Optimal Control and Reinforcement Learning for Stochastic Systems under Temporal Logic Specifications

by

Lening Li

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Robotics Engineering

by

December 2022

APPROVED:

_____

Professor Jie Fu, Electrical & Computer Engineering, University of Florida, Advisor

_____

Professor Andrew Clark, Electrical & Systems Engineering, Washington University in St. Louis

_____

Professor Raghvendra V. Cowlagi, Aerospace Engineering, Worcester Polytechnic Institute

_____

Professor Carlo Pinciroli, Robotics Engineering, Worcester Polytechnic Institute

Abstract

This thesis aims to explore methods of near-optimal stochastic planning given high-level formal specifications. High-level formal specifications specify the properties that system behaviors should satisfy; optimal stochastic planning aims to maximize the system performance given criteria. With high-level formal specifications, stochastic optimal planning desires to synthesize a control policy to maximize the probability of system behavior satisfying these specifications. We commonly encounter such problems in defense operations, robotics, and other cyber-physical systems. However, algorithms requiring full system model knowledge or not scaling well cannot achieve optimal performance due to the unavailability or intractable size of system models. This thesis presents approximate algorithms that achieve near-optimal performance. The main contributions of this thesis are listed as follows. 1) We translate system specifications in probabilistic computation tree logic formulas to hard constraints and present a probabilistically complete approximate dynamic programming algorithm for near-optimal planning with multiple objectives; 2) We present a comprehensive optimal planning framework that leverages the topological information to accelerate policy learning; 3) To tackle continuous-state dynamic systems, we offer a variant of the actor-critic algorithm inspired by the proposed approximate dynamic programming algorithm; 4) We extend our framework into a two-player setting, where the agent exploits asymmetric information between players to synthesize a policy that outmaneuvers the opponent.

This thesis first develops an approximate dynamic programming method given soft performance criteria and hard constraints specified in a class of probabilistic computation tree logic formulas for stochastic systems. We model a stochastic system as a Markov decision process. Our approach consists of two steps: First, with suitably defined cost functions, we translate a class of probabilistic computation tree logic formulas into chance constraints enforced during planning. Second, we devise a probabilistically complete sampling-based method by integrating randomized optimization and dynamic programming with the softmax Bellman operator. The optimization iteratively solves for an upper bound while satisfying translated constraints. By adopting an on-policy sampling fashion, we achieve a tight error bound between the upper bound given by the approximation and the ground truth of the value function.

In the case of the Markov decision process with a high-level formal specification expressed by linear temporal logic formulas, probabilistic optimal planning remains challenging because of sparse rewards. This thesis then presents a policy learning framework guided by topological information encoded in formal specifications to

address this issue. First, we follow the standard procedure to translate the specification into a corresponding deterministic finite-state automaton. Then, we take the product between the Markov decision process and deterministic finite-state automaton to construct a product system. Our algorithm finds topological order that describes the structural information about deterministic finite-state automaton. This topological order allows us to propose a framework for updating values with optimality guarantees and accelerating policy learning. Further, this thesis utilizes our probabilistically complete approximate dynamic programming algorithm to efficiently learn a near-optimal value function and the associated policy.

Further, this thesis investigates the formal policy synthesis of continuous-state stochastic systems given high-level specifications in linear temporal logic. Since the system has continuous-state space, the product system has a hybrid product state space that worsens the reward sparsity issue. We present an actor-critic reinforcement learning algorithm where topological order is applicable. This algorithm employs advanced mathematical techniques, enjoying the property of hyperparameter self-tuning. We prove the optimality and convergence of our actor-critic algorithm. This work uses neural networks to approximate the value and policy functions as an alternative to storing intractable hybrid-state system models. While constructing a deterministic finite-state automaton, assigning integer numbers to automaton states can rank the approximated value or policy functions. We use modular learning to break the ordinal relationship by using an individual neural network for each automaton state's value function (policy).

Dynamic systems interacting with stochastic environments consider the case of symmetric information, and this thesis investigates beyond that. Additionally, we develop the optimal probabilistic planning of deception using a concurrent stochastic game with high-level formal specifications. There are two players: agent (player 1) and adversary (player 2); the adversary holds incomplete knowledge about the agent's task specification. During the adversarial interaction, the adversary infers the agent's intention from observation and takes actions to prevent the agent from accomplishing the task. By contrast, the agent exploits the incomplete information of its adversary to outmaneuver its adversary. This thesis introduces a class of hypergame models that capture the dynamic interaction between the player and the adversary in the presence of asymmetric, incomplete information. Further, this thesis establishes a solution concept for this class of hypergames. We design an online detection mechanism that alarms the agent with potential errors in modeling the adversary's behavior.

The thesis concludes with an overview and future research directions.

Acknowledgments

I want to express my most profound appreciation for the support from my advisor, Dr. Jie Fu. Her enthusiasm, patient guidance, inspirational ideas, and ability have made my Ph.D. study at WPI a rewarding and productive journey. Throughout my doctoral program, she taught me how to think about research problems, helped me develop research skills, and showed me the importance of academic writing. Although I did suffer initially, I enjoyed my evident improvement during the later years. Her profound influence is present in every stage of my Ph.D. journey and will be in the future I write. Additionally, I want to thank my committee members, Dr. Andrew Clark, Dr. Raghvendra V. Cowlagi, and Dr. Carlo Pinciroli, for their valuable advice regarding my research and dissertation.

