an optimal strategy for a system interacting with an adversary under some common convergence conditions.

5.4.3. Connecting the Dots: Correct-By-Synthesis Learning

Having discussed permissive strategies and reinforcement learning, we are now ready to connect the pieces and discuss a solution to Problem 2, which is outlined in Algorithm 3. Maximally permissive strategies play a special role as they include all winning strategies for the system, and their existence naturally divide the solution into two cases.

For games whose maximally permissive strategies can be computed

If maximally permissive strategies can be computed for a game $\mathcal{G}$, μ_p in Algorithm 3 includes all winning strategies and is a winning strategy itself. Applying it to $\mathcal{G}$ not only guarantees winning for the system but also preserves all winning strategies for the system in all subsequent steps, which decouples the correctness requirements from optimality concerns. As the output of the reinforcement

Algorithm 3 Pseudo-algorithm for solving Problem 2

Require: A game $\mathcal{G} = (S, S_e, S_s, I, A_c, A_{uc}, T, W)$ with $W = (L, \varphi)$ in which φ is a realizable formula for $\mathcal{G}$, a reward function $J_{\mathcal{R}}^{\mathcal{G}}$ and $\bar{J}_{\mathcal{R}}^{\mathcal{G}}$ (e.g. as in (5.1), (5.2)) with respect to an unknown instantaneous reward function $\mathcal{R}$.

Ensure: A winning strategy μ for the system that maximizes $\bar{J}_{\mathcal{R}}^{\mathcal{G}}(\mu, s)$ for all $s \in I$.

1: Compute a (maximally) permissive strategy μ_p.
2: Apply μ_p to $\mathcal{G}$ and modify $\mathcal{G}$ into a new game $\hat{\mathcal{G}} = (\hat{S}, \hat{S}_s, \hat{S}_e, \hat{I}, A_c, A_{uc}, \hat{T}, \hat{W})$, where $\hat{W} = (L, True)$.
3: Compute $\hat{\mu}^*$ that maximizes $\hat{J}_{\mathcal{R}(\mu,s)}^{\hat{\mathcal{G}}}$ for all $s \in I$ with some reinforcement learning algorithm (e.g. the maximin-Q algorithm).
4: Map $\hat{\mu}^*$ in $\hat{\mathcal{G}}$ back to μ^* in $\mathcal{G}$.
5: **return** μ^*.

learning algorithm used in Step 3 is guaranteed to converge to an optimal deterministic winning strategy, the output of Algorithm 3 is guaranteed to be a solution of Problem 2. Theorem 5 summarizes this result in a special case.

Theorem 5. *If the conditions in Proposition 2 hold, the output of Algorithm 3 is a solution to Problem 2.*

For games whose maximally permissive strategies cannot be computed

If maximally permissive strategies for a game $\mathcal{G}$ are not solvable, the best we can expect is to extract a permissive strategy which includes a proper subset of winning strategies for the system. There can be many permissive strategies for the same game with different "permissiveness", i.e., including different subsets of winning strategies. For two different permissive strategies μ_1 and μ_2 for the system, if μ_2 includes μ_1, intuitively μ_2 would be more "permissive" and have higher worst-case reward, although it is also expected to consume more computation resources. Thus there is a natural trade-off between "permissiveness" and optimality for the solution of this case, which is illustrated in Section 5.5.

5.5. Experimental Results

We demonstrate the use of Algorithm 3 on robot motion planning examples in grid worlds with different sizes and winning specifications. The game in the first example has a maximally permissive strategy for the system as its specification is a safety formula, while for the second example we can at most compute a permissive strategy. The last example shows the trade-off between the performance of the learned system strategy of Algorithm 3 and the computation cost.

Example 1. Two robots, namely a system robot and an environment robot, move in an N-by-N square grid world strictly in turns. It is known that the two robots are in different cells initially and at each move, the environment robot must go to an adjacent cell, while the system robot can either go to an adjacent cell or stay in its current cell. The system robot should always avoid collision with the environment robot. Assume that the positions of both robots are always observable for the system.

This problem can be formulated as a game $\mathcal{G} = \{S, S_s, S_e, I, A_c, A_{uc}, T, W\}$ with $W = (L, \varphi_0)$. Let $Pos = \{0, \ldots, N^2 - 1\}$ be the set of cells in the map. Then

$$\begin{aligned} S =& Pos \times Pos \times \{0, 1\}, \\ S_s =& Pos \times Pos \times \{1\}, \\ S_e =& Pos \times Pos \times \{0\}, \\ I =& \{(x, y, 1) \mid x, y \in Pos, x \neq y\}, \\ A_c =& \{up_s, down_s, left_s, right_s, stay_s\}, \\ A_{uc} =& \{up_e, down_e, left_e, right_e\}. \end{aligned}$$

The transition function T guarantees that A_c and A_{uc} only change the first and second component of a state respectively. The set of atomic propositions is

$$AP = \Big(\bigcup_{i=0}^{N^2-1} x_i \Big) \cup \Big(\bigcup_{j=0}^{N^2-1} y_j \Big) \cup \{t_0, t_1\}.$$

The labeling function is $L(s) = \{x_i, y_j, t_k\}$ if $s = (i, j, k) \in S$. The requirements on the system

Table 5: Results for example 1.

N	t_e [s]	t_l [s]	Iterations	$\lvert\hat{S}\rvert$	$\lvert\hat{S}_s\rvert$
3	0.10	4.28	9×10^4	120	72
4	0.21	16.35	3.2×10^5	432	240
5	2.20	43.12	8.5×10^5	1120	600
6	19.40	88.69	1.81×10^6	2400	1260
8	30.29	305.77	6.05×10^6	7840	4032
10	300.00	771.73	1.562×10^7	19440	9900

robot can be expressed as

$$\varphi_0 = \bigwedge\nolimits_{i=0}^{N^2-1}(\neg x_i \vee \neg y_i) \wedge \square \bigwedge\nolimits_{i=0}^{N^2-1}(x_i \rightarrow \neg y_i).$$

Proposition 2 asserts that we can compute a maximally permissive strategy and construct $\hat{\mathcal{G}}$. By Theorem 5, Algorithm 3 is expected to output an optimal strategy for the system.

The reward functions $J_{\mathcal{R}}^{\mathcal{G}}$ and $\bar{J}_{\mathcal{R}}^{\mathcal{G}}$ are given as (5.1) and (5.2), with the discounting factor γ set to be 0.9. However, the instantaneous reward function $\mathcal{R}$ is a priori unknown to the system robot. In practical scenarios $\mathcal{R}$ is often given by some independent human operator or trainer of the system robot for unpredictable purposes with arbitrarily complicated structure and thus can neither be acquired nor be guessed ahead of time. For this numerical example, $\mathcal{R}$ is set to encourage the system robot to reach positions diagonal to the environment robot's position as often as possible. From a state $s \in S_s$, $\mathcal{R}(s,a) = 1$ if the two robots are diagonal to each other at $T(s,a)$; $\mathcal{R}(s,a) = 0$ for all other (s,a). But this information is not available to the system robot in advance and is only revealed through the learning process. The system robot can only get an instantaneous reward each time when it takes a corresponding transition.

The results for the cases when $N = 3, 4, 5, 6, 8, 10$ are shown in Table 5, where t_e is the time [s] spent extracting a maximally permissive strategy μ_p^{max} with `slugs`, and t_l is the time [s] used to learn an optimal strategy $\hat{\mu}^*$. The number of states and state-action tuples are for the game $\hat{\mathcal{G}}$ in Algorithm 3. All examples run on a laptop with a 2.4GHz CPU and 8GB memory.

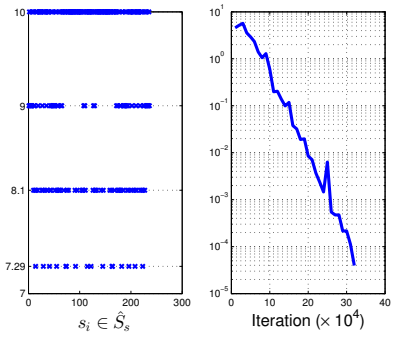


Figure 13: Results for Example 1 for $N = 4$: (Left) $\bar{J}^{\hat{\mathcal{G}}}_{\mathcal{R}}(\hat{\mu}, \hat{s})$ for all $\hat{s} \in \hat{S}_s$ and the learned greedy strategy $\hat{\mu}$; (right) the logarithm of the maximal change in V in every 10^4 iterations.

Now we illustrate the optimality of the learned greedy policy with the simulation result when $N = 4$, whose result is shown in Figure 13. Let $\hat{\mu}$ be the greedy strategy of the system learned by the maximin-Q learning algorithm against an adversarial environment. If from a state $\hat{s} \in \hat{S}_s$ the system robot can only reach a diagonal position with respect to the position of the environment in at least $k \in \mathbb{N}$ steps, $\bar{J}^{\hat{\mathcal{G}}}_{\mathcal{R}}(\hat{\mu}', \hat{s})$ is upper bounded by $\sum_{l=k}^{\infty} \gamma^l \cdot 1 = \frac{1}{1-\gamma}\gamma^k$ for any system strategy $\hat{\mu}'$. By definition, if $\hat{\mu}^*$ is an optimal strategy for the system against an adversarial environment, we have

$$\bar{J}^{\hat{\mathcal{G}}}_{\mathcal{R}}(\hat{\mu}', \hat{s}) \leq \bar{J}^{\hat{\mathcal{G}}}_{\mathcal{R}}(\hat{\mu}^*, \hat{s}) \leq \frac{1}{1-\gamma}\gamma^k.$$

In this 4-by-4 case, the system can always reach a diagonal position in 3 steps. Figure 13 shows that V converges to the values 10, 9, 8.1 and 7.29, which coincide with $\frac{1}{1-\gamma}\gamma^k$ when $k = 0, 1, 2, 3$ and $\gamma = 0.9$. Thus by the inequality above, $\bar{J}^{\hat{\mathcal{G}}}_{\mathcal{R}}(\hat{\mu}, \hat{s})$ also coincides with $\bar{J}^{\hat{\mathcal{G}}}_{\mathcal{R}}(\hat{\mu}^*, \hat{s})$, indicating that $\hat{\mu}$ itself is an optimal strategy of the system, as predicted by Algorithm 3.

Example 2. Now we construct a new game $\mathcal{G}_1$ with a new winning condition $W_1 = (L, \varphi_1)$ from $\mathcal{G}$ by adding liveness assumptions to the environment robot and liveness requirements to the system robot. To be more specific, we require the system robot to visit the upper left corner (cell $N^2 - N$) and the lower right corner (cell $N - 1$) infinitely often, provided that the environment robot visits the lower left corner (cell 0) and the upper right corner (cell $N^2 - 1$) infinitely often. $\mathcal{G}_1$ is the same as

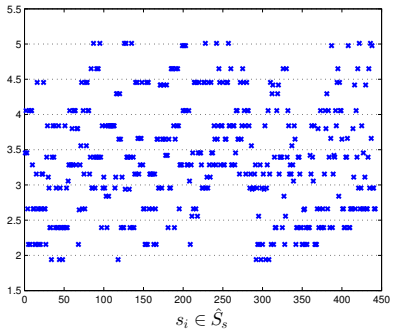


Figure 14: Result for Example 2 when $N = 4$: $\bar{J}_{\mathcal{R}}^{\hat{\mathcal{G}}_1}(\hat{\mu}, \hat{s})$ for all $\hat{s} \in \hat{S}_s$ and a learned greedy strategy $\hat{\mu}$.

$\mathcal{G}$ except that

$$\varphi_1 = \varphi_0 \wedge \big((\square\diamondsuit x_0 \wedge \square\diamondsuit x_{N^2-1}) \rightarrow (\square\diamondsuit y_{N-1} \wedge \square\diamondsuit y_{N^2-N})\big).$$

The definition of the instantaneous function $\mathcal{R}$ remains the same as in Example 1, and the learning result of $\bar{J}_{\mathcal{R}}^{\hat{\mathcal{G}}_1}(\hat{\mu}, \hat{s})$ when $N = 4$ is given as Figure 14. With this specification the system has no maximally permissive strategies, and it is expected that the true value of $\bar{J}_{\mathcal{R}}^{\hat{\mathcal{G}}_1}(\hat{\mu}, \hat{s})$ should be almost the same as $\bar{J}_{\mathcal{R}}^{\hat{\mathcal{G}}}(\hat{\mu}^*, \hat{s})$, as the system robot is allowed to follow $\hat{\mu}^*$ for as many finite moves as desired. However, Figure 14 shows that $\bar{J}_{\mathcal{R}}^{\hat{\mathcal{G}}_1}(\hat{\mu}, \hat{s})$ is smaller than $\bar{J}_{\mathcal{R}}^{\hat{\mathcal{G}}}(\hat{\mu}^*, \hat{s})$, indicating a sub-optimality due to the loss of some winning strategies by the permissive strategy.

Example 3. We now illustrate the trade-off between the performance of the learned strategy and the computation cost in Algorithm 3. Consider a new game $\mathcal{G}_2$ with winning condition $W_2 = (L, \varphi_2)$ which is slightly different from the game $\mathcal{G}$ of Example 1 as it also requires the system robot to visit one of two given cells (say cell $N^2 - N$ and cell $N - 1$) infinitely often. In other words,

$$\varphi_2 = \varphi_0 \wedge \square\diamondsuit(y_{N^2-N} \vee y_{N-1}),$$

which is in the form of GR(1). We compute a memoryless permissive strategy μ_2 for $\mathcal{G}_2$.

Now we design a sequence of games $\mathcal{G}_2^1$ to $\mathcal{G}_2^6$ from $\mathcal{G}$ in the following way. For each game we

add a counter as a new controlled state variable which counts the number of the system's moves since its last visit to cell $N^2 - N$ or cell $N - 1$, and the maximum allowed counter value increases monotonically from $\mathcal{G}_2^1$ to $\mathcal{G}_2^6$. The value of each counter should always be less than its corresponding maximum value. All these 6 games satisfy the condition in Proposition 2 and we can extract a maximally permissive strategy for each of them. With the counters, the system robot is forced to visit cell $N^2 - N$ or cell $N - 1$ infinitely often and as a result, any permissive strategies of any game in this sequence is also a permissive strategy for $\mathcal{G}_2$. Let μ_2^i be the extracted maximally permissive strategy of the game $\mathcal{G}_2^i$ for $i = 1, \cdots, 6$. By definition of maximally permissive strategies, μ_2^i includes μ_2^j if $i > j$, $i, j \in \{1, \cdots, 6\}$. In this way we extracted a sequence of permissive strategies with increasing permissiveness for the game $\mathcal{G}_2$.

We proceed the same learning procedure as the previous two examples on $\mathcal{G}_2$ and the game sequence from $\mathcal{G}_2^1$ to $\mathcal{G}_2^6$. For the 3-by-3 case, the maximum allowed counter values and the maximum values of the learned discounted reward are shown in Table 6. It is shown that the maximum discounted reward, which can be seen as the performance of the learned system strategy, increases monotonically with the maximum counter value, i.e., the permissiveness of the permissive strategy. In the meantime, the number of learning iterations and computation time grows. This illustrates the trade-off between the performance of the learned strategy and the computation cost.

Table 6: Results for example 3 (for the 3-by-3 case).

Strategy	Max counter value	Max discounted reward	Learning time [s]	Learning iterations [$\times 10^4$]
μ_2	N/A	5.7368	9.87	20
μ_2^1	4	8.0922	9.70	19
μ_2^2	6	8.8658	16.00	33
μ_2^3	8	9.2442	27.34	55
μ_2^4	10	9.4647	41.54	83
μ_2^5	14	9.7034	83.88	172
μ_2^6	20	9.8616	275.88	534

Chapter 6: Probably Approximately Correct Learning in Stochastic Games with Temporal Logic Specifications

6.1. Introduction

Reinforcement learning (RL) is a class of methods that allows agents to learn how to implement tasks through interaction with their environments. In general, tasks are specified using reward functions. The goal of RL is to maximize some reward-based objective function, for which some commonly used examples are the expected discounted reward and the expected average reward.

In this paper, we focus ourselves to the expected discounted-sum objective. The underlying dynamical system is modeled as a two-player zero-sum turn-based stochastic game, where the environment to be interacted with is considered as an uncontrolled adversarial player. It is well known that for both (single-player) Markov decision processes (MDPs) and (multi-player) stochastic games with discounted-sum objectives, deterministic memoryless strategies suffice for optimality [54]. Such a property plays several roles in the interpretation of the discounted-sum objective as a task description: On the one hand, it significantly simplifies the learning problem; on the other hand, it implies that discounted-sum objectives cannot be used to encode tasks that require memory. As memoryless strategies are sufficient to achieve optimality, agents lack the incentive to learn the more complicated finite-memory strategies.

Besides the memoryless property, the discounted-sum objective also suffers from some general limitations when using reward functions to specify tasks. For example, it cannot restrict the exploration behavior during the learning process. With rewards, the agent can only figure out the preferable actions after it tries all transitions, even the fatal ones such as crashing into some obstacle, which is obviously unacceptable. Also, there is usually a lack of theoretical guarantee that any strategy solved with the given reward function is desirable, except in some simple scenarios. For example, multi-dimensional reward functions are generally necessary to represent the conjunction of several requirements, in which case a strategy usually cannot be simultaneously optimized with every single

reward. It is hard to know intuitively from the reward function how different the learned strategy is from a desired one.

In order to compensate for these problems, we propose to use linear temporal logic (LTL) specifications to complement the task description. Practically, it is relatively straightforward to extract LTL specifications from high-level task requirements in robot planning and control [66, 93, 165, 191]. Algorithmically, all LTL formulas can be transformed to deterministic Rabin or parity automata (DRA or DPA), which can be further used to construct *product* stochastic Rabin or parity games. Strategies synthesized for such product Rabin or parity games are guaranteed to satisfy the corresponding LTL specifications with probability one (also called *almost surely*), treating LTL specifications as 'game rules' that should never be violated. Both the construction of DRA or DPA from LTL formulas and the synthesis can be performed using off-the-shelf tools [22, 57, 61, 89, 175]. Although it has been shown that deterministic memoryless strategies suffice for almost-sure winning in the product stochastic Rabin or parity games [34, 35], these strategies use memory in the original stochastic games. In this way, LTL specifications offer a systematic way of designing the memory for the desired strategies. We will show later that with the pre-computation of almost-sure winning regions in the product games, we can keep the agent safe even during the learning procedure.

In this paper we use both discounted-sum objectives and LTL specifications to encode task requirements. In particular, if an LTL specification is *realizable* (that is, there exists an almost-sure winning strategy for the agent) and can be transformed into a deterministic Büchi automaton (DBA), we prove the existence of a memoryless strategy which is both almost-sure winning with respect to the Büchi objective and ε-optimal with respect to the discounted-sum objective. We also propose a probably approximately correct (PAC) algorithm to learn such a strategy online when the reward function and the transition distributions are both unknown a priori [59, 167]. To the best of our knowledge, this is the first PAC learning algorithm for stochastic games with independent quantitative and qualitative objectives.