I want to thank all the Control and Intelligent Robotics Laboratory members. Among all members, I am thankful to Abhishek N. Kulkarni and Haoxiang Ma. It was an absolute pleasure to collaborate with Abhishek, and from him, I learned his optimistic life attitude and rigorous research attitude. Researching with Haoxiang is truly a wonderful and "scary" experience: his ability to point out the questionable results made my research full of "challenges." Further, I am fortunate to collaborate with Dr. Mitchell Colby from SSCI. Additionally, I want to express my gratitude to the funding sources: Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00111990015.

I feel blessed to meet every member of Team WPI-CMU, especially Matt DeDonato, Felipe Polido, and Dr. Michael A. Gennert. Specifically, I am indebted to Dr. Michael A. Gennert for offering me an opportunity to start my robotics research at WPI. The DRC was a tremendous experience not only because I made so many good friends but, more importantly, this experience determined my Ph.D. study. I thank all my friends who made my stay at WPI such a wonderful experience. Among all the people, I am fortunate to meet Vinayak Jagtap, Rahul Krishnan, Jayam Patel, Nandan Banerjee, and Zhiming Hong, who took care of me as their younger brother. I want to thank my middle school teacher, Feiyun Bao, who did not give up on me and believed in me. I am grateful to them in too many ways to list.

Finally, I want to thank my parent, Guang Li and Minli Lu, and my significant other, Rachel Alexandra Carter, for their endless source of love, affection, support, understanding, and motivation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Motivation

We commonly encounter stochastic dynamic systems in various applications, such as defense operations [5], robotics [1, 4, 82, 100], automatic control [42, 74, 131], economics [23, 87], manufacturing [79, 132], and other cyber-physical systems [91]. In each application, stochastic systems usually need to fulfill beyond a reachability objective, that is, to reach a target region, but with high-level specifications that describe specific requirements. The interaction between stochastic systems and environments can be captured by a rigorous mathematical framework, Markov decision process (MDP) [101] since its introduction by Bellman in 1957 [11], for modeling sequential decision-making problems. High-level formal specifications specify the properties that system behaviors should satisfy, including 1) liveness (something good will always eventually happen), 2) safety (nothing bad will happen), 3) and fairness (all constituent processes will be involved, and none of them will starve) [8]. For instance, defense operations utilize computation-limited autonomous drones for surveillance tasks. A typical surveillance task demands a drone to visit certain areas of interest in a predefined temporal order (liveness) while avoiding entering no-fly zones (safety) [83].

Previous work [121] synthesizes policies for the liveness property by specifying the arrival times of the drone for each area of interest. However, if there is redundancy in the solution: multiple trajectories satisfy the required liveness property, then defining arrival times for each region of interest in each trajectory becomes tedious. Expressing the temporal order of visiting regions of interest in a compact form becomes critical when designing optimal control algorithms. Furthermore, only specifying arrival time leaves

how to enforce the safety property open; that is, how to prevent the drone from entering no-fly zones.

Fortunately, *temporal logic* allows us to represent such high-level specifications rigorously and succinctly. It enables us to reason about the order of visiting areas of interest without explicitly introducing time. Besides, temporal logic permits reasoning more properties. For instance, if a drone has fuel or battery shortage, then the drone needs to retreat immediately. Although temporal logic formulas can specify numerous system behaviors, incorporating these into stochastic optimal planning remains difficult. Stochastic optimal planning aims to maximize the system performance given criteria. With high-level formal specifications, stochastic optimal planning desires to synthesize a control policy to maximize the probability of system behavior satisfying these specifications. Researchers did not start paying extensive attention and fully exploiting the potential of temporal logic until Duret *et al.* [40] presented a systematic translation that allows us to translate temporal logic formulas into well-studied graph-based models. Formal policy synthesis allows controlling a stochastic system to ensure desirable system performance with provably correct guarantees given high-level specifications in which researchers have expressed substantial interest [24, 29, 94, 130]. Despite the theoretical and experimental successes, policy synthesis with translated models remains challenging due to the following questions this thesis motivates to explore:

1. Given a stochastic system and its high-level specification, how do we *efficiently* learn *optimal* control policies for satisfying the given specification?

2. When it is hard to sample a trajectory that satisfies the specification, leading to a *sparse reward* issue, how can we alleviate this issue?

3. How do we learn a policy when there is *incomplete* knowledge about other stochastic systems' temporal objectives? What if the system has *continuous* state space?

4. If an adversary has *incomplete, asymmetrical* information in a two-player setting, how do we leverage this strategic information to maximize the satisfaction probability of given high-level specifications?

Thesis Contributions and Outline

This thesis bridges dynamic programming, formal verification and synthesis, and reinforcement learning to propose a coherent probabilistic planning framework for stochastic

systems under temporal objectives achieving near-optimal performance. Such a framework can reason the system's probabilistic and temporal properties. To further improve the policy learning's efficiency, we incorporate the structural information of the model to obtain a total order to guide the computation. Additionally, we embrace neural networks to deal with the continuity of the state space and present a novel model-free reinforcement learning (RL) algorithm suitable for applying graphic structural information. We aspire to extend this framework to the multi-player setting, where we leverage deception to exploit the adversary's unawareness to maximize the probability of completing our objectives.

Our work explores formal policy synthesis with a class of high-level specifications expressed in probabilistic computation tree logic (PCTL) [28] and linear temporal logic (LTL) [124] to guarantee stochastic systems' probabilistic and temporal properties. We now present the thesis's organization and each chapter's primary results.

Chapter 2 presents formal definitions of linear temporal logic, deterministic finite-state automaton, and Markov decision process.

Chapter 3 presents a probabilistically complete sampling-based approximate dynamic programming (ADP) algorithm to learn a near-optimal value function for temporal objectives in PCTL.

- We introduce a systematic way to translate a class of PCTL formulas into chance constraints with properly defined cost functions.
- We formulate a constrained optimization problem that incorporates constraints derived from the softmax Bellman operator in dynamic programming and PCTL formulas.
- We design a probabilistically complete sampling-based ADP algorithm for such a constrained problem. The method is scalable, technically sound, and can synthesize a near-optimal policy that satisfies temporal constraints in PCTL formulas.

Chapter 4 leverages the topological information encoded in the high-level specification in LTL to accelerate value function learning.

- We offer an algorithm to reveal the encoded topological information in temporal objectives in LTL formulas to guide the optimal value function learning process.
- We guarantee the optimality of learned value functions guided by the topological order.

- We apply the discovered topological information to the proposed ADP algorithm to build an efficient formal policy synthesis framework.

Chapter 5 proposes a model-free actor-critic algorithm to tackle continuous-state stochastic systems.

- We devise a model-free actor-critic algorithm inspired by the ADP algorithm for optimal planning with high-level specifications when the system models are unavailable.
- We leverage neural networks to approximate value and policy function to tackle continuous-state space.
- We adopt modular learning to break the ordinal relation between values in different task automaton states to improve the performance of the policy synthesis framework.

Chapter 6 analyzes the case of planning in adversarial interaction captured by a class of hypergame models in the presence of asymmetric, incomplete information.

- We establish a solution concept for a class of hypergames, where these hypergames enable probabilistic optimal planning in an adversarial interaction in the presence of asymmetric, incomplete information.
- We show the effectiveness of the obtained subjectively rationalizable strategy in deception.
- We propose an online detection mechanism to alert us when we model an adversary incorrectly.

Chapter 7 concludes the dissertation and discusses directions for future work.

This thesis opened up a vast research area of efficient formal policy thesis for stochastic dynamic systems with temporal objectives in single- and double-player views. It allows for generating policy when the system models are continuous and unknown with complex temporal objectives. Additionally, we consider an adversarial interaction when there is asymmetrical information. This research has wide-range potential usage in robotics, aerospace, economics, traffic control, networking, and other cyber-physical systems.

Chapter 2

Preliminaries

Notations We let $\mathbf{R}, \mathbf{R}_+, \mathbf{R}_{\geq 0}, \mathbf{N}$ denote the set of real, positive, non-negative, and natural numbers, respectively. For a finite set X, $|X|$ denotes the size of X, and 2^X denotes its power set. Let $\Delta(X)$ denote the probability simplex in $\mathbf{R}^{|X|}$. Given a distribution $\mu \in \Delta(X)$, $\mathsf{Supp}(\mu) = \{x \in X \mid \mu(x) \neq 0\}$ is the *support* of μ. We denote $Z \sim \mu$ to specify the random variable Z has a distribution μ. We denote expectation as $\mathbf{E}$, where $\mathbf{E}_{X \sim \mathbf{Pr}}[f(X)]$ is the expected summation of the $f(X)$, where random variable X is conditional on the distribution $\mathbf{Pr}$. We use the notation Σ to denote a finite set of symbols known as the *alphabet*. The indicator function is denoted by $\mathbf{1}$, where $\mathbf{1}(E)$ evaluates to be 1 if the event E is true and 0 otherwise.

In this chapter, we review the necessary background of the LTL and the labeled MDP used in the following chapters.

Linear Temporal Logic

We use *linear temporal logic* (LTL) to describe a complex high-level task. The basic components of LTL formulas are tautology true and falsum false, atomic propositions ($p \in \mathcal{AP}$ is an atomic proposition), the Boolean connector like conjunction $\wedge$ and negation $\neg$, and two basic temporal modalities $\bigcirc$ (read "next") and U (read "until"). The automatic proposition $p \in \mathcal{AP}$ stands for the state label $p \in \mathcal{AP}$ in a transition system. The operator $\bigcirc$ is a unary prefix operator and leads a single LTL formula φ. The formula $\bigcirc \varphi$ holds at the current moment if the formula φ is true at the next step. The operator U is a binary infix operator and stands between two LTL formulas φ_1 and φ_2. The formula $\varphi_1 \,\mathsf{U}\, \varphi_2$ holds at the current moment if φ_2 becomes true in some future moment, and before that,

φ_1 holds for every time step until the future moment.

Definition 1 (Syntax of LTL)**.**
An LTL formula is defined over the set $\mathcal{AP}$ of automatic propositions according to the following grammar:

$$\varphi \coloneqq \mathsf{true} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2,$$

where $p \in \mathcal{AP}$. The operators $\Diamond$ (read as eventually) and $\Box$ (read as always) are defined using the operator U as follows: $\Diamond \varphi = \mathsf{true}\,\mathsf{U}\,\varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$.

LTL formulas describe the properties of sequences of symbols. Each finite (resp. infinite) sequence of symbols, termed a finite (resp. ω-regular) word, has the following form: $w = \sigma_0\sigma_1\cdots\sigma_n$ with $\sigma_i \in \Sigma$ for any $0 \leq i \leq n$ (resp. $w = \sigma_0\sigma_1\cdots$ with $\sigma_i \in \Sigma$ for any $0 \leq i$). We use Σ^* and Σ^ω to denote the set of all finite words and the set of all ω-regular words, respectively. We denote the length of a word w as $|w|$. The *empty word*, denoted by ε, is the empty sequence Σ^0, and $|\varepsilon| = 0$. We define the set of nonempty finite words $\Sigma^+ = \Sigma^* \setminus \varepsilon$. The semantics of an LTL formula φ is defined as a language $\mathsf{Words}(\varphi)$ that contains all ω-regular words over the alphabet Σ that satisfy φ.