6.2. Related Work

The problem of strategy synthesis for two-player games with both qualitative and quantitative objectives has been extensively studied in the last decade. In many cases, the qualitative objective is to satisfy an ω-regular property, such as a parity condition or an LTL specification. The quantitative objective is to optimize some reward-based objective function, such as the expected discounted reward or the expected average reward. Examples include mean-payoff parity games [36], energy parity games [31], their extensions to multi-dimensional objectives [37] and stochastic games [32, 38]. There are also results on strategy synthesis in stochastic games with total reward constraints and LTL specifications [40].

Strategy synthesis methods generally require accurate knowledge of input games, such as transition distributions and reward functions. If such information is unavailable, exploration and learning techniques are necessary to ensure the high quality of output strategies. The problem of *learning* strategies with both qualitative and quantitative objectives is still relatively new. With qualitative objectives modeled as ω-regular properties, most existing work only works with MDPs instead of stochastic games, such as to maximize the probability to reach a given target set [29] or to satisfy a given temporal logic specification [59, 70] in a partially unknown MDP. Note that there are even no quantitative objectives in these works. For a special case where the qualitative objective is modeled as a safety property, Junges et al. [81] propose to restrict the exploration behavior with a pre-computed permissive strategy for MDPs with unknown rewards and known transitions. Alshiekh et al. [7] address a similar problem with unknown rewards by synthesizing a reactive system called a shield, which monitors the actions that the agent plans to take and makes corrections if a planned action leads to violations of the given specification. Recently, Kretínskỳ et al. [95] proposed an algorithm to maximize the mean-payoff value while satisfying a parity objective with probability 1 in MDPs with unknown probabilistic transition function and unknown reward function, which is similar to the topic of this paper but is for MDPs.

Only limited work has been done for two-player games. Our previous work [185] shows how to learn optimal strategies for deterministic games with unknown rewards and GR(1) specifications.

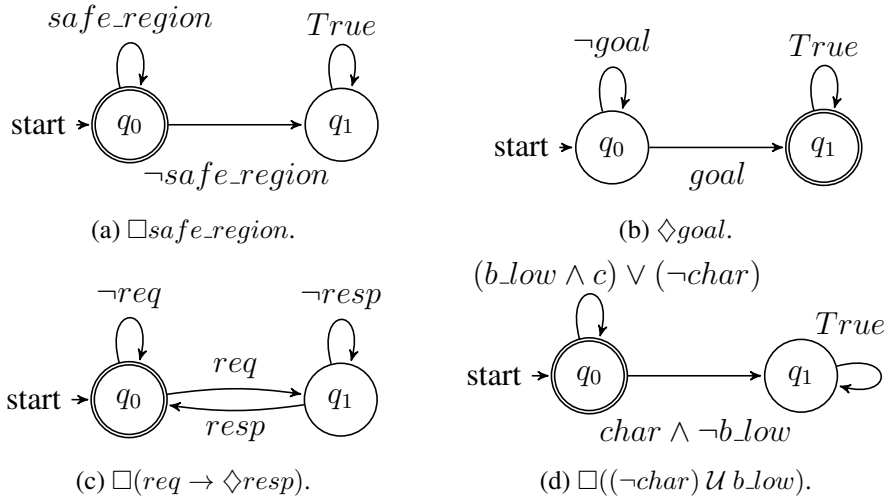
However, optimality can only be guaranteed if the specification encodes a safety property. In this paper, we consider a more general problem of learning near-optimal strategies for stochastic games with a deterministic Büchi objective, unknown rewards and unknown transition distributions. We extend a previous version of this paper [182] to better explain the high-level intuition behind our algorithms and show full proofs to our theoretical results.

6.3. Preliminaries

For any countable set M, let $|M|$ be its cardinality, M^ω be the set of infinite sequences composed of elements in M, and $\mathcal{D}(M)$ be the set of all probability distributions over M.

Turn-Based Labeled Stochastic Game. We first formulate the interaction between the agent and its environment as a turn-based labeled stochastic game. A *turn-based labeled stochastic game* between the controlled agent (the 'system') and the uncontrolled agent (the 'environment') is defined as a tuple $\mathcal{G} = \langle S_\mathcal{G}, S_{s,\mathcal{G}}, S_{e,\mathcal{G}}, I_\mathcal{G}, A_\mathcal{G}, T_\mathcal{G}, \mathcal{R}_\mathcal{G}, AP, L_\mathcal{G}\rangle$, where $S_\mathcal{G}$ is a finite state space; $S_{s,\mathcal{G}} \subseteq S_\mathcal{G}$ is the set of states at which the system chooses actions; $S_{e,\mathcal{G}} = S_\mathcal{G} \backslash S_{s,\mathcal{G}}$ is the set of states at which the environment chooses actions; $I_\mathcal{G} \subseteq S_\mathcal{G}$ is a set of initial states; $A_\mathcal{G}$ is a finite action space; $T_\mathcal{G} : S_\mathcal{G} \times A_\mathcal{G} \to \mathcal{D}(S_\mathcal{G})$ is a transition function; $\mathcal{R}_\mathcal{G} : S_\mathcal{G} \times A_\mathcal{G} \times S_\mathcal{G} \to \mathbb{R}^{\geq 0}$ is a non-negative reward function; AP is a finite set of atomic propositions (Boolean variables); $L_\mathcal{G} : S_\mathcal{G} \to 2^{AP}$ is a labeling function. For any $s, s' \in S_\mathcal{G}$ and $a \in A_\mathcal{G}$, we use $T_\mathcal{G}(s' \mid s, a)$ to represent the probability of transiting to s' by taking action a at state s. We assume that the game is zero-sum and the reward function $\mathcal{R}_\mathcal{G}$ is for the system. The reward function for the environment is exactly $-\mathcal{R}_\mathcal{G}$.

Deterministic Büchi Automata from LTL Specifications. LTL specifications put restrictions on the label sequences corresponding to the infinite state sequences of $\mathcal{G}$. Interested readers may refer to [13] for the detailed syntax and semantics of LTL. Instead of treating LTL specifications directly as formulas, we translate them into ω-regular automata, or deterministic Büchi automata, to be precise. Only a subclass of LTL formulas can be transformed into equivalent DBAs, but this subclass of specifications covers a wide range of task requirements. For example, $\square safe\_region$ (always stay in states labeled as 'safe_region'), $\diamondsuit goal$ (eventually reach a state labeled as 'goal'),



(a) $\Box safe\_region$.

(b) $\Diamond goal$.

(c) $\Box(req \rightarrow \Diamond resp)$.

(d) $\Box((\neg char)\, \mathcal{U}\, b\_low)$.

Figure 15: DBA constructed for some example specifications. Accepting states for each DFA are marked with double circles.

$\Box(request \rightarrow \Diamond response)$ (if a 'request' state is visited, a 'response' state should be visited later), $\Box((\neg charge)\, \mathcal{U}\, battery\_low)$ (never charge yourself before the battery gets low), just to name a few.

A *deterministic Büchi automaton (DBA)* is a tuple $\mathcal{A} = \langle Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \delta_{\mathcal{A}}, Q_{0,\mathcal{A}}, F_{\mathcal{A}} \rangle$ where $Q_{\mathcal{A}}$ is a finite set of states; $\Sigma_{\mathcal{A}}$ is a finite input alphabet; $\delta_{\mathcal{A}} : Q_{\mathcal{A}} \times \Sigma_{\mathcal{A}} \rightarrow Q_{\mathcal{A}}$ is a deterministic transition function; $Q_{0,\mathcal{A}} \subseteq Q_{\mathcal{A}}$ is a set of initial states; $F_{\mathcal{A}} \subseteq Q_{\mathcal{A}}$ is a set of accepting states. A *run* of $\mathcal{A}$ over an input sequence $(p_t)_{t \in \mathbb{N}} \in \Sigma_{\mathcal{A}}^{\omega}$ is an infinite sequence $(q_t)_{t \in \mathbb{N}} \in Q_{\mathcal{A}}^{\omega}$, where $q_0 \in Q_{0,\mathcal{A}}$ and $q_{t+1} = \delta_{\mathcal{A}}(q_t, p_t)$ for all $t \in \mathbb{N}$. A run $(q_t)_{t \in \mathbb{N}}$ is *accepted* by $\mathcal{A}$ if $|\{t \in \mathbb{N} : q_t \in F_{\mathcal{A}}\}| = \infty$. Some example DBAs corresponding to the example specifications in the last paragraph are illustrated in Figure 15.

Turn-Based Stochastic Büchi Game. If $\Sigma_{\mathcal{A}} = 2^{AP}$, we can construct a *turn-based stochastic Büchi game* $G = \langle S, S_s, S_e, I, A, T, \mathcal{R}, F \rangle$ from $\mathcal{G}$ and $\mathcal{A}$ with the standard product automata construction:

- $S = S_{\mathcal{G}} \times Q_{\mathcal{A}}$ is a finite state space;
- $S_s = S_{s,\mathcal{G}} \times Q_{\mathcal{A}}$ is the set of system states;

- $S_e = S \backslash S_s$ is the set of environment states;
- $I = I_{\mathcal{G}} \times Q_{0,\mathcal{A}}$ is a set of initial states;
- $A = A_{\mathcal{G}}$ is a finite action space;
- $T : S \times A \to \mathcal{D}(S)$ is the transition function such that for any $(s, q) \in S$, $s' \in S_{\mathcal{G}}$ and $a \in A$,

$$T\big((s', q') \mid (s, q), a\big) = \begin{cases} T_{\mathcal{G}}(s' \mid s, a), & \text{if } q' = \delta_{\mathcal{A}}\big(q, L_{\mathcal{G}}(s)\big), \\ 0, & \text{otherwise.} \end{cases}$$

- $\mathcal{R} : S \times A \times S \to \mathbb{R}^{\geq 0}$ is a non-negative reward function such that $\mathcal{R}\big((s, q), a, (s', q')\big) = \mathcal{R}_{\mathcal{G}}(s, a, s')$ for any $(s, q), (s', q') \in S$ and $a \in A$;
- $F = S_{\mathcal{G}} \times F_{\mathcal{A}}$ is a set of accepting states.

The product game G inherits the reward from $\mathcal{G}$ and the winning condition from $\mathcal{A}$. On the reward side, the system is to maximize its future discounted reward, while the environment is to minimize it. On the winning condition side, the system is to win with probability one, regardless of the behavior of the environment. We define the system strategies and then formulate the almost-sure winning objective as well as the discounted-sum objective.

A *(randomized) system strategy* is defined as a tuple $\sigma_s = \langle \sigma_s^m, \rho_s^m, M_s, m_s^0 \rangle$, where M_s is a (possibly countably infinite) set of memory states; $m_s^0 \in M_s$ is the initial memory state; $\sigma_s^m : S_s \times M_s \to \mathcal{D}(A)$, and $\rho_s^m : S \times M_s \to M_s$ is the memory update function. If M_s is a singleton, σ_s is a *memoryless* strategy; if M_s is a finite set, σ_s is a *finite-memory strategy*. With a slight abuse of notation, we use $\sigma_s(s, a)$ to represent $\sigma_s^m(a \mid s, m_s^0)$ for $s \in S_s$, $a \in A$ when σ_s is memoryless. If $|\{s' \in S : \sigma_s(s' \mid s, m) > 0\}| = 1$ for all $s \in S_s$ and $m \in M_s$, σ_s is a *deterministic* strategy. An environment strategy $\sigma_e = \langle \sigma_e^m, \rho_e^m, M_e, m_e^0 \rangle$ can be defined analogously.

The Almost-Sure Winning Objective. The winning condition for G is defined on its runs. Let $A^G : S \to 2^A \backslash \emptyset$ be a mapping from each state to its available actions in G. For each $s \in S$,

$a \in A^G(s)$, let $E^G(s,a) \subseteq S$ be the set of possible successors by taking a at s in G. A *run* $\pi = (s_\pi^{t-1}, a_\pi^t)_{t\in\mathbb{N}^+}$ of G is an infinite sequence of state-action pairs such that for all $t \in \mathbb{N}^+$, $s_\pi^{t-1} \in S$, $a_\pi^t \in A^G(s_\pi^t)$, and $s_\pi^t \in E^G(s_\pi^{t-1}, a_\pi^t)$. π is *winning for the system* with respect to the Büchi condition if and only if F is visited for infinitely many times in π, that is, $|\{t \in \mathbb{N} : s_\pi^t \in F\}| = \infty$.

A system strategy σ_s is *almost-sure winning* at $s \in S$ if a run π with $s_\pi^0 = s$ is winning for the system with probability one when the system takes σ_s, regardless of the environment strategy. The *almost-sure winning region* for the system, denoted by W_{as} (or W_{as}^G to explicitly indicate G), is the set of states at which the system has almost-sure winning strategies. By definition, there are no outgoing transitions from $S_e \bigcap W_{as}$ that leaves W_{as}.

The *almost-sure winning objective* for the system is to always take an almost-sure winning strategy. As a result, the system should always stay within its almost-sure winning region and try to figure out an almost-sure winning strategy through the learning process. It has been shown that deterministic memoryless strategies suffice for the almost-sure winning objective for the system in two-player zero-sum turn-based stochastic Büchi games [35]. In other words, if there exists an almost-sure winning strategy for the system, there exists a deterministic memoryless one.

The Discounted-Sum Objective. We assume that $\mathcal{G}$ is a zero-sum game with an infinite-horizon discounted-sum objective. For the system, the goal is to find a strategy σ_s to maximize the worst-case

expected discounted reward over all possible environment strategies.

$$\max_{\sigma_s} \min_{\sigma_e} \quad \mathbb{E}_{\sigma_s,\sigma_e}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}\big((s_t,q_t),a_t,(s_{t+1},q_{t+1})\big)\right]$$

$$\begin{aligned}
s.t. \qquad & m_s^{t+1} = \rho_s^m\big((s_{t+1},q_{t+1}),m_s^t\big),\ \forall t \in \mathbb{N}^+ \\
& m_e^{t+1} = \rho_e^m\big((s_{t+1},q_{t+1}),m_e^t\big),\ \forall t \in \mathbb{N}^+ \\
& a_t \sim \sigma_s^m\big(\cdot \mid (s_t,q_t),m_s^t\big),\ \text{if } (s_t,q_t) \in S_s \\
& a_t \sim \sigma_e^m\big(\cdot \mid (s_t,q_t),m_e^t\big),\ \text{if } (s_t,q_t) \in S_e \\
& (s_{t+1},q_{t+1}) \sim T(\cdot \mid (s_t,q_t),a_t),\ \forall t \in \mathbb{N}^+ \\
& m_s^0, m_e^0 \text{ given by } \sigma_s,\sigma_e,(s_0,q_0) \in I.
\end{aligned}$$

As common in the RL literature, we use state value functions and action value functions to evaluate system strategies. The *state value function* $V_{\sigma_s} : S \to \mathbb{R}^{\geq 0}$ specifies the worst-case expected discounted reward from each state when the system takes the strategy σ_s. The *action value function* $Q_{\sigma_s} : S \times A \to \mathbb{R}^{\geq 0}$ shows the worst-case expected discounted reward if the system takes a given action at the current step and follows the strategy σ_s thereafter. A system strategy σ_s is *optimal* if it maximizes the state value functions over all system strategies. When σ_s is an optimal strategy, its state value function and action value function are called the *optimal state value function* and the *optimal action value function*, denoted by V^* and Q^* respectively. V^* and Q^* satisfy the following optimality conditions [111]:

$$V^*(s) = \begin{cases} \max_{a \in A^G(s)} Q^*(s,a), & \text{if } s \in S_s \\ \min_{a \in A^G(s)} Q^*(s,a), & \text{if } s \in S_e \end{cases}$$

$$Q^*(s,a) = \sum_{s' \in E^G(s,a)} T(s' \mid s,a)\big(\mathcal{R}(s,a,s') + \gamma V^*(s')\big),$$

where $\gamma \in (0,1)$ is a *discount factor*. For any $\varepsilon > 0$, a system strategy σ_s is *ε-optimal at $s \in S$* if $V_{\sigma_s}(s) \geq V^*(s) - \varepsilon$. Given Σ_s as a set of system strategies, a system strategy $\sigma_s \in \Sigma_s$ is *optimal over Σ_s at $s \in S$* if $V_{\sigma_s}(s) \geq \max_{\sigma'_s \in \Sigma_s} V_{\sigma'_s}(s)$, or *$\varepsilon$-optimal over Σ_s at s* if $V_{\sigma_s}(s) \geq \max_{\sigma'_s \in \Sigma_s} V_{\sigma'_s}(s) - \varepsilon$.

Given $\varepsilon > 0$, the *discounted-sum objective* for the system is to be ε-optimal over all almost-sure winning system strategies at all states that are visited infinitely often. In other words, eventually the worst-case expected reward at any state that will be visited in future is ε-optimal.

6.4. Problem Formulation

We make the following assumptions in our formulation. The first two assumptions are on the observability of the game and the a-priori known knowledge of the game graph. The third assumption is on the unknown reward.

Assumption 3. *The game is fully observable for both the environment and the system. Both agents can observe the joint state, actions taken by either agent and the reward each time a transition is taken.*

Assumption 4. *The system knows the correct list of all possible successors for all state-action pairs, but does not know the exact transition distributions a priori. In other words, the system knows $E^G(s,a)$ for all $s \in S, a \in A$, but does not know $T(\cdot \mid s,a)$.*

Assumption 5. *The reward function is unknown a priori, but is upper bounded by the specified positive number $\mathcal{R}_{\mathsf{max}}$. The upper bound $\mathcal{R}_{\mathsf{max}}$ is not required to be tight.*

Since no system strategy can help guarantee almost-sure winning from outside W^{in}_{as}, we introduce the following assumption on I^{in}.

Assumption 6. $I^{in} \subseteq W^{in}_{as}$.

With the above assumptions, we formulate our learning problem as follows.

Problem 3. *Given a turn-based stochastic Büchi game $G^{in} = \langle S^{in}, S^{in}_s, S^{in}_e, I^{in}, A^{in}, T^{in}, \mathcal{R}^{in}, F^{in} \rangle$ satisfying Assumptions 3 - 6, a discount factor $\gamma \in (0,1)$ and suboptimality bound $\varepsilon > 0$, learn a memoryless system strategy $\sigma_{s,\varepsilon}$ in G^{in} that satisfies both the almost-sure winning objective and the*

discounted-sum objective.

The knowledge of all existing transitions in Assumption 4 is critical in order to achieve the almost-sure winning objective. Without Assumption 4, it is possible that an almost-sure winning strategy can never be learned from any finite observations. For example, even if $s' \in S$ is never witnessed as a successor of a state-action pair (s, a), we cannot confirm that $T(s' \mid s, a) = 0$ with probability 1. With only inaccurate knowledge of the topology of the game graph, the system may leave its almost-sure winning region during the exploration period and thus violate almost-sure winning objective.

6.5. Main Approach

We now propose an algorithm to solve Problem 3. We first solve Problem 3 when the reward functions and transition distributions are known, then discuss how to estimate the game model and learn an ε-optimal almost-sure winning in an online manner.