Definition 2 (Semantics of LTL (Interpretation over Words))**.**
Let φ be an LTL formula over $\mathcal{AP}$. The language induced by the formula φ is

$$\mathsf{Words}(\varphi) = \{w \in \Sigma^\omega \mid w \models \varphi\}$$

where the satisfaction relation $\models \subseteq \Sigma^\omega \times \text{LTL}$ is defined as the following:

$$\begin{aligned}
&w \models \mathsf{true}\\
&w \models p && \iff p \in \sigma_0 (\textit{ i.e., } \sigma_0 \models p),\\
&w \models \varphi_1 \wedge \varphi_2 && \iff w \models \varphi_1 \wedge w \models \varphi_2,\\
&w \models \neg\varphi && \iff w \not\models \varphi,\\
&w \models \bigcirc \varphi && \iff \sigma_1\sigma_2\sigma_3\cdots \models \varphi,\\
&w \models \varphi_1 \,\mathsf{U}\, \varphi_2 && \iff \exists j \geq 0, \sigma_j\sigma_{j+1}\cdots \models \varphi_2, \text{ and } \sigma_i\sigma_{i+1}\cdots \models \varphi_1, \text{ for all } 0 \leq i < j.
\end{aligned}$$

Deterministic Finite-state Automaton

A syntactically co-safe LTL (scLTL) formula [67], a subclass of LTL formulas, contains only temporal operators $\Diamond$, $\bigcirc$, and U when written in a positive normal form [8] (*i.e.,*

the negation operator $\neg$ appears only in front of atomic propositions). The unique property of scLTL formulas is that a word satisfying an scLTL formula φ only needs a *good prefix*, *i.e.,* given a good prefix $w \in \Sigma^*$, the word $ww' \models \varphi$ satisfies the scLTL formula φ for any $w' \in \Sigma^\omega$. The set of good prefixes can be compactly represented as the language accepted by a *deterministic finite-state automaton* (DFA) defined as follows.

Definition 3 (Deterministic Finite-state Automaton (DFA))**.**
A deterministic finite-state automaton (DFA) of an scLTL formula φ is a tuple

$$\mathcal{A}_\varphi = \langle Q, \Sigma, \delta, \iota, F \rangle,$$

where the components of $\mathcal{A}$ are defined as follows:

- a finite set Q of states;
- a finite set $\Sigma = 2^{\mathcal{AP}}$ of symbols;
- a deterministic function $\delta \colon Q \times \Sigma \to Q$;
- a unique initial state ι;
- a set F of accepting states.

When the context is clear, we write $\mathcal{A}_\varphi$ as $\mathcal{A}$. For a finite word $w = \sigma_0\sigma_1 \cdots \sigma_n \in \Sigma^*$, the DFA generates a sequence of states $q_0q_1 \cdots q_{n+1}$ such that $q_0 = \iota$ and $q_{i+1} = \delta(q_i, \sigma_i)$ for any $0 \leq i \leq n$. The word w is accepted by the DFA if and only if $q_{n+1} \in F$. The set of words accepted by the DFA $\mathcal{A}$ is called *its language*. Given an objective expressed as an scLTL formula φ, the set of good prefixes of words corresponding to φ is accepted by a DFA, which has a special property that all final states are sink states. If a finite prefix of an infinite run reaches a final state, it is ensured that the "last" state will be a final state, and the word corresponding to this run is accepted.

We assume that the DFA is complete — that is, for every state-action pair $(q, \sigma) \in Q \times \Sigma$, $\delta(q, \sigma)$ is defined. An incomplete DFA can be made complete by adding a sink state q_{sink} such that $\forall \sigma \in \Sigma, \delta(q_{\mathsf{sink}}, \sigma) = q_{\mathsf{sink}}$, and directing all undefined transitions to the sink state q_{sink}.

Example 1. A *sequential visiting* task requires a car to avoid obstacles and accomplish one of the following: (a) visit A and do not visit D or obstacles until C is visited; (b) visit D and do not visit A or obstacles until B is visited. To describe this task, we define a set $\mathcal{AP}$ of atomic propositions as follows:

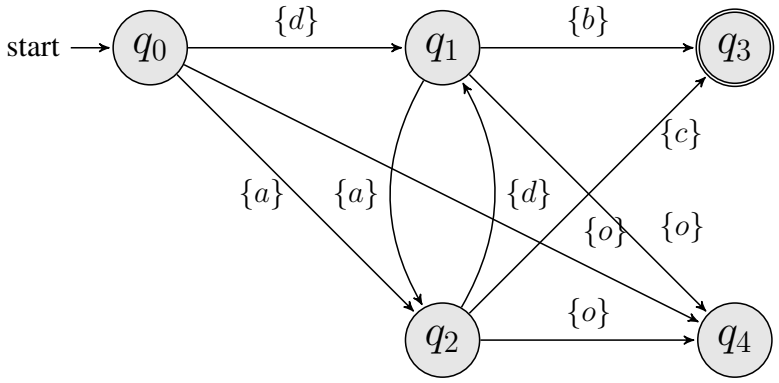


Figure 2.1: The DFA accepting the formula $\varphi = \neg o \,\mathsf{U}\, (\varphi_1 \vee \varphi_2)$, where $\varphi_1 = a \wedge ((\neg d \wedge \neg o) \,\mathsf{U}\, c)$, and $\varphi_2 = d \wedge ((\neg a \wedge \neg o) \,\mathsf{U}\, b)$.

- a: car reaches A.
- b: car reaches B.
- c: car reaches C.
- d: car reaches D.
- o: car reaches obstacles.

Given the set $\mathcal{AP}$ of atomic propositions, we can capture the sequential visiting task by an scLTL formula as follows:

$$\varphi = \neg O \,\mathsf{U}\, (\varphi_1 \vee \varphi_2),$$

where

$$\varphi_1 = a \wedge ((\neg d \wedge \neg o) \,\mathsf{U}\, c),$$
$$\varphi_2 = d \wedge ((\neg a \wedge \neg o) \,\mathsf{U}\, b),$$

where the corresponding DFA is depicted in Figure 2.1; for clarity, we trim self-loops in Figure 2.1. That is, for state q_0, we remove transitions from q_0 to q_0 via symbol σ, where $\sigma \in 2^{\mathcal{AP}} \setminus (\{a\} \cup \{d\})$. We similarly remove transitions for states q_1, q_2, q_3, and q_4.

Labeled Markov decision process

We use a *Markov decision process* to model a stochastic system. Given a labeling function L, a labeled MDP is defined as follows.

Definition 4 (Labeled Markov Decision Process (MDP) [84])**.**
A *labeled Markov decision process* is a tuple

$$\mathbf{M} = \langle S, A, P, s_0, L, \mathcal{AP} \rangle,$$

where the components of $\mathbf{M}$ are defined as follows:

- S is a set of states.
- A is a set of actions.
- $P\colon S \times A \times S \to [0, 1]$ is the transition probability function, where $P(\cdot \mid s, a)$ represents the probability distribution over the next states given an action $a \in A$ taken at the current state $s \in S$.
- $\mu_0 \in \Delta(S)$ is the initial state distribution.
- $\mathcal{AP}$ is a set of atomic propositions.
- $L\colon S \to 2^{\mathcal{AP}}$ is the labeling function that maps a state $s \in S$ to a subset of propositions $L(s) \subseteq \mathcal{AP}$ that hold true at state $s \in S$.

For $s \in S$, we denote $A(s) = \{a \in A \mid \exists s' \in S, P(s' \mid s, a) > 0\}$ the set of *admissible* actions at state $s \in S$. If the initial state distribution μ_0 degenerates to a Dirac delta function, then the MDP starts at a unique initial state, denoted by s_0. For any $s \in S$, we denote $A(s) = \{a \in A \mid \exists s' \in S, P(s' \mid s, a) > 0\}$ as the set of *admissible* actions at state s.

Optimal planning with MDP aims to learn a policy. We consider the following policies:

- A finite-memory, deterministic policy $\pi\colon S^* \to A$ maps a history of state sequence to an action;
- A finite-memory, stochastic policy $\pi\colon S^* \times A \to [0, 1]$ maps a history of state sequence to a distribution over actions.
- A Markov, deterministic policy $\pi\colon S \to A$ maps the current state to an action;
- A Markov, randomized policy $\pi\colon S \times A \to [0, 1]$ maps the current state into a distribution over actions.

We denote the set of policies as Π. Given an MDP $\mathbf{M}$ and a policy π, the policy induces a Markov chain $\mathbf{M}^\pi = S_0 A_0 S_1 A_1 S_2 \cdots$, where S_i, A_i are the random variables describing the $i+1$-th state and action in the chain. We omit the actions and refer to the π-induced Markov chain by $\mathbf{M}^\pi = \{S_n \mid n \geq 0\}$. A Markov chain is a set of state-action sequences termed *paths*. An infinite (resp. finite) path $\rho^\pi = s_0 a_0 s_1 a_1 \cdots \in (S \times A)^\omega$ (resp. $s_0 a_0 s_1 a_1 \cdots s_N \in (S \times A)^* S$) conditional on the policy π being followed satisfies: for all $t \geq 0$, we have $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, $a_t \sim \pi(\cdot \mid s_t)$, and $s_0 \sim \mu_0$. We say a state $s \in S$ is a *sink* or absorbing state if $P(s \mid s, a) = 1$ for all $a \in A$. We denote the set of all paths as PATH, the set of paths following policy π as PATH^π, and the set of paths starting at state $s \in S$ as $\mathrm{PATH}(s)$. We use $\mu^\pi \in \Delta(\mathrm{PATH}^\pi)$ to denote the trajectory distribution induced by the policy π and $\mu^\pi(s) \in \Delta(\mathrm{PATH}^\pi(s))$ to denote the trajectory distribution induced by the policy π starting at state $s \in S$. More specifically, we use $\mu^\pi(\rho)$ and $\mu^\pi(\rho; s)$ to denote the probability of the path ρ in the set of π-induced paths and the probability of the path ρ in the set of π-induced paths starting at s, respectively.

Given a finite path $\rho = s_0 a_0 s_1 a_1 \cdots s_N \in \mathrm{PATH}$, we obtain a sequence of labels $L(\rho) = L(s_0) L(s_1) \cdots L(s_N) \in \Sigma^*$. A finite path ρ satisfies the formula φ, denoted by $\rho \models \varphi$, if and only if the corresponding DFA accepts $L(\rho)$.