6.5.1. Restricting G^{in} into the Almost-Sure Winning Region

The first step is to compute the almost-sure winning region W_{as}^{in} for the system. By definition of almost-sure winning regions, the system can only win almost-surely if it always stays within W_{as}^{in}. The definition guarantees that for any $s \in S_s^{in} \bigcap W_{as}^{in}$, there exists an action $a \in A^{G^{in}}(s)$ such that $E^{G^{in}}(s, a) \subseteq W_{as}^{in}$; for any $s \in S_e^{in} \bigcap W_{as}^{in}$, $E^{G^{in}}(s, a) \subseteq W_{as}^{in}$ for all $a \in A^{G^{in}}(s)$. By limiting the set of available actions at each state and the set of initial states, we can construct a new game G from G^{in} to guarantee staying in W_{as}^{in}, regardless of the strategies taken by the system and the environment. Intuitively, $G = G^{in} \restriction W_{as}^{in} = \langle S, S_s, S_e, I, A, T, \mathcal{R}, F \rangle$ is a projection of G^{in} to

$W^{in}_{as} \subseteq S^{in}$ such that

$$S = W^{in}_{as}, S_s = S^{in}_s \bigcap W^{in}_{as}, S_e = S^{in}_e \bigcap W^{in}_{as},$$

$$I = I^{in} \bigcap W^{in}_{as}, F = F^{in} \bigcap W^{in}_{as},$$

$$A^G(s) = \begin{cases} \{a \in A^{G^{in}}(s) : E^{G^{in}}(s,a) \subseteq W^{in}_{as}\} & \text{if } s \in S_s \\ A^{G^{in}}(s) & \text{if } s \in S_e. \end{cases}$$

$$T(s' \mid s, a) = T^{G^{in}}(s' \mid s, a),\ \forall s, s' \in S, a \in A^G(s),$$

$$\mathcal{R}(s, a, s') = \mathcal{R}^{in}(s, a, s'),\ \forall s, s' \in S, a \in A^G(s).$$

Lemma 5 guarantees that we actually lose nothing by considering G instead of G^{in} as we are looking for almost-sure winning strategies.

Lemma 5. *Let σ_s be a system strategy in G^{in}. With Assumption 6, σ_s is almost-sure winning in G^{in} if and only if it is a almost-sure winning strategy in G.*

Proof. By definition, σ_s is almost-sure winning in G^{in} if and only if F^{in} is visited infinitely often with probability 1 by taking σ_s from any initial state in I^{in}. If σ_s is almost-sure winning in G^{in}, all reachable states from I^{in} when the system takes σ_s should be in W^{in}_{as} and thus in S. Accordingly, any state to be visited infinitely often by taking σ_s belongs to $F^{in} \bigcap W^{in}_{as} = F$.

As the set of allowed actions at each system state $s \in S_s$ in G is exactly the set of actions guaranteeing the stay within W^{in}_{as} with probability 1, σ_s is also a system strategy in G. As σ_s guarantees infinite visits to F with probability 1, it is an almost-sure winning system strategy in G.

If σ_s is an almost-sure winning strategy in G, then it is a system strategy in G^{in} and has been shown to be almost-sure winning in G^{in}. □

The set of almost-sure winning system strategies in G is a subset of all system strategies in G.

Therefore if an almost-sure winning system strategy σ_s is ε-optimal in G, it is also ε-optimal over all almost-sure winning system strategies in G and thus ε-optimal over all almost-sure winning strategies in G^{in}. So we can solve Problem 3 by learning a memoryless almost-sure winning system strategy that is ε-optimal in G.

6.5.2. Analysis with Known Reward $\mathcal{R}$ and Transition T

Without the Büchi objective, the discounted-sum objective degenerates to learning a system strategy that is ε-optimal at the infinitely-often-visited states, which can be solved with some slight modification of the R-max algorithm [28]. However, RL algorithms such as R-max are not directly usable to learn ε-optimal almost-sure winning strategies, as the Büchi condition is given independently from the reward function. Moreover, there is no standard parameterization of the space of all almost-sure winning strategies.

In the meantime, the structure of optimal strategies is well-understood for discounted reward. First, deterministic memoryless strategies suffice for optimality with respect to discounted reward. Second, a memoryless system strategy σ_s is optimal if and only if $Q_{\sigma_s}(s,a) = V^*(s)$ holds for all $s \in S_s$ and $a \in \{a' \in A^G(s) : \sigma_s(s,a') > 0\}$ [54]. In other words, σ_s is optimal when it only takes the *optimal actions*.

Suppose that a game G' is constructed from G such that the two games are identical, except that all actions available to the system states in G' are optimal actions in G. Thus any system strategy in G' is optimal over all system strategies in G. Since $A^{G'}(s) \subseteq A^G(s)$ for all $s \in S_s$ and $A^{G'}(s) = A^G(s)$ for all $s \in S_e$, the almost-sure winning region for the system in G' will be a subset of that in G. That is, $W^{G'}_{as} \subseteq W^G_{as}$. There can be two cases:

1. If $W^{G'}_{as} = W^G_{as}$, there exists a memoryless almost-sure winning strategy σ_s in G'. As there are no restrictions to the environment in G', σ_s is also almost-sure winning in G. As σ_s is also optimal in G, it is a solution to Problem 3.

2. If $W^{G'}_{as} \subset W^G_{as}$, there does not exist an optimal system strategy that is almost-sure winning in

G. It is necessary for the system to take some suboptimal actions in G in order to preserve the almost-sure winning region W_{as}^{G}.

We only need to deal with the second case. Lemma 6 shows one way to construct a set of almost-sure winning strategies in G^{in}. It is guaranteed that all system strategies that assign positive probability to all available actions in G are almost-sure winning in G^{in}.

Lemma 6. *Given a turn-based stochastic Büchi game $G^{in} = \langle S^{in}, S_s^{in}, S_e^{in}, I^{in}, A^{in}, T^{in}, \mathcal{R}^{in}, F^{in} \rangle$ and its almost-sure winning region W_{as}^{in}, define $G = G^{in} \restriction W_{as}^{in} = \langle S, S_s, S_e, I, A, T, \mathcal{R}, F \rangle$ as in Section 6.5.1. Then any memoryless system strategy σ_s in G^{in} such that $\{a \in A^{G^{in}}(s) \mid \sigma_s(s,a) > 0\} = A^{G}(s)$ holds for all $s \in S_s$ is almost-sure winning at any state $s \in W_{as}^{in}$.*

Proof. The proof is done by contradiction. Let the system agent take the strategy σ_s such that $\{a \in A^{G^{in}}(s) \mid \sigma_s(s,a) > 0\} = A^{G}(s)$ holds for all $s \in S_s$. By definition of G and σ_s, the environment cannot force the system to leave W_{as}^{in} while the system takes σ_s. The only possible scenario that σ_s is not almost-sure winning at some $s_{\sigma_s} \in W_{as}^{in}$ is that there exists an environment strategy σ_e such that starting from s_{σ_s}, the probability to prevent the system from visiting F infinitely often is positive. As $|W_{as}^{in}| < \infty$, it implies the existence of a state $s'_{\sigma_s} \in W_{as}^{in}$ from which it is impossible to visit F when the system takes σ_s and the environment takes σ_e.

However, the aforementioned scenario is impossible. By definition of the almost-sure winning region W_{as}^{in}, there exists a deterministic memoryless system strategy σ'_s that is almost-sure winning at all $s \in W_{as}^{in}$ regardless of the environment's strategy [33]. Therefore, there exists some finite positive integer $N(s'_{\sigma_s})$ such that starting from s'_{σ_s}, the probability to reach F within $N(s'_{\sigma_s})$ steps is positive when the system takes σ'_s, regardless of the environment's strategy. By definition of σ_s, there is some positive probability that the system takes the same actions as σ'_s for $N(s'_{\sigma_s})$ steps while taking the strategy σ_s. Therefore, there will be some positive probability to reach F from s'_{σ_s} regardless of the environment strategy, which contradicts our assumption that it is impossible to visit F from s'_{σ_s} when the system takes σ_s and the environment takes σ_e. As a result, σ_s is almost-sure winning at all states in W_{as}^{in}. □

We now show that an ε-optimal strategy can be constructed by bounding the probability of taking arbitrary suboptimal actions. Given $\varepsilon > 0$, we first define the set of ε-optimal actions, and then build a connection between the suboptimality of a given system strategy σ_s and its probability to take ε-optimal actions in Lemma 7.

Definition 5. *(ε-optimal actions) Let $G = \langle S, S_s, S_e, I, A, T, \mathcal{R}, F\rangle$ be a turn-based stochastic Büchi game and $\varepsilon > 0$ be a constant. Define the set of ε-optimal actions at each system state $s \in S_s$ as*

$$A^*_\varepsilon(s) = \{a \in A^G(s) \mid Q^*(s,a) \geq \max_{a' \in A^G(s)} Q^*(s,a') - \varepsilon\},$$

where Q^* is the optimal action value function of G.

Lemma 7. *Let $G = \langle S, S_s, S_e, I, A, T, \mathcal{R}, F\rangle$ be a turn-based stochastic Büchi game such that all rewards are upper bounded by $\mathcal{R}_{max} > 0$. Let V^* be the optimal value function of G, $\gamma \in (0,1)$ be the discount factor and $\varepsilon > 0, p_\varepsilon \in (0,1)$ be fixed scalars. Then if a system memoryless strategy σ_s satisfies $\sum_{a \notin A^*_\varepsilon(s)} \sigma_s(s,a) \leq p_\varepsilon$ for all $s \in S_s$, then*

$$||V^* - V_{\sigma_s}||_\infty = \max_{s' \in S} \left(V^*_{\sigma_s}(s') - V(s')\right) \leq p_\varepsilon \frac{\mathcal{R}_{max}}{(1-\gamma)^2} + \frac{\varepsilon}{1-\gamma}.$$

Proof. If $s \in S_s$ and $a \in A^*_\varepsilon(s)$, it holds that

$$Q^*(s,a) = \sum_{s' \in S} T(s' \mid s,a)(\mathcal{R}(s,a) + \gamma V^*(s')) \geq V^*(s) - \varepsilon.$$

We compare the optimal value function and the value function corresponding to σ_s at $s \in S_s$,

$$
\begin{aligned}
& V^*(s) - V_{\sigma_s}(s) \\
= & \sum_{a \notin A^*_\varepsilon(s)} \sigma_s(s,a) V^*(s) + \sum_{a \in A^*_\varepsilon(s)} \sigma_s(s,a) V^*(s) - \\
& \Big(\sum_{a \notin A^*_\varepsilon(s)} \sigma_s(s,a) Q_{\sigma_s}(s,a) + \sum_{a \in A^*_\varepsilon(s)} \sigma_s(s,a) Q_{\sigma_s}(s,a) \Big) \\
= & \sum_{a \notin A^*_\varepsilon(s)} \sigma_s(s,a) \big(V^*(s) - Q_{\sigma_s}(s,a) \big) \; + \sum_{a \in A^*_\varepsilon(s)} \sigma_s(s,a) \Big(Q^*(s, a^*(s)) - Q_{\sigma_s}(s,a) \Big) \\
& \big(\text{where } a^*(s) \in \arg\max_{a' \in A(s)} Q^*(s,a') \big) \\
\overset{(*)}{\leq} & \sum_{a \notin A^*_\varepsilon(s)} \sigma_s(s,a) V^*(s) + \sum_{a \in A^*_\varepsilon(s)} \sigma_s(s,a) \cdot \Big(Q^*\big(s, a^*(s)\big) - Q^*(s,a) + Q^*(s,a) - Q_{\sigma_s}(s,a) \Big) \\
\overset{(**)}{\leq} & \sum_{a \notin A^*_\varepsilon(s)} \sigma_s(s,a) V^*(s) + \sum_{a \in A^*_\varepsilon(s)} \sigma_s(s,a) \cdot \Big(\varepsilon + \gamma \sum_{s' \in S} T(s' \mid s,a) \big(V^*(s') - V_{\sigma_s}(s') \big) \Big) \\
\overset{(***)}{\leq} & \; p_\varepsilon \frac{\mathcal{R}_{\mathsf{max}}}{1-\gamma} + \varepsilon + \gamma ||V^* - V_{\sigma_s}||_\infty .
\end{aligned}
$$

Inequality (*) holds as $\mathcal{R}$ is non-negative and $V_{\sigma_s}(s) \geq 0$ for all $s \in S$. Inequality (**) holds by definition of ε-optimal actions: $Q^*\big(s, a^*(s)\big) - \varepsilon \leq Q^*(s,a)$ for all $a \in A^*_\varepsilon(s)$. Inequality (***) holds as $\max_{s \in S} V^*(s) \leq \frac{1}{1-\gamma} \mathcal{R}_{\mathsf{max}}$ and $\max_{s \in S} \sum_{a \in A^*_\varepsilon(s)} \sigma_s(s,a) \leq p_\varepsilon$.

Let σ_e be an optimal environment strategy when the system takes σ_s, and σ^*_e be an optimal environment strategy when the system takes an optimal strategy σ^*_s. Then V^* and V can be reached respectively when the strategy pair of the system and the environment is (σ^*_s, σ^*_e) and (σ_s, σ_e), which we denote by $V^* = V_{(\sigma^*_s, \sigma^*_e)}$ and $V_\sigma = V_{(\sigma_s, \sigma_e)}$. As σ^*_e is optimal for the environment when the system takes σ^*_s and the environment is trying to minimize the value function at each $s \in S_e$, it holds

that

$$\sum_{a \in A^G(s)} \sigma_e(s,a) \sum_{s' \in S} T(s' \mid s,a)\big(\mathcal{R}(s,a,s') + \gamma V^*(s')\big)$$

$$\geq \sum_{a \in A^G(s)} \sigma_e^*(s,a) \sum_{s' \in S} T(s' \mid s,a)\big(\mathcal{R}(s,a,s') + \gamma V^*(s')\big).$$

Then for any environment state $s \in S_e$, we have

$$\begin{aligned}
&V^*(s) - V_{\sigma_s}(s) \\
&= \sum_{a \in A^G(s)} \sigma_e^*(s,a) \sum_{s' \in S} T(s' \mid s,a)(\mathcal{R}(s,a,s') + \gamma V^*(s')) \\
&- \sum_{a \in A^G(s)} \sigma_e(s,a) \sum_{s' \in S} T(s' \mid s,a)(\mathcal{R}(s,a,s') + \gamma V_{\sigma_s}(s')) \\
&\leq \sum_{a \in A^G(s)} \sigma_e(s,a) \sum_{s' \in S} T(s' \mid s,a)(\mathcal{R}(s,a,s') + \gamma V^*(s')) \\
&- \sum_{a \in A^G(s)} \sigma_e(s,a) \sum_{s' \in S} T(s' \mid s,a)(\mathcal{R}(s,a,s') + \gamma V_{\sigma_s}(s')) \\
&= \sum_{a \in A^G(s)} \sigma_e(s,a) \sum_{s' \in S} T(s' \mid s,a)\gamma(V^*(s') - V_{\sigma_s}(s')) \\
&\leq \gamma ||V^* - V_{\sigma_s}||_\infty.
\end{aligned}$$

Note that since V is the value function of the system, $V^*(s) \geq V_{\sigma_s}(s)$ holds even if $s \in S_e$.

Therefore if there exists $s \in S_e$ such that $V^*(s) - V_{\sigma_s}(s) = ||V^* - V_{\sigma_s}||_\infty$, then as $V^* \geq V_{\sigma_s}$ and $\gamma \in (0,1)$, $||V^* - V_{\sigma_s}||_\infty = 0$. Otherwise $||V^* - V_{\sigma_s}||_\infty$ can only be reached by system states and thus

$$||V^* - V_{\sigma_s}||_\infty \leq p_\varepsilon \frac{\mathcal{R}_{\mathsf{max}}}{1-\gamma} + \varepsilon + \gamma ||V^* - V_{\sigma_s}||_\infty,$$

thus

$$||V^* - V_{\sigma_s}||_\infty \leq p_\varepsilon \frac{\mathcal{R}_{\mathsf{max}}}{(1-\gamma)^2} + \frac{\varepsilon}{1-\gamma}.$$

Algorithm 4 HatGame and RecoverHatStrategy

function HATGAME(G, Q^*, ε_1, p_{ε_1})

For $i \in \{1,2\}$, $S_s^i \leftarrow \{s^i \mid s \in S_s\}$.

Define $B : S_s^1 \bigcup S_s^2 \to S_s$ such that for all $s^i \in S_s^i$ and $i \in \{1,2\}$, $B(s^i) = s$.

$\hat{S}_s \leftarrow S_s \bigcup S_s^1 \bigcup S_s^2$, $\hat{S} \leftarrow \hat{S}_s \bigcup S_e$.

Define $A_{\varepsilon_1}^* : S_s \to 2^A \backslash \emptyset$ such that for all $s \in S_s$, $A_{\varepsilon_1}^*(s) = \{a \in A^G(s) \mid \max_{a' \in A^G(s)} Q^*(s,a') - Q^*(s,a) \le \varepsilon_1\}$.

$\hat{A} \leftarrow A \bigcup \{\hat{a}\}$.

Define the set of available actions for all $s \in \hat{S}$: $A^{\hat{G}}(s) = \begin{cases} A^G(s) & \text{if } s \in S_e, \\ \{\hat{a}\} & \text{if } s \in S_s, \\ A^G(B(s)) & \text{if } s \in S_s^1, \\ A_{\varepsilon_1}^*(B(s)) & \text{if } s \in S_s^2. \end{cases}$

Define the transition function for all $s, s' \in \hat{S}$ and $a \in A^{\hat{G}}(s)$: $\hat{T}(s' \mid s, a) = \begin{cases} p_{\varepsilon_1} & \text{if } s \in S_s, s' = s^1, \\ 1 - p_{\varepsilon_1} & \text{if } s \in S_s, s' = s^2, \\ T(s' \mid s, a) & \text{if } s \in S_e, s' \in S, \\ T(s' \mid B(s), a) & \text{if } s \in S_s^1 \bigcup S_s^2, s' \in S. \end{cases}$

return $\hat{G} = (\hat{S}, \hat{S}_s, \hat{S}_e, I, \hat{T}, \mathcal{R}, F)$.

end function

function RECOVERHATSTRATEGY($\hat{G}$, $\hat{\sigma}_s$)

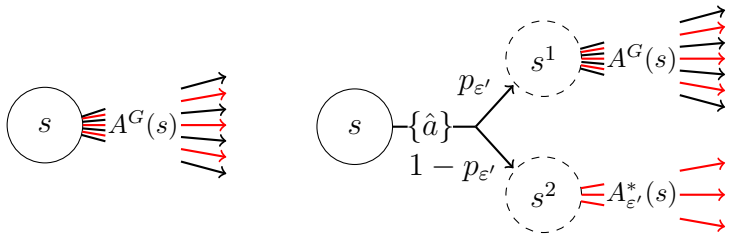
For all $s \in S_s$ and $a \in A^G(s)$, $\bar{\sigma}_s(s,a) \leftarrow \hat{T}(s^1 \mid s, \hat{a})\hat{\sigma}_s(s^1, a) + \hat{T}(s^2 \mid s, \hat{a})\hat{\sigma}_s(s^2, a)$, where $\hat{T}$ is the transition function of $\hat{G}$.

return $\bar{\sigma}_s$.

end function

And it confirms that σ_s is $\left(p_\varepsilon \frac{\mathcal{R}_{\max}}{(1-\gamma)^2} + \frac{\varepsilon}{1-\gamma}\right)$-optimal. □

We use Lemma 7 to construct a game $\hat{G} = \textbf{HatGame}(G, Q^*, \varepsilon, p_\varepsilon)$ as in Algorithm 4, such that any system strategy in $\hat{G}$ corresponds to an ε-optimal system strategy in G. The idea is to select $\varepsilon' > 0$ from ε and partition the set of available actions $A^G(s)$ at each system state $s \in S_s$ into two parts: the set $A_{\varepsilon'}^*(s)$ of ε'-optimal actions and the other actions. For each system state $s \in S_s$, we add two additional system states s^1 and s^2 in $\hat{G}$ such that $\hat{a}$ is the unique available action at s and $E^{\hat{G}}(s, \hat{a}) = \{s^1, s^2\}$. At s^1 the system can take all actions in $A^G(s)$; at s^2 the system can only take the ε'-optimal actions. The transition probability from s to s^1 is confined to be a small number $p_{\varepsilon'}$.