Example 2. We have a reachability objective that is to reach a state s_2 eventually. Slightly abusing notation, we define a set of atomic proposition $\mathcal{AP} = \{s_2\}$, where the atomic proposition $s_2 \in \mathcal{AP}$ means that system visits state s_2. Given the defined set $\mathcal{AP}$, we model the system as a labeled MDP $\mathbf{M} = \langle S = \{s_0, s_1, s_2\}, A = \{a_1\}, P, s_0, L, \mathcal{AP} = \{s_2\} \rangle$, where labeling function L is defined as follows: $L(s_0) = \emptyset$, $L(s_1) = \emptyset$, and $L(s_2) = \{s_2\}$, and transition probability function P is visualized in Figure 2.2.

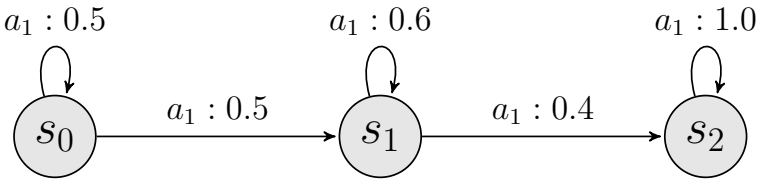


Figure 2.2: A MDP example.

Given such a reachability objective, RL uses a reward function to give feedback to the system, *i.e.,* a positive reward is only given when the system reaches the goal. The reward function $R\colon S \times A \to \mathbf{R}$ maps the current state and action into a real value, where $R(s, a)$ is the reward for executing action $a \in A$ at state $s \in S$.

Standard RL is to maximize the expected rewards. We term the excepted sum of rewards as value function $\mathcal{V}$. A value function $\mathcal{V}$ starting from an initial state distribution μ_0 following policy π is defined as follows:

$$\mathcal{V}^{\pi}(\mu_0) = \mathbf{E}_{\rho\sim\mu^{\pi}}[\sum_{t=0}^{\infty}\gamma^t R(S_t = s_t, A_t = a_t) \mid s_0 \sim \mu_0] \tag{2.1}$$

where $\rho = s_0 a_0 s_1 a_1 \cdots s_t a_t \cdots \in \mathrm{PATH}^{\pi}(s)$, and $\gamma \in [0, 1)$ is a discount factor.

The goal of the optimal planning problem is to learn an optimal policy that maximizes that expected sum of rewards. The optimal policy π^* is achieved only if: for all $s \in S$,

$$\pi^*(s) = \operatorname*{argmax}_{\pi\in\Pi} \mathcal{V}^{\pi}(s). \tag{2.2}$$

We recall the optimal bellman operator $\mathcal{B}$ as follows:

$$\mathcal{B}\,\mathcal{V}(s) = \max_{a\in A}\left(R(s,a) + \gamma\,\mathbf{E}_{s'\sim P(\cdot|s,a)}[\mathcal{V}(s')]\right),$$

and bellman operator $\mathcal{B}^{\pi}$ *w.r.t.* the policy π as follows:

$$\mathcal{B}^{\pi}\,\mathcal{V}(s) = \pi(a \mid s)\left(R(s,a) + \gamma\,\mathbf{E}_{s'\sim P(\cdot|s,a)}[\mathcal{V}(s')]\right)$$

It is well known that the Bellman operator $\mathcal{B}$ is monotonic [116]. From this and the fact that $\mathcal{B}$ is a contraction, there is a fixed point $\mathcal{V}^*(s) = \mathcal{B}\,\mathcal{V}^*(s)$, for all $s \in S$, which follows that for any $s \in S$, due to $\mathcal{V}(s) \geq \mathcal{B}\,\mathcal{V}(s)$, we have

$$\mathcal{V}(s) \geq \mathcal{B}\,\mathcal{V}(s) \geq \mathcal{B}^2\,\mathcal{V}(s) \geq \mathcal{B}^3\,\mathcal{V}(s) \geq \cdots \geq \mathcal{V}^*(s).$$

Most methods leverage monotonic and contractive properties of operator $\mathcal{B}$. We recall two classic methods briefly: (a) linear programming (LP); (b) value iteration. Authors [34] show that the feasible solution to the problem (2.3) is the solution for the optimal planning problem with the given MDP.

$$\begin{aligned} \min_{\mathcal{V}} \quad & \sum_{s\in S} c(s)\,\mathcal{V}(s) \\ \text{subject to:} \quad & \mathcal{V}(s) \geq \max_{a\in A}\left(R(s,a) + \gamma\,\mathbf{E}_{s'\sim P(\cdot|s,a)}[\mathcal{V}(s')]\right), \text{ for all } s \in S \end{aligned} \tag{2.3}$$

where $\mathcal{V} = [\mathcal{V}(s)]_{s\in S}$ be a vector of variables, one for each state, and $c = [c(s)]_{s\in S}$ is a vector of non-negative state-relevance weights, *i.e.,* $c(s) \geq 0$, for all $s \in S$. However, the

problem (2.3) is not linear because $\max$ operators are included in constraints. We can convert such a problem to an equivalent LP problem:

$$\begin{aligned} \min_{\mathcal{V}} \quad & \sum_{s \in S} c(s)\, \mathcal{V}(s) \\ \text{subject to:} \quad & \mathcal{V}(s) \geq R(s,a) + \gamma\, \mathbf{E}_{s' \sim P(\cdot|s,a)}[\mathcal{V}(s')], \text{ for all } (s,a) \in S \times A \end{aligned} \tag{2.4}$$

In the exact solution using LP, as long as $c(s)$ is positive, for all $s \in S$, the value function obtained by solving the LP is optimal. Once $\mathcal{V}$ is obtained, a deterministic, optimal policy can be generated using the Bellman equation,

$$\pi(s) = \operatorname*{argmax}_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s, a)\, \mathcal{V}(s') \right).$$

Besides the optimization-based solution, value iteration approaches the optimal planning for MDP from a different perspective. Value iteration constantly applies bellman operator $\mathcal{B}$ for every state until the values converge. We recall the value iteration algorithm in Algorithm 1.

Algorithm 1: Value Iteration Algorithm

Input: ϵ: a small positive number
Output: π: a deterministic policy *s.t.* $\pi \approx \pi^*$
Initialization: Initialize $\mathcal{V}(s)$ arbitrarily, *e.g.,* $\mathcal{V}(s) = 0$, for all $s \in S$

$\Delta \leftarrow 0$
while $\Delta \geq \epsilon$ **do**
  for each $s \in S$ **do**
    $v \leftarrow \mathcal{V}(s)$
    $\mathcal{V}(s) \leftarrow \max_{a \in A} \left(R(s,a) + \gamma\, \mathbf{E}_{s' \sim P(\cdot|s,a)}[\mathcal{V}(s')] \right)$
    $\Delta \leftarrow \max(\Delta, |v - \mathcal{V}(s)|)$
  end for
end while
return π *s.t.* $\pi(s) = \operatorname{argmax}_{a \in A} \left(R(s,a) + \gamma\, \mathbf{E}_{s' \sim P(\cdot|s,a)}[\mathcal{V}(s')] \right)$

Chapter 3

Approximate Dynamic Programming with Probabilistic Computation Tree Logic Constraints

3.1 Overview

For safety-critical dynamic systems, one primary control objective is to ensure desirable system performance with provable correctness guarantees given high-level system specifications. In this chapter, we investigate the following problem: Given a stochastic system modeled as an MDP, how can *efficiently* synthesize an optimal policy regarding a performance criterion while satisfying safety-critical and mission-critical constraints expressed in temporal logic? This problem considers multiple objectives, including soft constraints that maximize the total reward and hard constraints that satisfy safety properties [37, 65].

In this chapter, we develop an ADP algorithm, a class of ADP methods, in MDP given both soft performance criteria and hard temporal logic constraints. The hard constraints are given by PCTL [68] formulas used to reason about probabilistic properties in stochastic systems. For instance, a PCTL formula can specify that the probability is greater than 0.85 that the goal can be reached with a cost less than 100. Our approach includes two steps. In the first step, we show that a large subclass of PCTL formulas can be equivalently represented by chance constraints over the path distribution in a stochastic system by following the current policy with appropriately defined cost functions. We introduce the mixing time for the Markov chain to approximately verify probabilistic properties in time-unbounded PCTL using trajectories with finite lengths. It is noted that

PCTL involves global properties in a system, which is often hard to enforce using locally optimal policy search methods. In the second step, we develop a chance-constrained ADP method to solve the planning problem with reward maximization and PCTL constraint satisfaction. In literature, a chance-constrained approximate policy iteration method has been developed [27]. We consider ADP methods to obtain guarantees for global properties in PCTL formulas. This is achieved via integrating randomized optimization with the approximate LP formulation proposed in [34].

Our sampling-based ADP method has the following desirable properties:

- It achieves a tight error bound by weighing the approximation errors over the state space using the state visitation frequencies of an approximately optimal policy.
- It is probabilistic complete. It converges to an approximately optimal policy that satisfies the PCTL constraints with probability one.

The rest of the chapter is structured as follows. Section 3.3 provides necessary preliminaries on PCTL formulas. Section 3.4 contains the main results of this chapter, including the translation from a subclass of PCTL to chance constraints in MDP and the ADP algorithm. Presented in section 3.5 are case studies with robotic motion planning examples to validate the optimality and correctness of the proposed method. We conclude in Section 3.6.

3.2 Related Work

Most existing methods assume that the system's current state is known precisely. However, noisy sensors and actuators can invalidate such an assumption, resulting in guaranteeing systems' probabilistic properties difficult. Lahijanian *et al.* [71, 72] considered task specifications in PCTL formulas, where the learned policy could provide probabilistic guarantees against system noise. Specifically, in 2010, Lahijanian *et al.* [72] investigated an automatic deployment of a robot, where this deployment needs to satisfy a specification over a set of properties of interest given an environment. They presented a framework that first modeled the robot's motion as an MDP, then translated the deployment problem into a policy synthesis problem to maximize the probability of satisfying a PCTL formula. However, this paper only focused on a small fragment of PCTL formulas. Later, Lahijanian *et al.* [71] extended their previous work to the full range of PCTL formulas. Embracing the full range of the PCTL formulas could improve expressiveness. In this framework, they leveraged

some existing PCTL model-checking algorithms [102] as sub-algorithms corresponding to each temporal operator as building blocks for constructing a policy from a formula with multiple operators. Despite successful experimental results, their methods heavily rely on abstraction methods, which requires prior knowledge or expertise. Without abstraction methods, the existing graph-based model-checking algorithms would fail on continuous systems.