(a) Transitions from $s \in S_s$ in G.

(b) Transitions from $s \in S_s$ in $\hat{G}$.

Figure 16: Illustration of the construction of $\hat{G} = \textbf{HatGame}(G, Q^*, \varepsilon', p_{\varepsilon'})$. Each arrow represents an available action from the starting state. The red arrows correspond to ε'-optimal actions in $A^*_{\varepsilon'}(s)$. For each system state $s \in S_s$ in G, there are three states in $\hat{G}$: a copy of the state s with only one available action $\hat{a}$, and two virtual states s^1 and s^2 such that $A^{\hat{G}}(s^1) = A^G(s)$ and $A^{\hat{G}}(s^2) = A^*_{\varepsilon'}(s)$.

The comparison between transitions from $s \in S_s$ in G and in $\hat{G}$ is illustrated in Figure 16.

For each memoryless system strategy $\hat{\sigma}_s$ in $\hat{G}$, we can construct a memoryless system strategy $\sigma_s = \textbf{RecoverHatStrategy}(\hat{G}, \hat{\sigma}_s)$ in G. The following lemma bounds the suboptimality of any system strategy that is derived from $\hat{G}$.

Lemma 8. *Let G and $\hat{G}$ be two turn-based Büchi games such that $\hat{G} = \textbf{HatGame}(G, Q^*, \varepsilon_1, p_{\varepsilon_1})$, where Q^* is the optimal action value function of G, and $\varepsilon_1 > 0, p_{\varepsilon_1} \in (0, 1)$ are constants. The reward function $\mathcal{R}$ in G is bounded by $\frac{\mathcal{R}_{max}}{1-\gamma}$, where $\gamma \in (0, 1)$ is the discount factor. If $\hat{\sigma}_s$ is a strategy for the system in $\hat{G}$, then $\sigma_s = \textbf{RecoverHatStrategy}(\hat{G}, \hat{\sigma}_s)$ is $\left(p_{\varepsilon_1} \frac{\mathcal{R}_{max}}{(1-\gamma)^3} + \frac{\varepsilon_1}{1-\gamma}\right)$-optimal in G.*

Proof. The proof can be done by checking that $\sum_{a \in A^*_{\varepsilon_1}(s)} \sigma_s(s, a) \leq p_{\varepsilon_1}$ holds for each system state $s \in S_s$ and applying Lemma 7. Note that the reward upper bound is $\frac{\mathcal{R}_{\max}}{1-\gamma}$. □

Lemma 9 guarantees that the system strategies $\hat{\sigma}_s$ in $\hat{G}$ and $\sigma_s = \textbf{RecoverHatStrategy}(\hat{G}, \hat{\sigma}_s)$ in G can only be almost-sure winning simultaneously.

Lemma 9. *Let G and $\hat{G}$ be two turn-based Büchi games such that $\hat{G} = \textbf{HatGame}(G, Q^*, \varepsilon_1, p_{\varepsilon_1})$, where $\varepsilon_1 > 0, p_{\varepsilon_1} > 0$ and Q^* is the optimal action function of G. Let $\hat{\sigma}_s$ be a system strategy in $\hat{G}$ and $\sigma_s = \textbf{RecoverHatStrategy}(\hat{G}, \hat{\sigma}_s)$ as in Lemma 8. Then $\hat{\sigma}_s$ is almost-sure winning for the system in $\hat{G}$ if and only if the constructed σ_s is almost-sure winning for the system in G.*

Proof. By construction of $\hat{G} = \langle \hat{S}, \hat{S}_s, \hat{S}_e, I, A, \hat{T}, \mathcal{R}, F \rangle$, the system states of $\hat{G}$ can be partitioned into three non-overlapping parts: $\hat{S} = S_s \bigcup S_s^1 \bigcup S_s^2$. Let $P_{\hat{\sigma}_s}^{\hat{G}}(\cdot, \cdot) : S_s \times S \to [0, 1]$ be a function such that for any $s, s' \in S = S_s \bigcup S_e$, $P_{\hat{\sigma}_s}^{\hat{G}}(s, s')$ is the probability that the next visited state in S from s is s' when the system takes $\hat{\sigma}_s$ in $\hat{G}$. Similarly, define $P_{\sigma_s}^{G}(\cdot, \cdot) : S_s \times S \to [0, 1]$ to represent the transition probability by taking σ_s in G. Then by definition of $\hat{T}, \hat{\sigma}_s, \sigma_s$ and $A^{\hat{G}}(\cdot)$ in Algorithm 4, for any $s \in S_s, s' \in S$:

$$
\begin{aligned}
& P_{\hat{\sigma}_s}^{\hat{G}}(s, s') \\
=& \hat{T}(s^1 \mid s, \hat{a}) \sum_{a \in A^G(B(s))} \hat{\sigma}_s(s^1, a) \hat{T}(s' \mid s^1, a) + \hat{T}(s^2 \mid s, \hat{a}) \sum_{a \in A_{\varepsilon_1}^*(B(s))} \hat{\sigma}_s(s^2, a) \hat{T}(s' \mid s^2, a) \\
=& \sum_{a \in A^{\hat{G}}(B(s^1))} \hat{T}(s^1 \mid s, \hat{a}) \hat{\sigma}_s(s^1, a) T(s' \mid B(s^1), a) \\
& + \sum_{a \in A^{\hat{G}}(B(s^2))} \hat{T}(s^2 \mid s, \hat{a}) \hat{\sigma}_s(s^2, a) T(s' \mid B(s^2), a) \\
=& \sum_{a \in A^G(s)} \sigma_s(s, a) T(s' \mid s, a) = P_{\sigma_s}^{G}(s, s').
\end{aligned}
$$

In other words, the probability that the system transits from $s \in S_s$ to $s' \in S$ is the same if the system takes σ_s in G or if it takes $\hat{\sigma}_s$ in $\hat{G}$.

Let $M_G = \langle S, \emptyset, S_e, I, A, T_G, \mathcal{R}, F \rangle$ and $M_{\hat{G}} = \langle S, \emptyset, S_e, I, A, T_{\hat{G}}, \mathcal{R}, F \rangle$ be the two generated MDPs for the environment when the system takes σ_s and $\hat{\sigma}_s$ in G and $\hat{G}$, respectively. For any $s \in S_e$ and $a \in A^G(s)$, $s' \in S$:

$$
T_G(s' \mid s, a) = T_{\hat{G}}(s' \mid s, a) = T(s' \mid s, a).
$$

For any $s \in S_s$ and $a \in A$, $s \in S$:

$$
T_{\hat{G}}(s' \mid s, a) = P_{\hat{\sigma}_s}^{\hat{G}}(s, s') = P_{\sigma_s}^{G}(s, s') = T_G(s' \mid s, a).
$$

Therefore the transition distributions between states in S are the same in these two MDPs. The winning condition for the environment in both games is to prevent the system from visiting F infinitely often with positive probability. An environment strategy σ_e prevents the system from visiting $F \subset S$ for infinite times in M_G with positive probability if and only if it prevents the system from visiting F for infinite times in $M_{\hat{G}}$ with positive probability. In other words, $\hat{\sigma}_s$ is almost-sure winning for the system in $\hat{G}$ if and only if the constructed σ_s is almost-sure winning for the system in G. □

In summary, given the optimal action value function Q^*, we can compute a memoryless ε-optimal almost-sure winning strategy in G as follows. First, construct a game $\hat{G} = \textbf{HatGame}(G, Q^*, \varepsilon', p_{\varepsilon'})$ for some ε' and $p_{\varepsilon'}$ using Algorithm 4, such that any system strategy $\hat{\sigma}_s$ in $\hat{G}$ corresponds to an ε-optimal system strategy $\sigma_s = \textbf{RecoverHatStrategy}(\hat{G}, \hat{\sigma}_s)$ in G. Then synthesize a memoryless almost-sure winning strategy $\hat{\sigma}_s$ in $\hat{G}$. By Lemma 9, the corresponding system strategy σ_s is also almost-sure winning in G. The existence of almost-sure winning system strategies in $\hat{G}$ is guaranteed by Lemma 6.

Remark 4. *One way to construct a memoryless almost-sure winning system strategy $\hat{\sigma}_s$ is as follows:*

$$\hat{\sigma}_s(s, a) = \frac{1}{|A^{\hat{G}}(s)|}, \textit{ for all } s \in \hat{S}_s.$$

The idea to prove that such a system strategy $\hat{\sigma}_s$ is almost-sure winning is the same as that of Lemma 6.

The construction of $\hat{G}$ from G requires the knowledge of the ground-truth optimal action value function Q^*. If the reward function $\mathcal{R}$ and the transition function T are approximated, the estimated Q function and the estimated sets of ε-optimal actions may not be accurate. Therefore the system strategy σ_s derived from $\hat{\sigma}_s$ may not be ε-optimal for the ground-truth game G. σ_s is always almost-sure winning in G if $\hat{\sigma}_s$ is almost-sure winning in $\hat{G}$, no matter what Q^* is.

6.5.3. Algorithm with Unknown Reward $\mathcal{R}$ and Transition T

Now we return to Problem 3, where both the reward function and the transition distributions are unknown. Motivated by the R-max algorithm [28], we learn an optimistically-initialized game model $\bar{G}$ of the ground-truth game G in an online manner. The key idea is as follows: We initially mark all state-action pairs in G as unknown. Given a suboptimality bound $\varepsilon > 0$ and confidence level $1 - \delta_c \in (0, 1)$, we compute a constant K_{ε,δ_c} such that a state-action pair is considered as *known* or *learned* if it has been taken for K_{ε,δ_c} times. The system strategy $\sigma_{s,\varepsilon}$ is always almost-sure winning and ε-optimal with respect to its current game model $\bar{G}$. Both $\bar{G}$ and $\sigma_{s,\varepsilon}$ are updated every time a new state-action pair is learned. The overall algorithm is shown in Algorithm 5, with guarantees stated in Theorem 6.

Theorem 6. *Let G^{in} be a turn-based Büchi game, $\mathcal{R}_{\mathsf{max}}$ be a positive upper bound of the reward, $\gamma \in (0, 1)$ be a discount factor, $\varepsilon > 0$ be a suboptimality bound and $1 - \delta_c \in (0, 1)$ be a confidence lower bound, as given in the input of Algorithm 5. The system strategy $\sigma_{s,\varepsilon}$ in Algorithm 5 is always memoryless and almost-sure winning. With probability no less than $1 - \delta_c$, by taking $\sigma_{s,\varepsilon}$, the future expected discounted reward from the current state s is at least $V^*(s) - \varepsilon$, except for some number of steps polynomial in $|S|, |A|, \frac{1}{\varepsilon}$ and $\frac{1}{\delta_c}$.*

Algorithm 5 is sketched as follows. We first compute the almost-sure winning region W_{as}^{in} for the system in the game G^{in} (Step 1). Then we construct a new game G such that regardless of the system's strategy in G, the system will always stay within W_{as}^{in} (Step 2). G cannot be used directly in Algorithm 5, as the transition T and reward function $\mathcal{R}$ of G are unknown. Instead, we keep an estimated model $\bar{G}$ of G, which is initialized to have uniform transition distribution and constant reward for all state-action pairs (Step 3). The initial reward value is set to be an upper bound of the value function of G with the following two purposes. First, the optimistic initial value function encourages the system to take unknown transitions, as the Q values of the unknown state-action pairs are higher than those of the known ones. Second, once an unknown transition is taken, the discounted reward in $\bar{G}$ will be higher than that in G. $\sigma_{s,\varepsilon}$ is initialized to be an almost-sure winning system strategy for G and $\bar{G}$ (Step 5). In each iteration, the system takes $\sigma_{s,\varepsilon}$ for one step and observe

the transition (s_t, a_t, s_{t+1}, r_t) (Step 10). We count the frequency of taking each transition (s, a, s') (Step 11): If the number of visits to (s_t, a_t) exceeds some given threshold K (Step 13), the game model $\bar{G}$ is updated (Step 16) and the system strategy $\sigma_{s,\varepsilon}$ is recomputed (Step 20).

The almost-sure winning objective and the discounted-sum objective are no longer intertwined in Algorithm 5. The almost-sure winning objective is achieved as follows. By constructing the game G, the actions taken by the system are restricted such that the system always stays in the almost-sure winning region (Step 1). By Lemma 6, the system strategies $\sigma_{s,\varepsilon}$ (Step 5) and $\hat{\sigma}_s$ (Step 19) are almost-sure winning in G and $\hat{G}$, respectively. By Lemma 9, the updated system strategy $\sigma_{s,\varepsilon}$ in Step 20 is also almost-sure winning in $\bar{G}$. It remains to be shown that $\sigma_{s,\varepsilon}$ is also almost-sure winning in G.

The discounted-sum objective is addressed by the construction of $\hat{G}$. By Lemma 8, $\hat{\sigma}_s$ is a system strategy in $\hat{G}$ and thus $\sigma_{s,\varepsilon}$ is $\frac{\varepsilon}{6}$-optimal in $\bar{G}$. The suboptimality of $\sigma_{s,\varepsilon}$ in G will be analyzed later. As the construction of $\hat{G}$ is independent on the almost-sure winning objective and any system strategy of $\hat{G}$ can be used to construct an ε-optimal strategy of G, the two objectives are now fully separated in our solution.

6.6. Proof of Theorem 6

We show a proof of Theorem 6 in this section. With the previous analysis, there are two key points in this proof: First, any memoryless system strategy that is almost-sure winning in the game model $\bar{G}$ is also almost-sure winning in the ground-truth game G; second, with probability at least $(1 - \delta_c)$, $\sigma_{s,\varepsilon}$ is ε-optimal in the ground-truth game G except for some number of steps that is polynomial in $|S|, |A|, \frac{1}{\varepsilon}, \frac{1}{\delta_c}$. These two points are addressed consecutively in this section.

6.6.1. Proof of Almost-Sure Winning Objective

We first show that any almost-sure winning system strategy in $\bar{G}$ is also almost-sure winning in G^{in}.

Lemma 10. *Let $G = \langle S, S_s, S_e, I, A, T, \mathcal{R}, F\rangle$ and $\bar{G} = \langle S, S_s, S_e, I, A, \bar{T}, \bar{\mathcal{R}}, F\rangle$ be two turn-based stochastic Büchi games sharing the same transitions. In other words, for all $s, s' \in S, a \in A$,*

Algorithm 5 Overall algorithm for Problem 3

Require: A turn-based Büchi game G^{in}, $\mathcal{R}_{\max} > 0$, $\gamma \in (0,1)$, a suboptimality bound $\varepsilon > 0$, a confidence bound $\delta_c \in (0,1)$, maximum number of iterations K_{max}.

Compute W^{in}_{as}.
$G \leftarrow G^{in} \restriction W^{in}_{as} = \langle S, S_s, S_e, I, A, T, \mathcal{R}, F \rangle$.
$\bar{G} \leftarrow \langle S, S_s, S_e, I, A, \bar{T}, \bar{\mathcal{R}}, F \rangle$ such that $\bar{T}(s' \mid s, a) = \frac{1}{|E^G(s,a)|}$ and $\bar{\mathcal{R}}(s,a,s') \leftarrow \frac{\mathcal{R}_{\max}}{1-\gamma}$ for all $s \in S$, $a \in A^G(s)$ and $s' \in E^G(s,a)$.
for $s \in S_s$, $a \in A^G(s)$, $s' \in E^G(s,a)$ **do**
    $\sigma_{s,\varepsilon}(s,a) \leftarrow \frac{1}{|A^G(s)|}$.
    $k(s,a,s') \leftarrow 0$, $L(s,a) \leftarrow 0$, $\bar{Q}^*(s,a) \leftarrow \frac{\mathcal{R}_{\max}}{(1-\gamma)^2}$.
end for
$\delta \leftarrow \frac{\varepsilon(1-\gamma)^2 \log(\gamma)}{6\mathcal{R}_{\max}|S| \log(\varepsilon(1-\gamma)^2/6\mathcal{R}_{\max})}$, $K \leftarrow \frac{1}{2\delta^2} \log \frac{4|A||S|^2}{\delta_c}$, $\varepsilon_1 \leftarrow \frac{1-\gamma}{12}\varepsilon$, $p_{\varepsilon_1} \leftarrow \frac{\varepsilon(1-\gamma)^3}{12\mathcal{R}_{\max}}$.
for $t = 0, 1, \ldots, K_{max}$ **do**
    $a_t, s_{t+1}, r_t = \textbf{Simulate}(s_t, \sigma_{s,\varepsilon})$.
    $k(s_t, a_t, s_{t+1}) \leftarrow k(s_t, a_t, s_{t+1}) + 1$.
    if $L(s_t, a_t) = 0$ **then**
        if $\sum_{s' \in S} k(s_t, a_t, s') \geq K$ or $|E^G(s_t, a_t)| = 1$ **then**
            $L(s_t, a_t) \leftarrow 1$.
            $k(s_t, a_t, s') \leftarrow \max(k(s_t, a_t, s'), 1)$ for all $s' \in E^G(s_t, a_t)$.
            $\bar{T}(s' \mid s_t, a_t) \leftarrow \frac{k(s_t,a_t,s')}{\sum_{s' \in S} k(s_t,a_t,s')}$ for all $s' \in E^G(s_t, a_t)$, $\bar{\mathcal{R}}(s_t, a_t, s_{t+1}) \leftarrow r_t$.
            Update $\bar{Q}^*$.
            Construct $\hat{G} \leftarrow \textbf{HatGame}(\bar{G}, \bar{Q}^*, \varepsilon_1, p_{\varepsilon_1})$.
            Compute a memoryless almost-sure winning strategy $\hat{\sigma}_s$ for the system in $\hat{G}$.
            $\sigma_{s,\varepsilon} \leftarrow \textbf{RecoverHatStrategy}(\hat{G}, \hat{\sigma}_s)$.
        end if
    end if
end for
function SIMULATE(s, σ_s)
    if $s \in S_s$ **then**
        Draw action $a \sim \sigma_s(s, \cdot)$.
    else
        The environment takes an action $a \in A^G(s)$.
    end if
    Observe the next state s' and the reward r.
    return a, s', r.
end function

$T(s' \mid s, a) > 0$ *if and only if* $\bar{T}(s' \mid s, a) > 0$*. Then for any* $s_0 \in S$*, a memoryless system strategy* σ_s *is almost-sure winning in* G *at* s_0 *if and only if it is almost-sure winning in* $\bar{G}$ *at* s_0*.*

Proof. The proof is done by contradiction. Turn-based stochastic Büchi games is a special case of stochastic turn-based parity games. It has been shown that in (finite-state) stochastic turn-based parity games, deterministic memoryless strategies suffice for the environment to minimize the winning probability for the system [203]. Without loss of generality, assume that the strategies σ_e, σ_s taken by the environment and the system are both deterministic and memoryless.

Then for any finite sequence of state-action pairs $\pi = (s_{i-1}, a_i)_{i=1,\cdots,K}$, the probability that π is part of a run of G when the system takes σ_s is

$$P_G(\pi) = \prod_{i=1}^{K-1} T(s_i \mid s_{i-1}, a_i) \prod_{i=1}^{K} \big(\sigma_s(s_{i-1}, a_i)\mathbf{1}_{S_s}(s_{i-1}) + \sigma_e(s_{i-1}, a_i)\mathbf{1}_{S_e}(s_{i-1})\big)$$

and the corresponding probability in $\bar{G}$ is

$$P_{\bar{G}}(\pi) = \prod_{i=1}^{K-1} \bar{T}(s_i \mid s_{i-1}, a_i) \prod_{i=1}^{K} \big(\sigma_s(s_{i-1}, a_i)\mathbf{1}_{S_s}(s_{i-1}) + \sigma_e(s_{i-1}, a_i)\mathbf{1}_{S_e}(s_{i-1})\big),$$

where for any set S, $\mathbf{1}_S(\cdot)$ denotes the characteristic function of S. By assumption, it holds for all $i = 1, \cdots, K-1$ that $T(s_i \mid s_{i-1}, a_i) > 0$ if and only if $\bar{T}(s_i \mid s_{i-1}, a_i) > 0$. Then for any $\pi = (s_{i-1}, a_i)_{i=1,\cdots,K}$ of finite length, $\prod_{i=1}^{K-1} T(s_i \mid s_{i-1}, a_i) > 0$ if and only if $\prod_{i=1}^{K-1} \bar{T}(s_i \mid s_{i-1}, a_i) > 0$, therefore $P_G(\pi) > 0$ if and only if $P_{\bar{G}}(\pi) > 0$. Given any state $s \in S$ and any pair of memoryless strategies σ_s and σ_e for the system and the environment, the sets of states that can be reached from s with positive probability (referred to as 'reachable' below) are the same in G and $\bar{G}$.

Suppose σ_s is not almost-sure winning at $s_0 \in S$ in G. Then as shown in the proof of Lemma 6, there exist a state $s_{\sigma_e} \in S$ and an environment strategy σ_e such that the following two conditions hold in G when the environment takes σ_e:

1. s_{σ_e} can be reached with positive probability from s_0;

2. the probability of reaching F from s_{σ_e} is zero.

In other words, s_{σ_e} is reachable from s_0 and F is not reachable from s_{σ_e} in G, when the environment takes σ_e and the system takes σ_s. With the previous analysis, s_{σ_e} is reachable from s_0 and F is not reachable from s_{σ_e} in $\bar{G}$. Therefore, there is some positive probability that F cannot be visited for infinitely many times from s_0, which shows that σ_s cannot be almost-sure winning in $\bar{G}$ at s_0.

We can similarly show that if σ_s is not almost-sure winning in $\bar{G}$ at s_0, it cannot be almost-sure winning in G at s_0. Therefore σ_s is almost-sure winning in G at s_0 if and only if it is almost-sure winning in $\bar{G}$ at s_0. □

Now we show that $\sigma_{s,\varepsilon}$ in Algorithm 5 is almost-sure winning for the system in G^{in}.

In Step 5 of Algorithm 5, $\sigma_{s,\varepsilon}$ is initialized to assign positive probability to all actions at all system states in G. By Lemma 6, the initial $\sigma_{s,\varepsilon}$ is almost-sure winning at any state $s \in W_{as}^{in} = S$ in G^{in}. In Step 19, $\hat{\sigma}_s$ is updated as a memoryless almost-sure winning system strategy in $\hat{G}$. By Lemma 9, $\sigma_{s,\varepsilon} = \textbf{RecoverHatStrategy}(\hat{G}, \hat{\sigma}_s)$ is also almost-sure winning in G. Finally by Lemma 5, the updated system strategy $\sigma_{s,\varepsilon}$ is also almost-sure winning in G^{in}.

6.6.2. Proof of Discounted-Sum Objective

We follow a similar proof as in [28] to show that the future expected discounted reward of $\sigma_{s,\varepsilon}$ is ε-optimal at the infinitely-often-visited states. There are several parts:

- Any transition distribution can be learned with arbitrarily small positive error bound with confidence arbitrarily close to 1, if the transition is taken for reasonably many times. (Lemma 11)
- (Implicit exploration or exploitation) Suppose that all known transition distributions are approximated with enough accuracy. Given $p \in (0, 1)$, there exists a constant N_p which is polynomial in $1/\varepsilon$ such that if $\sigma_{s,\varepsilon}$ constructed in Step 20 of Algorithm 5 is taken for N_p steps from the current state, either the N_p-step expected discounted reward is ε-optimal in the real game G, or the probability of taking an unknown transition is at least p. (Lemma 13)

- With high confidence, the number of N_p-step periods in which the discounted reward are not ε-optimal is bounded by a polynomial of $|S|, |A|$ and $1/p$.

The following lemma shows that if a state-action pair (s, a) is taken for sufficiently many times, the transition distribution $T(\cdot \mid s, a)$ can be approximated with arbitrarily small error with high confidence.

Lemma 11. *Let $G = \langle S, S_s, S_e, I, A, T, \mathcal{R}, F\rangle$ be a turn-based stochastic Büchi game. For arbitrarily small $\varepsilon_T > 0$ and $\delta_4 \in (0, 1)$, if a state-action pair (s, a) with $s \in S$ and $a \in A^G(s)$ is taken for at least $K = \frac{1}{2\varepsilon_T^2} \log \frac{2|A||S|^2}{\delta_4}$ times, then with probability at least $(1 - \delta_4)$, the estimated transition distribution $\bar{T}(s, a)$ satisfies $|\bar{T}(s' \mid s, a) - T(s' \mid s, a)| \leq \varepsilon_T$ for all $s' \in S$.*

Proof. Let $(s, a, s') \in S \times A \times S$ be an existing transition of G. Let $X_1, \cdots, X_K$ be a sequence of independent and identically distributed binary random variables. For each $i = 1, \cdots, K$, $X_i \in \{0, 1\}$ and $X_i = 1$ if and only if the successor state is s' when a is taken from s for the i^{th} time. Thus $\mathbb{E}(X_i) = T(s' \mid s, a) \in [0, 1]$. By Hoeffdings inequality [73],

$$Pr\Big(\Big| \frac{1}{K} \sum_{i=1}^{K} X_i - T(s' \mid s, a) \Big| \geq \varepsilon_T \Big) \leq 2e^{-2\varepsilon_T^2 K}. \tag{6.1}$$

holds for any

$$\varepsilon_T \in \Big(0, \min\big(T(s' \mid s, a),\ 1 - T(s' \mid s, a)\big)\Big).$$

If

$$K \geq \frac{1}{2\varepsilon_T^2} \log \frac{2|A||S|^2}{\delta_4},$$

then

$$Pr\Big(\Big| \frac{1}{K} \sum_{i=1}^{K} X_i - T(s' \mid s, a) \Big| \geq \varepsilon_T \Big) \leq \frac{\delta_4}{|S|^2|A|}.$$

In other words, if the state-action pair (s, a) is taken for K times, then $\frac{1}{K} \sum_{i=1}^{K} X_i$ estimates $T(s' \mid s, a)$ with error no more than ε_T with probability no less than $\big(1 - \frac{\delta_4}{|S|^2|A|}\big)$. As there cannot be more than $|S|^2|A|$ such transitions, the probability that all transitions that are taken for at least K times are

ε_T-accurate is at least $(1 - \delta_4)$. $\square$

Then we compare the value functions of two games with similar transition distributions. To be specific, we compare the value functions of a game and its ε_T-*approximation*, which is defined as below.

Definition 6. *Let $G_1 = \langle S, S_s, S_e, I, A, T_1, \mathcal{R}, F\rangle$ and $G_2 = \langle S, S_s, S_e, I, A, T_2, \mathcal{R}, F\rangle$ be two turn-based stochastic Büchi games with different transition distributions. For any $\varepsilon_T > 0$, we say that* G_2 is a ε_T-approximation of G_1, *or* G_2 ε_T-approximates G_1, *if $|T_1(s' \mid s, a) - T_2(s' \mid s, a)| \leq \varepsilon_T$ holds for all $s, s' \in S$ and $a \in A$.*

We introduce the following notations, which will later be used in lemmas and their proofs. Let G be a turn-based stochastic Büchi game, N be a positive integer and $s \in S$.

- $V_G(s, \sigma_s, \sigma_e, N)$ denotes the N-step expected discounted reward when the system takes σ_s and the environment takes σ_e at s in G,
- $V_G(s, \sigma_s, \sigma_e) = \lim_{N\to\infty} V_G(s, \sigma_s, \sigma_e, N)$ denotes the infinite-horizon expected discounted reward,
- $V_G(s, \sigma_s) = \min_{\sigma_e} V_G(s, \sigma_s, \sigma_e)$ denotes the worst-case expected discounted reward when the system takes σ_s at s in G and
- $V^*_G(s) = \max_{\sigma_s} V_G(s, \sigma_s)$ denotes the optimal value of $s \in S$ in G.

The following lemma shows that if two turn-based stochastic games have sufficiently similar transition distributions, the expected discounted rewards with the same pair of system and environment strategies can be arbitrarily close.

Lemma 12. *Let G_1 and G_2 be turn-based stochastic Büchi games with state space S and reward upper bound $\frac{\mathcal{R}_{max}}{1-\gamma}$. Assume that G_2 $\frac{\varepsilon_2(1-\gamma)^2}{\mathcal{R}_{max}|S|K}$-approximates of G_1 for some $K \in \mathbb{N}^+$ and $\varepsilon_2 > 0$. Then for every state $s \in S$, system strategy σ_s and environment strategy σ_e, it holds that*

$$|V_{G_1}(s, \sigma_s, \sigma_e, K) - V_{G_2}(s, \sigma_s, \sigma_e, K)| \leq \varepsilon_2.$$

Proof. The proof of this lemma is the same as that of Lemma 4 in [28], except that the expected discounted reward is bounded by $\frac{\mathcal{R}_{\mathsf{max}}}{(1-\gamma)^2}$ rather than $\mathcal{R}_{\mathsf{max}}$. □

We have shown by Lemma 11 and 12 that if all state-action pairs in G are visited for sufficiently many times, the game model $\bar{G}$ will be arbitrarily close to the ground-truth game G with high confidence. However, the agent is not allowed to arbitrarily reset to any state, therefore it may not be feasible to learn all transitions before computing $\sigma_{s,\varepsilon}$. If there are both known and unknown transitions in $\bar{G}$, the suboptimality of $\sigma_{s,\varepsilon}$ in $\bar{G}$ cannot be effectively used to bound the suboptimality in G. For example, $\sigma_{s,\varepsilon}$ is always ε-optimal in $\bar{G}$, but it can be arbitrarily suboptimal in G. Intuitively, it is only possible to bound the suboptimality of $\sigma_{s,\varepsilon}$ in G if the probability of taking unknown transitions in $\bar{G}$ is small. In the following lemma, we show that for some a finite horizon N, either the probability of taking an unknown transition within N steps is greater than a given value (exploration), or the suboptimality of the current system strategy is bounded (exploitation).

Lemma 13. *(Implicit exploration or exploitation) Let G, $\bar{G}$ and $L : S \times A \to \{0, 1\}$ be defined as in Algorithm 5, and $\varepsilon_3 > 0$, $\varepsilon_4 > 0$, $\varepsilon_T > 0$ and $\delta_3 \in (0, 1)$ as constants. Assume that for any known transition (s, a, s') such that $L(s, a) = 1$, $|\bar{T}(s' \mid s, a) - T(s' \mid s, a)| \leq \varepsilon_T$. Let $\bar{\sigma}_s$ be an ε_3-optimal memoryless system strategy in $\bar{G}$. If the system takes $\bar{\sigma}_s$ from $s_0 \in S$ for $N \geq \frac{\log(\varepsilon_4(1-\gamma)^2/\mathcal{R}_{max})}{\log(\gamma)}$ steps in G, then either the expected discounted reward in G is at least $V^*_G(s_0) - \big(\frac{\mathcal{R}_{max}}{(1-\gamma)^2}\delta_3 + \frac{2\mathcal{R}_{max}|S|N\varepsilon_T}{(1-\gamma)^2} + 2\varepsilon_4 + \varepsilon_3\big)$, or it takes an unknown transition with probability no less than δ_3.*

Proof. There are multiple games in Algorithm 5 and in this proof. For clarification, we summarize their interpretation as follows to avoid confusion.

- $G^{in} = \langle S^{in}, S^{in}_s, S^{in}_e, I^{in}, A^{in}, T^{in}, \mathcal{R}^{in}, F^{in}\rangle$ is the input (ground-truth) turn-based stochastic Büchi game whose transition distribution function and reward function are unknown.
- $G = \langle S, S_s, S_e, I, A, T, \mathcal{R}, F\rangle$ is a subgame of G^{in} when the state space is restricted to the

almost-sure winning region W_{as}^{in}.

- $\bar{G} = \langle S, S_s, S_e, I, A, \bar{T}, \bar{\mathcal{R}}, F \rangle$ is an approximate model of G that is learned from exploration. For any known transition (s, a, s') such that $L(s, a) = 1$, it holds that

$$\left| T(s' \mid s, a) - \bar{T}(s' \mid s, a) \right| \leq \varepsilon_T,$$

$$\bar{R}(s, a, s') = \mathcal{R}(s, a, s').$$

For any other transition (s, a, s'),

$$\bar{T}(s' \mid s, a) = \frac{1}{|E^G(s, a)|},$$

$$\bar{R}(s, a, s') = \frac{\mathcal{R}_{\mathsf{max}}}{1 - \gamma}.$$

The value $\frac{\mathcal{R}_{\mathsf{max}}}{1-\gamma}$ is selected such that regardless of the strategies taken by the environment and the system, the expected discounted reward in G is upper bounded by that in $\bar{G}$.

- $\hat{G} = \textbf{HatGame}(\bar{G}, \bar{Q}^*, \varepsilon_1, p_{\varepsilon_1})$ is an auxiliary game constructed from $\bar{G}$ such that

  1. any memoryless system strategy $\hat{\sigma}_s$ in $\hat{G}$ can derive a memoryless ε-optimal system strategy $\bar{\sigma}_s = \textbf{RecoverHatStrategy}(\hat{G}, \hat{\sigma}_s)$ in $\bar{G}$ (Lemma 8);

  2. $\bar{\sigma}_s$ is almost-sure winning in $\bar{G}$ if and only if $\hat{\sigma}_s$ is almost-sure winning in $\hat{G}$ (Lemma 9);

- $\bar{G}_L = \langle S, S_s, S_e, I, A, \bar{T}_L, \bar{\mathcal{R}}_L, F \rangle$ is another game model that mixes the transition distributions and rewards of $\bar{G}$ and G. For each known transition (s, a, s'),

$$\bar{T}_L(s' \mid s, a) = T(s' \mid s, a),$$

$$\bar{\mathcal{R}}_L(s, a, s') = \mathcal{R}(s, a, s').$$

For any other transition (s, a, s'),

$$\bar{T}_L(s' \mid s, a) = \bar{T}(s' \mid s, a),$$

$$\bar{\mathcal{R}}_L(s, a, s') = \bar{\mathcal{R}}(s, a, s') = \frac{\mathcal{R}_{\max}}{1-\gamma}.$$

Remember that a transition (s, a, s') is known if $L(s, a) = 1$. By construction, $\bar{G}_L$ ε_T-approximates $\bar{G}$.

Let (σ_s^*, σ_e^*) be a pair of optimal strategies for the system and the environment in G, $(\bar{\sigma}_s^*, \bar{\sigma}_e^*)$ be a pair of optimal strategies for the system and the environment in $\bar{G}$. To show the result in Lemma 13, we show that if the probability of taking an unknown transition in

$$N \geq \frac{\log(\varepsilon_4(1-\gamma)^2/\mathcal{R}_{\max})}{\log(\gamma)}$$

steps is less than δ_3, then $V_G(s_0, \bar{\sigma}_s, \sigma_e', N)$ is lower bounded by

$$V_G^*(s_0) - \big(\frac{\mathcal{R}_{\max}}{(1-\gamma)^2}\delta_3 + \frac{2\mathcal{R}_{\max}|S|N\varepsilon_T}{(1-\gamma)^2} + 2\varepsilon_4 + \varepsilon_3\big).$$

Step 1. First, since $\mathcal{R}$ and $\bar{\mathcal{R}}$ are non-negative and bounded by $\mathcal{R}_{\max}$ and $\frac{\mathcal{R}_{\max}}{1-\gamma}$ respectively, it holds for any system strategy σ_s' and environment strategy σ_e' that

$$V_G(s_0, \sigma_s', \sigma_e', N) \geq V_G(s_0, \sigma_s', \sigma_e') - \frac{\mathcal{R}_{\max}\gamma^N}{1-\gamma},$$

$$V_{\bar{G}}(s_0, \sigma_s', \sigma_e', N) \geq V_{\bar{G}}(s_0, \sigma_s', \sigma_e') - \frac{\mathcal{R}_{\max}\gamma^N}{(1-\gamma)^2}.$$

As $\bar{\sigma}_s$ is ε_3-optimal in $\bar{G}$, it holds that

$$V_{\bar{G}}(s_0, \bar{\sigma}_s, \sigma_e') \geq \min_{\sigma_e''} V_{\bar{G}}(s_0, \bar{\sigma}_s, \sigma_e'') \geq V_{\bar{G}}^*(s_0) - \varepsilon_3.$$

Then

$$V_{\bar{G}}(s_0, \bar{\sigma}_s, \sigma'_e, N) \geq V^*_{\bar{G}}(s_0) - \frac{\mathcal{R}_{\max}\gamma^N}{(1-\gamma)^2} - \varepsilon_3.$$

As $N \geq \frac{\log(\varepsilon_4(1-\gamma)^2/\mathcal{R}_{\max})}{\log(\gamma)}$, we know

$$\frac{\mathcal{R}_{\max}\gamma^N}{1-\gamma} \leq \frac{\mathcal{R}_{\max}\gamma^N}{(1-\gamma)^2} \leq \varepsilon_4.$$

Therefore

$$\begin{aligned} V_G(s_0, \sigma'_s, \sigma'_e, N) &\geq V_G(s_0, \sigma'_s, \sigma'_e) - \varepsilon_4, \\ V_{\bar{G}}(s_0, \bar{\sigma}_s, \sigma'_e, N) &\geq V^*_{\bar{G}}(s_0) - \varepsilon_4 - \varepsilon_3. \end{aligned} \tag{6.2}$$

Since $\bar{\mathcal{R}}$ is nonnegative,

$$V_{\bar{G}}(s_0, \sigma'_s, \bar{\sigma}^*_e, N) \leq V_{\bar{G}}(s_0, \sigma'_s, \bar{\sigma}^*_e) \leq V_{\bar{G}}(s_0, \bar{\sigma}^*_s, \bar{\sigma}^*_e) = V_{\bar{G}}(s_0). \tag{6.3}$$

Step 2. Then we show that if the probability of taking an unknown transition in N steps is bounded by δ_3, the difference between the N-step expected discounted rewards in G and $\bar{G}_L$ is bounded when the system takes $\bar{\sigma}_s$, regardless of the environment strategy σ'_e. That is,

$$|V_{\bar{G}_L}(s_0, \bar{\sigma}_s, \sigma'_e, N) - V_G(s_0, \bar{\sigma}_s, \sigma'_e, N)| \leq \frac{\mathcal{R}_{\max}}{(1-\gamma)^2}\delta_3. \tag{6.4}$$

The idea is the same as the proof of Lemma 6 in [28].

For any environment strategy σ'_e, let $U(s_0, \bar{\sigma}_s, \sigma'_e, N)$ be the set of N-step run segments when the system takes $\bar{\sigma}_s$ and the environment takes σ'_e at s_0. Since all the games G, $\bar{G}$ and $\bar{G}_L$ share the same set of transitions, their induced sets of runs are the same. Depending on whether there exists an unknown transition in each N-step run segment, we partition $U(s_0, \bar{\sigma}_s, \sigma'_e, N)$ into two non-overlapping subsets $\Omega(s_0, \bar{\sigma}_s, \sigma'_e, N)$ and $\Lambda(s_0, \bar{\sigma}_s, \sigma'_e, N)$ such that each $\pi \in \Omega(s_0, \bar{\sigma}_s, \sigma'_e, N)$ contains some unknown transitions. For each $\pi \in U(s_0, \bar{\sigma}_s, \sigma'_e, N)$, define $P_G(\pi \mid \bar{\sigma}_s, \sigma'_e)$ as the probability of generating π by taking $(\bar{\sigma}_s, \sigma'_e)$ from s_0 in G and $R^{\mathcal{R}}(\pi)$ as the N-step discounted

reward of π with reward function $\mathcal{R}$. For notational simplicity, we use U, Ω and Λ to represent $U(s_0, \bar{\sigma}_s, \sigma'_e, N)$, $\Omega(s_0, \bar{\sigma}_s, \sigma'_e, N)$ and $\Lambda(s_0, \bar{\sigma}_s, \sigma'_e, N)$.

By construction of $\bar{G}_L$, it holds for all $\lambda \in \Lambda(s_0, \bar{\sigma}_s, \sigma'_e, N)$ that

$$\begin{aligned} R^{\mathcal{R}}(\lambda) &= R^{\bar{\mathcal{R}}_L}(\lambda), \\ P_{\bar{G}_L}(\lambda \mid \bar{\sigma}_s, \sigma'_e) &= P_G(\lambda \mid \bar{\sigma}_s, \sigma'_e). \end{aligned}$$

Therefore

$$\sum_{\lambda \in \Lambda} P_{\bar{G}_L}(\lambda \mid \bar{\sigma}_s, \sigma'_e) R^{\bar{\mathcal{R}}_L}(\lambda) = \sum_{\lambda \in \Lambda} P_G(\lambda \mid \bar{\sigma}_s, \sigma'_e) R^{\mathcal{R}}(\lambda). \tag{6.5}$$

If the probability of taking an unknown transition in N steps is bounded by δ_3,

$$\sum_{\omega \in \Omega} P_{\bar{G}_L}(\omega \mid \bar{\sigma}_s, \sigma'_e) = \sum_{\omega \in \Omega} P_G(\omega \mid \bar{\sigma}_s, \sigma'_e) \leq \delta_3.$$

As $R^{\bar{\mathcal{R}}_L}(\omega)$ and $R^{\mathcal{R}}(\omega)$ are nonnegative and bounded by $\frac{\mathcal{R}_{\mathsf{max}}}{(1-\gamma)^2}$ for all $\omega \in \Omega(s_0, \bar{\sigma}_s, \sigma'_e, N)$, it holds that

$$\begin{aligned} & |V_{\bar{G}_L}(s_0, \bar{\sigma}_s, \sigma'_e, N) - V_G(s_0, \bar{\sigma}_s, \sigma'_e, N)| \\ =& \Big| \sum_{\pi \in U} \Big(P_{\bar{G}_L}(\pi \mid \bar{\sigma}_s, \sigma'_e) R^{\bar{\mathcal{R}}_L}(\pi) - P_G(\pi \mid \bar{\sigma}_s, \sigma'_e) R^{\mathcal{R}}(\omega) \Big) \Big| \\ \leq& \Big| \sum_{\lambda \in \Lambda} \Big(P_{\bar{G}_L}(\lambda \mid \bar{\sigma}_s, \sigma'_e) R^{\bar{\mathcal{R}}_L}(\lambda) - P_G(\lambda \mid \bar{\sigma}_s, \sigma'_e) R^{\mathcal{R}}(\lambda) \Big) \Big| + \\ & \Big| \sum_{\omega \in \Omega} \Big(P_{\bar{G}_L}(\omega \mid \bar{\sigma}_s, \sigma'_e) R^{\bar{\mathcal{R}}_L}(\omega) - P_G(\omega \mid \bar{\sigma}_s, \sigma'_e) R^{\mathcal{R}}(\omega) \Big) \Big| \\ =& \Big| \sum_{\omega \in \Omega} \Big(P_{\bar{G}_L}(\omega \mid \bar{\sigma}_s, \sigma'_e) R^{\bar{\mathcal{R}}_L}(\omega) - P_G(\omega \mid \bar{\sigma}_s, \sigma'_e) R^{\mathcal{R}}(\omega) \Big) \Big| \\ \leq& \Big| \sum_{\omega \in \Omega} \Big(P_{\bar{G}_L}(\omega \mid \bar{\sigma}_s, \sigma'_e) \frac{\mathcal{R}_{\mathsf{max}}}{(1-\gamma)^2} - P_G(\omega \mid \bar{\sigma}_s, \sigma'_e) \cdot 0 \Big) \Big| \\ \leq& \frac{\mathcal{R}_{\mathsf{max}}}{(1-\gamma)^2} \delta_3, \end{aligned}$$

which proves (6.4).

Step 3. Now we show that for any system strategy σ'_s and environment strategy σ'_e,

$$V_{\bar{G}_L}(s_0, \sigma'_s, \sigma'_e, N) \geq V_G(s_0, \sigma'_s, \sigma'_e, N). \tag{6.6}$$

For all $i \in \{1, \cdots, N\}$ and $\tau \in \Lambda(s_0, \sigma'_s, \sigma'_e, i-1)$, define $U_{\tau,N} \subseteq \Omega(s_0, \sigma'_s, \sigma'_e, N)$ as the set of N-step run segments with the prefix τ such that τ only contains known transitions and the transition following τ is unknown, that is, $L(s^{i-1}, a^{i-1}) = 0$. Therefore it holds that

$$\Omega(s_0, \sigma'_s, \sigma'_e, N) = \bigcup_{i=1}^{N} \bigcup_{\tau \in \Lambda(s_0, \sigma'_s, \sigma'_e, i-1)} U_{\tau,N}. \tag{6.7}$$

As $T(s,a) = \bar{T}_L(s,a)$ for all known (s,a), it holds for all $i \in \{1, \cdots, N\}$ and $\tau \in \Lambda(s_0, \sigma'_s, \sigma'_e, i-1)$ that

$$\sum_{\omega \in U_{\tau,N}} P_G(\omega \mid \sigma'_s, \sigma'_e) = \sum_{\omega \in U_{\tau,N}} P_{\bar{G}_L}(\omega \mid \sigma'_s, \sigma'_e),$$

$$R^{\mathcal{R}}(\tau) = R^{\bar{\mathcal{R}}_L}(\tau).$$

For each $\omega \in U^{\Omega}(\tau, \sigma'_s, \sigma'_e, N)$, the i^{th} transition visited in ω is unknown and thus the reward in $\bar{G}_L$ is $\frac{\mathcal{R}_{\max}}{1-\gamma}$. Since the reward function $\mathcal{R}$ of G is bounded by $\mathcal{R}_{\max}$, the total discounted reward after i steps in G is upper bounded by $\gamma^i \frac{\mathcal{R}_{\max}}{1-\gamma}$, which is exactly the the discounted reward at the i^{th} step of ω in $\bar{G}_L$. As both $\mathcal{R}$ and $\bar{\mathcal{R}}_L$ are nonnegative, $R^{\mathcal{R}}(\omega) \leq R^{\bar{\mathcal{R}}_L}(\omega')$ holds for any $\omega, \omega' \in U^{\Omega}(\tau, \sigma'_s, \sigma'_e, N)$. Therefore

$$\sum_{\omega \in U_{\tau,N}} P_G(\omega \mid \bar{\sigma}_s, \sigma'_e) R^{\mathcal{R}}(\omega) \leq \sum_{\omega \in U_{\tau,N}} P_{\bar{G}_L}(\omega \mid \sigma'_s, \sigma'_e) R^{\bar{\mathcal{R}}_L}(\omega) \tag{6.8}$$

holds for all $i \in \{1, \cdots, N\}$ and $\tau \in \Lambda(s_0, \sigma'_s, \sigma'_e, i-1)$.

$$
\begin{aligned}
&V_{\bar{G}_L}(s_0, \bar{\sigma}_s, \sigma'_e, N) - V_G(s_0, \bar{\sigma}_s, \sigma'_e, N) \\
=&\sum_{\pi \in U} \Big(P_{\bar{G}_L}(\pi \mid \bar{\sigma}_s, \sigma'_e) R^{\bar{\mathcal{R}}_L}(\pi) - P_G(\pi \mid \bar{\sigma}_s, \sigma'_e) R^{\mathcal{R}}(\pi) \Big) \\
=&\sum_{\lambda \in \Lambda} \Big(P_{\bar{G}_L}(\lambda \mid \bar{\sigma}_s, \sigma'_e) R^{\bar{\mathcal{R}}_L}(\lambda) - P_G(\lambda \mid \bar{\sigma}_s, \sigma'_e) R^{\mathcal{R}}(\lambda) \Big) + \\
&\sum_{\omega \in \Omega} \Big(P_{\bar{G}_L}(\omega \mid \bar{\sigma}_s, \sigma'_e) R^{\bar{\mathcal{R}}_L}(\omega) - P_G(\omega \mid \bar{\sigma}_s, \sigma'_e) R^{\mathcal{R}}(\omega) \Big).
\end{aligned} \tag{6.9}
$$

Substituting (6.5) and (6.8) into (6.9), we can prove that

$$
V_{\bar{G}_L}(s_0, \bar{\sigma}_s, \sigma'_e, N) \geq V_G(s_0, \bar{\sigma}_s, \sigma'_e, N). \tag{6.10}
$$

Step 4. As $\bar{G}_L$ is an ε_T-approximation of $\bar{G}$, Lemma 12 guarantees that

$$
\begin{aligned}
|V_{\bar{G}_L}(s_0, \bar{\sigma}_s, \sigma'_e, N) - V_{\bar{G}}(s_0, \bar{\sigma}_s, \sigma'_e, N)| &\leq \frac{\mathcal{R}_{\mathsf{max}} |S| N \varepsilon_T}{(1-\gamma)^2}, \\
|V_{\bar{G}_L}(s_0, \sigma^*_s, \bar{\sigma}^*_e, N) - V_{\bar{G}}(s_0, \sigma^*_s, \bar{\sigma}^*_e, N)| &\leq \frac{\mathcal{R}_{\mathsf{max}} |S| N \varepsilon_T}{(1-\gamma)^2}.
\end{aligned} \tag{6.11}
$$

Step 5. We can now finish the proof.

As σ^*_s and σ^*_e are a pair of optimal strategies in G, $V^*_G(s_0) = V_G(s_0, \sigma^*_s, \sigma^*_e)$. Since $\bar{\sigma}_e$ may not be

optimal in G, $V_G(s_0, \sigma_s^*, \sigma_e^*) \leq V_G(s_0, \sigma_s^*, \bar{\sigma}_e^*)$. By (6.2), (6.6) and (6.11),

$$
\begin{aligned}
V_G^*(s_0) &= V_G(s_0, \sigma_s^*, \bar{\sigma}_e^*) \\
&\overset{\text{By (6.2)}}{\leq} V_G(s_0, \sigma_s^*, \bar{\sigma}_e^*, N) + \varepsilon_4 \\
&\overset{\text{By (6.10)}}{\leq} V_{\bar{G}_L}(s_0, \sigma_s^*, \bar{\sigma}_e^*, N) + \varepsilon_4 \\
&\overset{\text{By (6.11)}}{\leq} V_{\bar{G}}(s_0, \sigma_s^*, \bar{\sigma}_e^*, N) + \varepsilon_4 + \frac{\mathcal{R}_{\mathsf{max}}|S|N\varepsilon_T}{(1-\gamma)^2} \\
&\overset{\text{By (6.3)}}{\leq} V_{\bar{G}}^*(s_0) + \varepsilon_4 + \frac{\mathcal{R}_{\mathsf{max}}|S|N\varepsilon_T}{(1-\gamma)^2} \\
&\overset{\text{By (6.2)}}{\leq} V_{\bar{G}}(s_0, \bar{\sigma}_s, \sigma_e', N) + 2\varepsilon_4 + \frac{\mathcal{R}_{\mathsf{max}}|S|N\varepsilon_T}{(1-\gamma)^2} + \varepsilon_3 \\
&\overset{\text{By (6.11)}}{\leq} V_{\bar{G}_L}(s_0, \bar{\sigma}_s, \sigma_e', N) + \frac{2\mathcal{R}_{\mathsf{max}}|S|N\varepsilon_T}{(1-\gamma)^2} + 2\varepsilon_4 + \varepsilon_3.
\end{aligned}
\tag{6.12}
$$

We assume that the probability of taking a transition (s, a, s') such that $L(s, a) = 0$ is less than δ_3. By (6.4) we get

$$
\begin{aligned}
&V_G^*(s_0) - V_G(s_0, \bar{\sigma}_s, \sigma_e', N) \\
&\overset{\text{By (6.12)}}{\leq} V_{\bar{G}_L}(s_0, \bar{\sigma}_s, \sigma_e', N) - V_G(s_0, \bar{\sigma}_s, \sigma_e', N) + \frac{2\mathcal{R}_{\mathsf{max}}|S|N\varepsilon_T}{(1-\gamma)^2} + 2\varepsilon_4 + \varepsilon_3 \\
&\overset{\text{By (6.4)}}{\leq} \frac{\mathcal{R}_{\mathsf{max}}}{(1-\gamma)^2}\delta_3 + \frac{2\mathcal{R}_{\mathsf{max}}|S|N\varepsilon_T}{(1-\gamma)^2} + 2\varepsilon_4 + \varepsilon_3.
\end{aligned}
$$

Therefore we have proved that if the probability of taking a transition (s, a, s') such that $L(s, a) = 0$ is less than δ_3, then

$$
V_G(s_0, \bar{\sigma}_s, \sigma_e', N) \geq V_G^*(s_0) - \big(\frac{\mathcal{R}_{\mathsf{max}}}{(1-\gamma)^2}\delta_3 + \frac{2\mathcal{R}_{\mathsf{max}}|S|N\varepsilon_T}{(1-\gamma)^2} + 2\varepsilon_4 + \varepsilon_3\big)
$$

holds for any environment strategy σ_e'. □

We now show the satisfaction of the discounted-sum objective as stated in Theorem 6, that is, with probability no less than $(1-\delta_c)$, the future expected discounted reward of the current state s when the system takes $\sigma_{s,\varepsilon}$ is at least $V^*_G(s)-\varepsilon$, except for some number of steps that is polynomial in $|S|, |A|, \frac{1}{\varepsilon}$ and $\frac{1}{\delta_c}$. The system strategy $\bar{\sigma}_s$ in Lemma 13 corresponds to the strategy $\sigma_{s,\varepsilon}$ in Algorithm 5.

Proof. By Lemma 8, the system strategy $\sigma_{s,\varepsilon}$ constructed from **HatGame** and **RecoverHatStrategy** with parameters $\varepsilon_1 = \frac{1-\gamma}{12}\varepsilon$, and $p_{\varepsilon_1} = \frac{\varepsilon(1-\gamma)^3}{12\mathcal{R}_{\mathsf{max}}}$ is $\frac{\varepsilon}{6}$-optimal for the system in $\bar{G}$.

In Step 13, a state-action pair $(s,a) \in S \times A$ is relabeled as known if it is visited for at least $K = \frac{1}{2\delta^2}\log\frac{4|A||S|^2}{\delta_c}$ times. By Lemma 11, with probability at least $\left(1-\frac{\delta_c}{2|S|^2|A|}\right)$,

$$\left|\bar{T}(s' \mid s,a) - T(s' \mid s,a)\right| \leq \delta$$

holds for each known (s,a) and $s' \in S$. As there are at most $|S|^2|A|$ different transitions (s,a,s'), the probability that all known transitions are estimated with precision δ is at least $(1-\frac{\delta_c}{2})$.

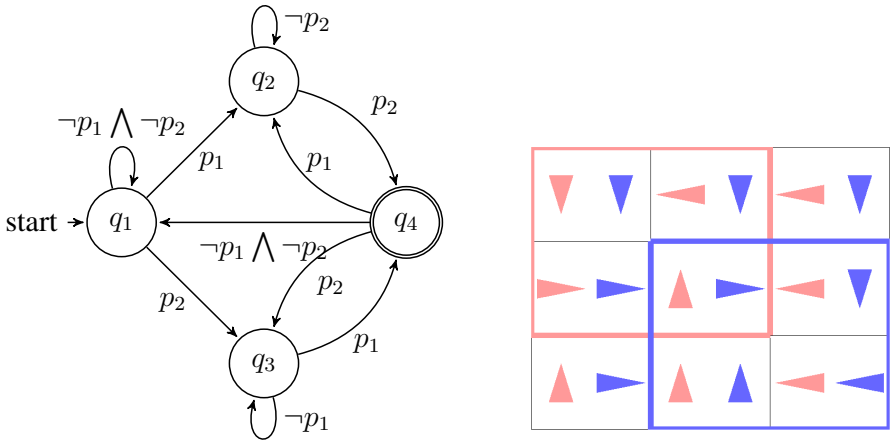
Then we estimate the number of steps necessary to learn an ε-optimal strategy for the system. Let

$$\begin{aligned}
&\varepsilon_3 = \varepsilon_4 = \frac{\varepsilon}{6},\ \delta_3 = \frac{\varepsilon(1-\gamma)^2}{6\mathcal{R}_{\mathsf{max}}},\\
&N = \frac{\log(\varepsilon_4(1-\gamma)^2/\mathcal{R}_{\mathsf{max}})}{\log(\gamma)} = \frac{\log(\varepsilon(1-\gamma)^2/6\mathcal{R}_{\mathsf{max}})}{\log\gamma},\\
&\varepsilon_T = \delta = \frac{\varepsilon(1-\gamma)^2\log\gamma}{6\mathcal{R}_{\mathsf{max}}|S|\log(\varepsilon(1-\gamma)^2/6\mathcal{R}_{\mathsf{max}})},
\end{aligned}$$

then by Lemma 13, if the system takes $\sigma_{s,\varepsilon}$ from $s_0 \in S$ for N steps in G, then either the probability to take an unknown transition is no less than δ_3, or the suboptimality is bounded by

$$\frac{\mathcal{R}_{\mathsf{max}}}{(1-\gamma)^2}\delta_3 + \frac{2\mathcal{R}_{\mathsf{max}}|S|N\varepsilon_T}{(1-\gamma)^2} + 2\varepsilon_4 + \varepsilon_3 = \frac{\varepsilon}{6} + \frac{\varepsilon}{3} + 2\cdot\frac{\varepsilon}{6} + \frac{\varepsilon}{6} = \varepsilon.$$

Thus Lemma 13 guarantees that if all known transitions are approximated with precision δ, then either the expected discounted reward in N steps is ε-optimal, or the agent takes an unknown transition



(a) (b)

Figure 17: (17a) A DBA constructed for the example. p_1 stands for the lower left block, and p_2 stands for the upper right block. (17b) The optimal strategy for the system with only the discounted reward. The pink and blue squares represent the dangerous areas when the light is on and off. The triangles show the optimal transition directions from each block (pink ones for light on, blue ones for light off).

with probability at least δ_3.

We now show that the number of N-step periods in which the probability of taking unknown transitions is at least δ_3 can be polynomially bounded. As there are at most $|S||A|$ state-action pairs in G, at most $|S||A|K$ exploration steps can be taken before learning all transitions in G. Again by the Hoeffding's inequality, we can show that if the probability of taking unknown transitions in each period is at least δ_3, there exists K_2 which is polynomial in $|S|$, $|A|$, $\frac{1}{\varepsilon}$ and $\frac{1}{\delta_c}$ such that with confidence at least $(1 - \frac{\delta_c}{2})$, $|S||A|K$ explorations will be made within K_2 N-periods. Therefore with probability at least $(1 - \delta_c)$, the system by running Algorithm 5 behaves ε-optimally except for at most some polynomial number of steps, which completes the proof. □

6.7. Experimental Results

We show the usage of our algorithm with a robot motion planning problem which involves simultaneous resource collection and surveillance. This example was run on a laptop with an 8 Intel(R) Core(TM) 2.40GHz CPU and 8 GB memory.

We first introduce the turn-based game and task requirements. The system moves in a 3-by-3 grid world, and it has to move to an adjacent block if the current state is a system state. The environment is a signal light that indicates the dangerous area in the world at the current step which needs to be monitored closely. If the environment light is on, the upper left four blocks are dangerous; otherwise, the lower right four blocks are dangerous. The environment can arbitrarily decide the status of the light in the next step if the current state is an environment state. Furthermore, the lower left block and the upper right block are labeled as post offices. For ease of demonstration, we assume that all transitions are deterministic. In other words, $|E^G(s,a)| = 1$ for all state-action pair (s,a).

We want to learn a strategy for the system to both patrol the dangerous areas and persistently visit the two post offices. We first interpret the task requirements as a almost-sure winning objective and a discounted-sum objective, encode them as inputs to our algorithm, and then show the results.

Almost-Sure Winning Objective. The task of visiting the two post offices can be expressed by the four-state DBA in Figure 17a. The initial state is q_1, and the set of accepting states is $\{q_4\}$. The upper right block is labeled by 'p_1' ('post office #1') and the lower left block is labeled by 'p_2' ('post office #2'). We show that the Büchi condition is satisfied if and only if both p_1 and p_2 are visited infinitely often. Starting from the initial state, the system transits to q_2 if it visits p_1, or transits to q_3 if it visits p_2. If it visits neither of them, it stays at q_1. From q_2 and q_3, the system should visit the other post office (p_2 for q_2 and p_1 for q_3) in order to enter q_4. q_4 has the same outgoing transitions as q_1. The transitions show that one new visit to q_4 requires at least one new visit to p_1 and one new visit to p_2. Therefore to satisfy the Büchi condition, that is, to visit q_4 infinitely often, the system has to visit p_1 and p_2 infinitely often. All initial states are with DBA state q_1.

The Discounted-Sum Objective. The task of monitoring the dangerous area is interpreted as a discounted-sum objective. A reward function is designed to encourage the system to patrol the dangerous area. The system will be rewarded by 1 in the following cases: (1) when the system transits into the dangerous area; (2) when the light is on and the system moves counterclockwise in the dangerous area; (3) when the light is off and the system moves clockwise in the dangerous area.

For all other system transitions and all environment transitions, there is no reward. Throughout this example, the discount factor γ is 0.6, and $\mathcal{R}_{\mathsf{max}} = 1$.

The system does not know this reward function ahead of time, but eventually manages to learn a strategy with optimal worst-case discounted reward. As shown in Figure 17b, the system learns to approach the area specified by the environment as soon as possible and then move in the corresponding direction to maximize the reward.

We get the product of the original turn-based game and the DBA, which results in a turn-based Büchi game G^{in}. The almost-sure winning region W_{as}^{in} and a memoryless almost-sure winning strategy σ_s are computed with the off-the-shelf tool PGSolver [58]. It turns out that W_{as}^{in} is the whole state space, and σ_s is illustrated in Figure 19a. The suboptimality bound ε is set to be 0.0001. To output the learned strategy in a timely manner, we added a terminating condition to the while loop in Algorithm 5 such that the algorithm stops if there are no updates in the last 10,000 steps.

Upon termination, the learned strategy for the system is shown in Figure 19b. The strategy is randomized and allows two actions at each system state, one with probability $(1 - p_{\varepsilon_1})$ and the other

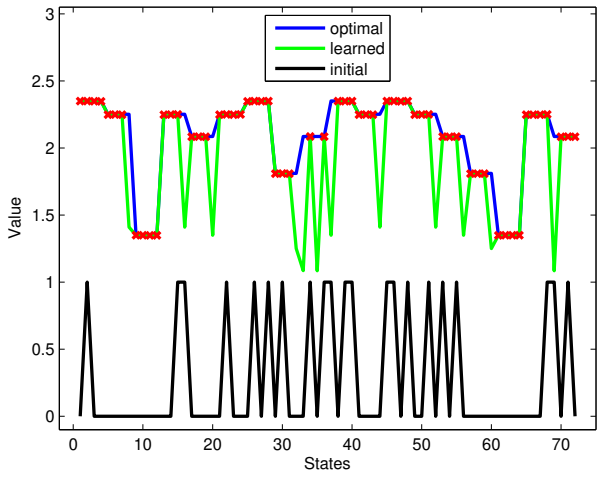


Figure 18: Comparison of the value function of the initial almost-sure winning strategy, the learned strategy and an optimal strategy (which may not be almost-sure winning) for all system states. The red crosses mark all the strongly connected components in which there is at least one state whose value is learned to be ε-optimal.

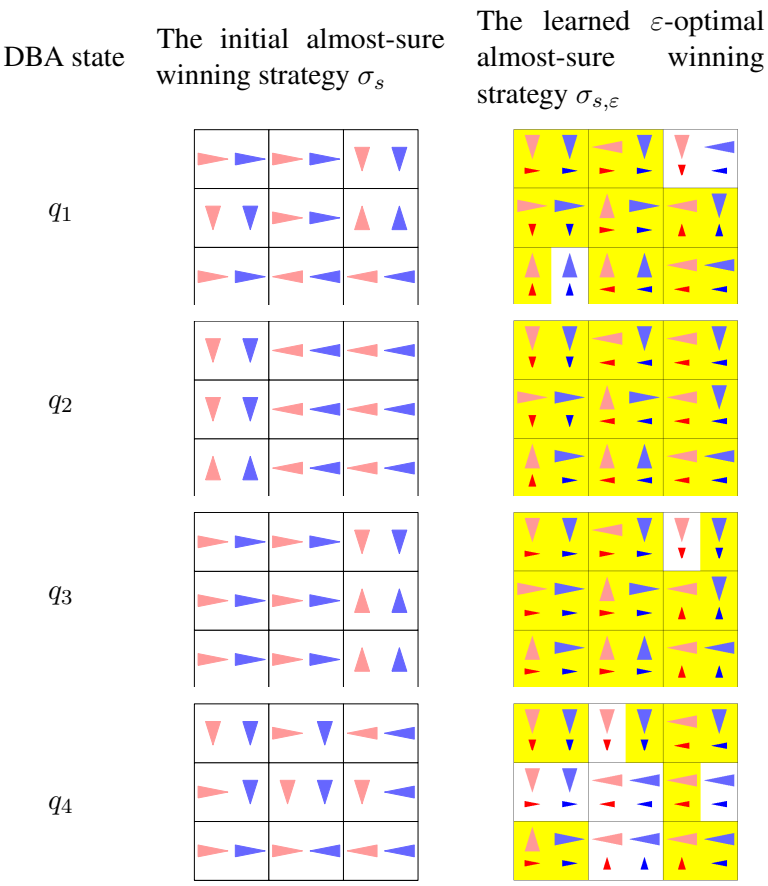


Figure 19: Illustration of the initial almost-sure winning strategy σ_s (the middle column) and the learned ε-optimal almost-sure winning system strategy $\sigma_{s,\varepsilon}$ (the right column). From top to bottom, the four figures in each column show the system strategy with DBA state q_1 to q_4. In each figure, the pink and blue triangles point to the transition directions at each block when the light is on and off respectively. In the right column, big triangles represent actions with probability $(1 - p_{\varepsilon_1})$, and small triangles represent actions with probability p_{ε_1}. Triangles with yellow background are ε-optimal over all almost-sure winning system strategies.

with probability p_{ε_1}, represented by the big triangles and small triangles respectively. The worst-case value functions for the learned strategy, the initial almost-sure winning strategy, and an optimal strategy are shown in Figure 18. These value functions are evaluated with the true reward function, and thus are not accessible to the system. It can be found that the value of the learned strategy is much better than that of the initial strategy, although it is not at all close to the optimal strategy. In Figure 19, we marked all states where the learned strategy is ε-optimal with yellow background. It turns out that all system states are marked with yellow, i.e., are ε-optimal, except those that are not reachable from the initial states. The product Büchi game has 144 states, 72 system states and 192

transitions. On average of ten repetitive experiments, the algorithm terminates at 58.05 seconds with the last update occurs at 29.77 seconds.

Chapter 7: Conclusion

Reinforcement learning (RL) algorithms solve sequential decision-making problems by accumulating intermediate feedback as rewards from an environment, and gradually improve the long-term expected reward. For most RL algorithms, little prior knowledge, such as the underlying dynamics or the analytical form of the reward function, is required. Such flexibility makes it tempting to apply RL techniques in a variety of applications. The downside is the lack of guarantees and understanding of the learned policies that the RL algorithms provide. It is very challenging and demanding to reliably represent high-level task specifications as reward functions. The reward function would have to capture heterogeneous and possibly competing requirements of the task; the resulting optimal policies should achieve high task performance, and the reward functions should work not just in the training environment, but also in similar testing environments. Moreover, RL algorithms are not always capable of converging to globally optimal policies, widening the gap between the learned policy and the high-level task which RL was applied to solve.

This thesis developed reinforcement learning algorithms with high-level task specifications that learn policies with high task performance. It merges and extends ideas from a diverse range of conventionally disparate research fields, including learning from demonstrations, model checking, and reactive synthesis.

Chapter 2 and Chapter 3 combine ideas from the field of learning from demonstrations and model checking. Instead of specifying a reward function, the expert trains learning agents by providing demonstrations of how to implement the task successfully and a temporal logic specification that directly encodes the high-level task requirements. The temporal logic specification can improve the task performance of the learned policies in several ways. First, it is used to automatically construct a memory transition system and extend the state space of the original MDP. Policies with the extended state space are of finite-memory for the original MDP, and thus can implement more tasks than memoryless policies. In essence, each state in the memory transition system corresponds to a different reward function and optimal policy. Second, the temporal logic specifications act as a task

performance criterion for the learned policies, which is objective and independent on the inferred reward function. We observe that the policies learned merely from demonstrations cannot generalize well to states uncovered by expert demonstrations. To overcome this difficulty, we augment the original optimization objective to account for task performance explicitly in Chapter 2. In Chapter 3, we extend the previous framework to nonlinearly parameterized reward functions such as reward networks, which automatically construct reward features by themselves. The resulting algorithm learns a reward network that maps local neighborhoods to reward values and directly applies the learned reward network to new environments with no expert demonstrations. Numerical experiments show that both the memory transition system generated from task specifications and the ability to replan in new environments play critical roles to enable good generalization performance. Related papers include [184, 186].

We solve a constrained RL problem with a novel policy search algorithm in Chapter 4. We use trajectory-based objective and constraint functions to represent high-level task specifications. Compared with state-based or transition-based reward functions, trajectory-based functions are both more expressive and more straightforward to encode task specifications. The proposed algorithm is a variant of an existing cross-entropy algorithm, in which both objective and constraints are assumed to be black boxes. We prove almost-sure asymptotic convergence properties of the proposed algorithm. Although the convergence to global optima is not guaranteed, it is observed to happen with high probability in a constrained linear quadratic regulator example. The related publication is [183].

Chapter 5 and Chapter 6 combine RL with reactive synthesis with temporal logic specifications. The high-level task specifications are represented both qualitatively as a temporal logic specification and quantitatively as a reward function. We model the interaction between the learning agent and its environment as a two-player turn-based zero-sum game. Besides constructing memory transition systems, temporal logic specifications restrict exploration and guarantee safety even during learning. In Chapter 5, we first compute a nondeterministic (possibly maximally) permissive strategy for the given temporal logic specification. The learning agent can only take actions that are allowed by the permissive strategy in exploration. If the permissive strategy is not maximal, the learned policy may

not be globally optimal but is still guaranteed to be winning for the system. In Chapter 6, we propose an online model-based RL algorithm to solve this problem. For the qualitative objective, we compute the almost-sure winning region of the system agent and prune the game graph, such that the learning agent always has a winning policy at any reachable state. For the quantitative objective, we design an auxiliary game for the original game model, such that any policy in the auxiliary game model corresponds to a ε-optimal policy in the original game model. Moreover, the two policies can only be almost-surely winning simultaneously. The proposed algorithm guarantees that the exploration policy is always almost-surely winning in the auxiliary game model. Properties of the auxiliary game model guarantee that the exploration policy is always almost-sure winning for the learning agent, and will be ε-optimal in the ground-truth game if the game model is accurate enough. We show that the proposed algorithm is probably approximately correct, which is the first PAC-learning algorithm in stochastic games with independent quantitative and qualitative objectives. Publications for this topic include [182, 185].

7.1. Future Research Directions

The problem of incorporating high-level task specifications into RL algorithms is just a starting point for a much broader picture of problems: **How to allow RL algorithms to build upon existing knowledge of underlying problems and thus achieve better solutions?** With the environment modeled as a general-purpose MDP with unknown transition distributions and unknown reward functions, an RL agent knows very little about the underlying problem to be solved, which includes transition distributions (system dynamics, uncertainties), constraints (such as game rules) and optimization objectives (such as reward function). Intuitively, lack of prior knowledge of the underlying problem raises many difficulties and limitations for RL algorithms. Which of these difficulties can be addressed by incorporating heterogeneous prior knowledge and how?

One promising research direction is to use prior knowledge to select function approximators for RL problems. Except for problems with moderately sized state spaces, RL algorithms need function approximators to represent policies and value functions. A common practice is to resort to general-purpose function approximators such as fully connected neural networks and Gaussian processes.

Recently, several attempts have been made to compare different function approximators on some commonly used RL benchmarks [115, 145]. It turns out that policies with linear or radial-basis-function-based parameterizations may match or even outperform the performance of policies modeled by fully connected neural networks, yet with much fewer training episodes. Additionally, the variance of the values of the learned policies is high. It remains an open problem to find an optimal function approximator for a given RL problem, or even just what it means to be an optimal function approximator. Prior knowledge of the underlying problem may help facilitate this difficulty. For example, under some assumptions on the quadratic objective function, a constrained linear quadratic regulator problem has a piecewise-affine optimal controller, which can be perfectly represented as a fully connected neural network with ReLU activations. The idea of learning an explicit policy function is also related to the topic of explicit model predictive control, which has been intensively studied [16, 39, 168].

Another exciting direction is to study learning from demonstration problems with abstract task specifications. In this thesis, all the given high-level task specifications are accurate and complete. For example, for each problem which uses temporal logic specifications, there is a well-defined labeling function that explicitly connects each state with a subset of symbolic labels. Therefore, there is no need for learning agents to infer the interpretation of the specifications. Without labeling functions, agents may not be able to interpret task specifications accurately, and that is why the task specifications are called *abstract*. As a result, agents will not be able to directly use the memory transition systems that are built from task specifications. However, it is possible to approach this problem by learning from demonstrations, where the learned policies outputs decide not only which action to take at each state, but also when to transit from one memory state to another. In other words, the learning procedure both benefit from abstract task specifications (by introducing memory states) and help refine the given task information (by predicting the conditions for each memory transition).

BIBLIOGRAPHY

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

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

[3] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[4] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 22–31. JMLR. org, 2017.

[5] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin. Reachability-based safe learning with gaussian processes. In *53rd IEEE Conference on Decision and Control*, pages 1424–1431. IEEE, 2014.

[6] A. Albert and J. A. Anderson. On the existence of maximum likelihood estimates in logistic regression models. *Biometrika*, 71(1):1, 1984. doi: 10.1093/biomet/71.1.1. URL `+http://dx.doi.org/10.1093/biomet/71.1.1`.

[7] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[8] E. Altman. Asymptotic properties of constrained markov decision processes. *Mathematical Methods of Operations Research*, 37(2):151–170, 1993.

[9] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[10] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[11] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin. Provably safe and robust learning-based model predictive control. *Automatica*, 49(5):1216–1226, 2013.

[12] M. Babes, V. Marivate, K. Subramanian, and M. L. Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 897–904, 2011.

[13] C. Baier, J.-P. Katoen, and K. G. Larsen. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

[14] C. L. Baker, J. B. Tenenbaum, and R. R. Saxe. Goal inference as inverse planning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 29, 2007.

[15] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(5):834–846, 1983.

[16] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[17] M. Benaim. A dynamical system approach to stochastic approximations. *SIAM Journal on Control and Optimization*, 34(2):437–472, 1996.

[18] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.

[19] J. Bernet, D. Janin, and I. Walukiewicz. Permissive strategies: from parity games to safety games. *RAIRO-Theoretical Informatics and Applications*, 36(03):261–275, 2002.

[20] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II*. Athena Scientific, 3rd edition, 2007. ISBN 1886529302, 9781886529304.

[21] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[22] F. Blahoudek. Ltl3dra - ltl to deterministic rabin automata translator based on ltl3ba, 2015. URL `http://sourceforge.net/projects/ltl3dra/`.

[23] M. Bloem and N. Bambos. Infinite time horizon maximum causal entropy inverse reinforcement learning. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 4911–4916. IEEE, 2014.

[24] L. Bobadilla, O. Sanchez, J. Czarnowski, K. Gossman, and S. M. LaValle. Controlling wild bodies using linear temporal logic. In *Robotics: Science and systems*, volume 7, page 17, 2012.

[25] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[26] F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[27] A. Boularias, J. Kober, and J. Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[28] R. I. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

[29] T. Brázdil, K. Chatterjee, M. Chmelik, V. Forejt, J. Křetínskỳ, M. Kwiatkowska, D. Parker, and M. Ujma. Verification of markov decision processes using learning algorithms. In *International*

Symposium on Automated Technology for Verification and Analysis, pages 98–114. Springer, 2014.

[30] B. Burchfiel, C. Tomasi, and R. Parr. Distance minimization for reward learning from scored trajectories. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[31] K. Chatterjee and L. Doyen. Energy parity games. In *Automata, Languages and Programming*, pages 599–610. Springer, 2010.

[32] K. Chatterjee and L. Doyen. Perfect-information stochastic games with generalized mean-payoff objectives. In *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10. IEEE, 2016.

[33] K. Chatterjee and T. A. Henzinger. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences*, 78(2):394–413, 2012.

[34] K. Chatterjee, M. Jurdziński, and T. A. Henzinger. Simple stochastic parity games. In *Computer Science Logic*, pages 100–113. Springer, 2003.

[35] K. Chatterjee, L. De Alfaro, and T. A. Henzinger. *The complexity of stochastic Rabin and Streett games*. Springer, 2005.

[36] K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Mean-payoff parity games. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 178–187. IEEE, 2005.

[37] K. Chatterjee, M. Randour, and J.-F. Raskin. Strategy synthesis for multi-dimensional quantitative objectives. In *CONCUR 2012–Concurrency Theory*, pages 115–131. Springer, 2012.

[38] K. Chatterjee, L. Doyen, H. Gimbert, and Y. Oualhadj. Perfect-information stochastic mean-payoff parity games. In *Foundations of Software Science and Computation Structures*, pages 210–225. Springer, 2014.

[39] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas, and M. Morari. Approximating explicit model predictive control using constrained neural networks. In *2018 Annual American Control Conference (ACC)*, pages 1520–1527. IEEE, 2018.

[40] T. Chen, M. Kwiatkowska, A. Simaitis, and C. Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *Quantitative Evaluation of Systems*, pages 322–337. Springer, 2013.

[41] J. Choi and K.-E. Kim. Bayesian nonparametric feature construction for inverse reinforcement learning. In *IJCAI*, pages 1287–1293, 2013.

[42] Y. Chow, M. Ghavamzadeh, L. Janson, and M. Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1): 6070–6120, 2017.

[43] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A lyapunov-based approach

to safe reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 8092–8101, 2018.

[44] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[45] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.

[46] X. Ding, S. L. Smith, C. Belta, and D. Rus. Optimal control of markov decision processes with linear temporal logic constraints. *Automatic Control, IEEE Transactions on*, 59(5):1244–1257, 2014.

[47] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[48] K. Dvijotham and E. Todorov. Inverse optimal control with linearly-solvable MDPs. In *International Conference on Machine Learning*, pages 335–342, 2010.

[49] R. Ehlers and B. Finkbeiner. Reactive safety. In *Proceedings of Symposium on Games, Automata, Logics and Formal Verification*, pages 178–191, 2011.

[50] R. Ehlers, V. Raman, and C. Finucane. Slugs GR(1) synthesizer, 2013. Available at `https://github.com/LTLMoP/slugs`.

[51] L. El Asri, B. Piot, M. Geist, R. Laroche, and O. Pietquin. Score-based inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 457–465. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[52] M. Fahad, Z. Chen, and Y. Guo. Learning how pedestrians navigate: A deep inverse reinforcement learning approach. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 819–826. IEEE, 2018.

[53] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for mobile robots. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2020–2025. IEEE, 2005.

[54] J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer-Verlag New York, Inc., 1996.

[55] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2786–2793. IEEE, 2017.

[56] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[57] O. Friedmann and M. Lange. The pgsolver collection of parity game solvers. *University of Munich*, 2009.

[58] O. Friedmann and M. Lange. tcsprojects/pgsolver, 2015. URL https://github.com/tcsprojects/pgsolver.

[59] J. Fu and U. Topcu. Probably approximately correct mdp learning and control with temporal logic constraints. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014. doi: 10.15607/RSS.2014.X.039.

[60] J. Fu, I. Papusha, and U. Topcu. Sampling-based approximate optimal control under temporal logic constraints. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pages 227–235. ACM, 2017.

[61] A. Gaiser, J. Křetínskỳ, and J. Esparza. Rabinizer: Small deterministic automata for ltl (f, g). In *Automated Technology for Verification and Analysis*, pages 72–76. Springer, 2012.

[62] J. Garcıa and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[63] M. Ghavamzadeh, M. Petrik, and Y. Chow. Safe policy improvement by minimizing robust baseline regret. In *Advances in Neural Information Processing Systems*, pages 2298–2306, 2016.

[64] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, Mar. 2014.

[65] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2829–2838, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/gu16.html.

[66] M. Guo, K. H. Johansson, and D. V. Dimarogonas. Revising motion planning under linear temporal logic specifications in partially known workspaces. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5025–5032. IEEE, 2013.

[67] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org, 2017.

[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 *International Conference on Machine Learning*, pages 1856–1865, 2018.

[69] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan. Inverse reward design. In *Advances in Neural Information Processing Systems*, pages 6765–6774, 2017.

[70] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 395–412, 2019.

[71] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[72] M. Herman, T. Gindele, J. Wagner, F. Schmitt, and W. Burgard. Inverse reinforcement learning with simultaneous estimation of rewards and dynamics. In *Artificial Intelligence and Statistics*, pages 102–110, 2016.

[73] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[74] T. Homem-de Mello. A study on the cross-entropy method for rare-event probability estimation. *INFORMS Journal on Computing*, 19(3):381–394, 2007.

[75] J. Hu, M. C. Fu, S. I. Marcus, et al. A model reference adaptive search method for stochastic global optimization. *Communications in Information and Systems*, 8(3):245–276, 2008.

[76] J. Hu, P. Hu, and H. S. Chang. A stochastic approximation framework for a class of randomized optimization algorithms. *IEEE Transactions on Automatic Control*, 57(1):165–178, 2012.

[77] B.-Q. Huang, G.-Y. Cao, and M. Guo. Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. In *Proceedings of Conference on Machine Learning and Cybernetics*, volume 1, pages 85–89, 2005.

[78] C. Innocenti, H. Lindén, G. Panahandeh, L. Svensson, and N. Mohammadiha. Imitation learning for vision-based lane keeping assistance. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 425–430. IEEE, 2017.

[79] G. H. John. When the best move isn't optimal: Q-learning with exploration. In *AAAI*, page 1464. Citeseer, 1994.

[80] A. G. Joseph and S. Bhatnagar. Revisiting the cross entropy method with applications in stochastic global optimization and reinforcement learning. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands*, pages 1026–1034, 2016.

[81] S. Junges, N. Jansen, C. Dehnert, U. Topcu, and J.-P. Katoen. Safety-constrained reinforcement learning for mdps. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 130–146. Springer, 2016.

[82] M. Jurdziński. Small progress measures for solving parity games. In *Symposium on Theoretical Aspects of Computer Science 2000*, pages 290–301, 2000.

[83] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[84] R. E. Kalman. When is a linear control system optimal? *Journal of Basic Engineering*, 86(1): 51–60, 1964.

[85] H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng. Autonomous helicopter flight via rein-

forcement learning. In *Advances in neural information processing systems*, pages 799–806, 2004.

[86] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[87] T. Kipf, Y. Li, H. Dai, V. Zambaldi, E. Grefenstette, P. Kohli, and P. Battaglia. Compositional imitation learning: Explaining and executing one task at a time. *arXiv preprint arXiv:1812.01483*, 2018.

[88] E. Klein, M. Geist, B. Piot, and O. Pietquin. Inverse reinforcement learning through structured classification. In *Advances in Neural Information Processing Systems*, pages 1007–1015, 2012.

[89] J. Klein. Ltl2dstar - ltl to deterministic streett and rabin automata, 2015. URL `http://www.ltl2dstar.de/`.

[90] M. Kobilarov. Cross-entropy randomized motion planning. In *Robotics: Science and Systems*, 2011.

[91] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2619–2624. IEEE, 2004.

[92] T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause. Learning-based model predictive control for safe exploration. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 6059–6066. IEEE, 2018.

[93] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3116–3121. IEEE, 2007.

[94] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu. Correct, reactive, high-level robot control. *IEEE Robotics & Automation Magazine*, 18(3):65–74, 2011.

[95] J. Kretínskỳ, G. A. Pérez, and J.-F. Raskin. Learning-based mean-payoff optimization in an unknown mdp under omega-regular constraints. In *29th International Conference on Concurrency Theory (CONCUR 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[96] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard. Socially compliant mobile robot navigation via inverse reinforcement learning. *The International Journal of Robotics Research*, 35(11):1289–1307, 2016.

[97] S. Krishnan, A. Garg, S. Patil, C. Lea, G. Hager, P. Abbeel, and K. Goldberg. Unsupervised surgical task segmentation with milestone learning. In *Proc. Intl Symp. on Robotics Research (ISRR)*, 2015.

[98] M. Kuderer, S. Gulati, and W. Burgard. Learning driving styles for autonomous vehicles from

demonstration. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2641–2646. IEEE, 2015.

[99] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[100] M. Lai. Giraffe: Using deep reinforcement learning to play chess. *arXiv preprint arXiv:1509.01549*, 2015.

[101] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on Robot Learning*, pages 143–156, 2017.

[102] K. Lee, S. Choi, and S. Oh. Inverse reinforcement learning with leveraged gaussian processes. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3907–3912. IEEE, 2016.

[103] S. H. Lee, I. H. Suh, S. Calinon, and R. Johansson. Learning basis skills by autonomous segmentation of humanoid motion trajectories. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 112–119. IEEE, 2012.

[104] S. Levine, Z. Popovic, and V. Koltun. Feature construction for inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1342–1350, 2010.

[105] S. Levine, Z. Popovic, and V. Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27, 2011.

[106] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[107] L. Li and J. Fu. Sampling-based approximate optimal temporal logic planning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1328–1335. IEEE, 2017.

[108] Y. Li. Deep reinforcement learning. *CoRR*, abs/1810.06339, 2018. URL `http://arxiv.org/abs/1810.06339`.

[109] E. Liebman, M. Saar-Tsechansky, and P. Stone. Dj-mc: A reinforcement-learning agent for music playlist recommendation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 591–599. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[110] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, pages 157–163, 1994.

[111] M. L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.

[112] M. L. Littman and C. Szepesvári. A generalized reinforcement-learning model: Convergence and applications. In *Proceedings of Conference on Machine Learning*, pages 310–318, 1996.

[113] S. C. Livingston, E. M. Wolff, and R. M. Murray. Cross-entropy temporal logic motion planning. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 269–278. ACM, 2015.

[114] J. Macglashan and M. L. Littman. Between imitation and intention learning. In *International Conference on Artificial Intelligence*, pages 3692–3698, 2015.

[115] 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*, pages 1800–1809, 2018.

[116] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, 1984.

[117] S. Mannor, R. Rubinstein, and Y. Gat. The cross entropy method for fast policy search. In *In International Conference on Machine Learning*, pages 512–519. Morgan Kaufmann, 2003.

[118] T. Matsui, T. Goto, K. Izumi, and Y. Chen. Compound reinforcement learning: Theory and an application to finance. In *European Workshop on Reinforcement Learning*, pages 321–332. Springer, 2011.

[119] B. Michini and J. P. How. Bayesian nonparametric inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 148–163. Springer, 2012.

[120] B. Michini, T. J. Walsh, A.-A. Agha-Mohammadi, and J. P. How. Bayesian nonparametric reward learning from demonstration. *IEEE Transactions on Robotics*, 31(2):369–386, 2015.

[121] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[122] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[123] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[124] T. M. Moldovan and P. Abbeel. Safe exploration in markov decision processes. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1451–1458. Omnipress, 2012.

[125] W. Montgomery, A. Ajay, C. Finn, P. Abbeel, and S. Levine. Reset-free guided policy search: Efficient deep reinforcement learning with stochastic initial states. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3373–3380. IEEE, 2017.

[126] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006.

[127] G. Neu and C. Szepesvári. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 295–302. AUAI Press, 2007.

[128] Y. Nevmyvaka, Y. Feng, and M. Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680. ACM, 2006.

[129] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670. Morgan Kaufmann Publishers Inc., 2000.

[130] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5239–5246. IEEE, 2012.

[131] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157, 2015.

[132] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

[133] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[134] X. Pan and Y. Shen. Human-interactive subgoal supervision for efficient inverse reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1380–1387. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[135] I. Papusha, J. Fu, U. Topcu, and R. M. Murray. Automata theory meets approximate dynamic programming: Optimal control with temporal logic constraints. In *Decision and Control (CDC), 2016 IEEE 55th Conference on*, pages 434–440. IEEE, 2016.

[136] T. J. Perkins and A. G. Barto. Lyapunov-constrained action sets for reinforcement learning. In *Proceedings of Conference on Machine Learning*, volume 1, pages 409–416, 2001.

[137] M. Pirotta, M. Restelli, A. Pecorino, and D. Calandriello. Safe policy iteration. In *International Conference on Machine Learning*, pages 307–315, 2013.

[138] N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive (1) designs. In *Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

[139] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical computer science*, 13 (1):45–60, 1981.

[140] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Automata, Languages and Programming*, pages 652–671. Springer, 1989.

[141] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of Symposium on Principles of Programming Languages*, pages 179–190, 1989.

[142] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[143] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[144] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[145] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade. Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems*, pages 6550–6561, 2017.

[146] D. Ramachandran and E. Amir. Bayesian inverse reinforcement learning. *Urbana*, 51(61801): 1–4, 2007.

[147] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006.

[148] N. D. Ratliff, D. Silver, and J. A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

[149] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

[150] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[151] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. Learning monocular reactive uav control in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE, 2013.

[152] R. Y. Rubinstein and B. Melamed. *Modern simulation and modeling*, volume 7. Wiley New York, 1998.

[153] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

[154] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[155] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[156] A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of Conference on Machine Learning*, volume 93, pages 298–305, 1993.

[157] L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095, 1953.

[158] K. Shiarlis, M. Wulfmeier, S. Salter, S. Whiteson, and I. Posner. Taco: Learning task decomposition via temporal alignment for control. In *International Conference on Machine Learning*, pages 4661–4670, 2018.

[159] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395, 2014.

[160] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[161] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[162] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[163] S. Singh, R. L. Lewis, and A. G. Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pages 2601–2606. Cognitive Science Society, 2009.

[164] M. Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[165] S. L. Smith, J. Tumova, C. Belta, and D. Rus. Optimal path planning for surveillance with temporal logic constraints. *The International Journal of Robotics Research*, page 0278364911417911, 2011.

[166] S. Sohail and F. Somenzi. Safety first: a two-stage algorithm for the synthesis of reactive systems. *International Journal on Software Tools for Technology Transfer*, 15(5-6):433–454, 2013. doi: 10.1007/s10009-012-0224-3. URL `http://dx.doi.org/10.1007/s10009-012-0224-3`.

[167] A. L. Strehl, L. Li, and M. L. Littman. Reinforcement learning in finite mdps: Pac analysis. *The Journal of Machine Learning Research*, 10:2413–2444, 2009.

[168] S. Summers, D. M. Raimondo, C. N. Jones, J. Lygeros, and M. Morari. Fast explicit nonlinear model predictive control via multiresolution function approximation with guaranteed stability. *IFAC Proceedings Volumes*, 43(14):533–538, 2010.

[169] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[170] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[171] I. Szita and A. Lörincz. Learning tetris using the noisy cross-entropy method. *Learning*, 18 (12), 2006.

[172] N. Taghipour, A. Kardan, and S. S. Ghidary. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 113–120. ACM, 2007.

[173] G. Tesauro. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[174] W. Thomas, T. Wilke, et al. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

[175] M.-H. Tsai, Y.-K. Tsay, and Y.-S. Hwang. Goal for games, omega-automata, and logics. In *Computer Aided Verification*, pages 883–889. Springer, 2013.

[176] M. Turchetta, F. Berkenkamp, and A. Krause. Safe exploration in finite markov decision processes with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 4312–4320, 2016.

[177] E. Uchibe and K. Doya. Constrained reinforcement learning from intrinsic and extrinsic rewards. In *2007 IEEE 6th International Conference on Development and Learning*, pages 163–168. IEEE, 2007.

[178] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency*, pages 238–266. Springer, 1996.

[179] K. P. Wabersich and M. N. Zeilinger. Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning. *arXiv preprint arXiv:1812.05506*, 2018.

[180] A. Wachi, Y. Sui, Y. Yue, and M. Ono. Safe exploration and optimization of constrained mdps using gaussian processes. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[181] C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[182] M. Wen and U. Topcu. Probably approximately correct learning in stochastic games with temporal logic specifications. In *IJCAI*, pages 3630–3636, 2016.

[183] M. Wen and U. Topcu. Constrained cross-entropy method for safe reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 7450–7460, 2018.

[184] M. Wen, F. Memarian, and U. Topcu. Task-oriented deep inverse reinforcement learning. submitted.

[185] M. Wen, R. Ehlers, and U. Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4983–4990. IEEE, 2015.

[186] M. Wen, I. Papusha, and U. Topcu. Learning from demonstrations with high-level side information. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[187] M. Wigness, J. G. Rogers, and L. E. Navarro-Serment. Robot navigation from human demonstration: Learning control behaviors. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1150–1157. IEEE, 2018.

[188] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[189] E. Wolff, U. Topcu, and R. Murray. Optimal control with weighted average costs and temporal logic specifications. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012. doi: 10.15607/RSS.2012.VIII.057.

[190] E. M. Wolff, U. Topcu, and R. M. Murray. Optimal control with weighted average costs and temporal logic specifications. In *Robotics: Science and Systems*, 2012.

[191] E. M. Wolff, U. Topcu, and R. M. Murray. Efficient reactive controller synthesis for a fragment of linear temporal logic. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5033–5040. IEEE, 2013.

[192] M. Wulfmeier, P. Ondruska, and I. Posner. Deep inverse reinforcement learning. *CoRR, abs/1507.04888*, 2015.

[193] M. Wulfmeier, P. Ondruska, and I. Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[194] M. Wulfmeier, D. Z. Wang, and I. Posner. Watch this: Scalable cost-function learning for path planning in urban environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2089–2095. IEEE, 2016.

[195] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner. Large-scale cost function learning for path planning using deep inverse reinforcement learning. *The International Journal of Robotics Research*, 36(10):1073–1087, 2017.

[196] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 79–86. IEEE, 2017.

[197] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. *arXiv preprint arXiv:1608.05742*, 2016.

[198] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 528–535. IEEE, 2016.

[199] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li. Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, pages 167–176. International World Wide Web Conferences Steering Committee, 2018.

[200] Z. Zheng, J. Oh, and S. Singh. On learning intrinsic rewards for policy gradient methods. In *Advances in Neural Information Processing Systems*, pages 4644–4654, 2018.

[201] Z. Zhou, M. Bloem, and N. Bambos. Infinite time horizon maximum causal entropy inverse reinforcement learning. *IEEE Transactions on Automatic Control*, 63(9):2787–2802, 2018.

[202] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[203] W. Zielonka. Perfect-information stochastic parity games. In *International Conference on Foundations of Software Science and Computation Structures*, pages 499–513. Springer, 2004.