3.3 Preliminaries: Probabilistic Computation Tree Logic with Reachability Reward/Cost Properties

PCTL provides syntax and semantics to quantify probabilistic properties [43]. Recall that $\mathcal{AP}$ is the set of atomic propositions. The syntax of PCTL with reachability reward/cost properties [68] is defined as follows:

$$\phi \;\coloneqq\; \mathsf{true} \;\Big|\; \alpha \;\Big|\; \phi_1 \wedge \phi_2 \;\Big|\; \neg\phi \;\Big|\; P_{\bowtie p}[\psi] \;\Big|\; C_{\bowtie m}(\Diamond^{\leq k}\phi)$$

$$\psi \;\coloneqq\; \bigcirc \phi \;\Big|\; \phi \,\mathsf{U}^{\leq k}\, \phi \;\Big|\; \phi_1 \,\mathsf{U}\, \phi_2$$

where ϕ is a state formula, and ψ is a path formula. $\alpha \in \mathcal{AP}$ is an atomic proposition. $k \in \mathbf{N}$ is a non-negative integer. $\bowtie \in \{\leq, \geq, >, <\}$, $p \in [0, 1]$ is a scalar variable. $m \in \mathbf{R}$ is a threshold for cost.

A path formula is interpreted on paths and is composed of state formulas and temporal operators "Next" ($\bigcirc$) and "Until" (U). $\mathsf{U}^{\leq k}$ is the *bounded until* operator. Recall that $\Diamond \phi = \mathsf{true}\,\mathsf{U}\,\phi$ is "eventually," and $\bigcirc \phi$ asserts that the next state satisfies a state formula ϕ. The formula $\phi_1 \,\mathsf{U}^{\leq k} \phi_2$ asserts that ϕ_2 is satisfied within k steps, and all preceding states satisfy ϕ_1. The formula $\phi_1 \,\mathsf{U}\, \phi_2$ asserts that ϕ_2 is satisfied sometime in the future, and all preceding states satisfy ϕ_1. The formula $\Diamond^{\leq k}\phi$ means ϕ becomes true in no more than k steps.

Similarly, a path in an MDP is related to a PCTL formula through the labeling function $L\colon S \to 2^{\mathcal{AP}}$. The labeling function L assigns each state $s \in S$ with a set of atomic propositions $L(s) \subseteq \mathcal{AP}$ that are valid at the state s. The labeling function L can be extended to paths in the usual way, *i.e.,* $L(\rho_1\rho_2) = L(\rho_1)L(\rho_2)$, for any $\rho_1, \rho_2 \in (S \times A)^*$.

A state formula is evaluated on states. A state s that satisfies a state formula ϕ is denoted by $s \models \phi$. The statement $s \models P_{\bowtie p}[\psi]$ means that the probability of a path starting

from s that satisfies path formula ψ is $\bowtie p$, for any $\bowtie \in \{\leq, \geq, >, <\}$. The reachability reward formula [68] $s \models C_{\bowtie m}[\Diamond^{\leq k}\phi]$ means that the total accumulated rewards/costs along a path of length no greater than k that reaches a state satisfying ϕ are $\bowtie m$ for a predefined reward/cost function. Note that we consider a PCTL formula is always a state formula, and path formulas only appear in the composition of a state formula.

Example 3. For ease of understanding, we give several examples of PCTL formulas.

- $P_{\geq 0.95}(\text{true}\,\mathsf{U}^{\leq 10}\text{reach goal})$: From any state, the goal can be reached in less than 10 steps with a probability of at least 0.95.
- $P_{>0.9}(\bigcirc C_{\leq 100}[\Diamond^{\leq 5}\alpha])$: The probability is larger than 0.9 that from the next step onward, the system reaches a state satisfying α with a total accumulated cost no greater than 100 in no more than 5 steps.

Our goal is to develop an ADP algorithm that efficiently solves the following planning problem:

Problem 1 (PCLT-constrained Optimal Planning)**.** Given an MDP $\mathbf{M} = \langle S, A, P, \mu_0, L, \mathcal{AP}\rangle$ and a PCTL state formula ϕ, the goal is to find a policy π that solves

$$\begin{aligned} \max_{\pi} \quad & \mathcal{V}^{\pi}(\mu_0) \\ \text{subject to:} \quad & S_0 \models \phi \end{aligned}$$

where S_0 is the initial state random variable conditional on μ_0, and $S_0 \models \phi$ means $\mathbf{Pr}(S_0 \models \phi) = 1$, *i.e.,* with probability one, the state formula ϕ is satisfied.

3.4 Main Result

Our approach to solving the PCTL-constrained optimal planning problem includes two steps: First, we show that a class of PCTL constraints can be translated into chance constraints. Second, we introduce an ADP algorithm that solves the optimal planning problem with chance constraints.

3.4.1 Translate PCTL formulas into Chance Constraints

Chance constraints are introduced to capture *risk-sensitive* optimization criteria in MDPs. Let $d\colon S \times A \to \mathbf{R}$ be a cost function. For a policy π, we define the cost of a state-action pair $(s, a) \in S \times A$ (resp. a state $s \in S$) as the summation of finite-horizon

(discounted) total costs encountered by the decision-maker when it starts at the state-action pair $(s, a) \in S \times A$ (resp. a state $s \in S$) and then follows policy π. Formally, for the cost of any state-action pair $(s, a) \in S \times A$:

$$D(s, a, T; \pi) = \mathbf{E}_{\rho \sim \mu^\pi}[\sum_{t=0}^{T-1} \gamma^t d(S_t = s_t, A_t = a_t) \mid s_0 = s, a_0 = a],$$

where $\rho = s_0 a_0 \cdots s_{T-1} a_{T-1} \cdots \in \mathrm{PATH}^\pi(s)$, γ is a discount factor, and $T \in \mathbf{N} \cup \{\infty\}$ is a stopping time. The cost of a state can be defined through the cost of the state-action pair. Formally,

$$D(s, T; \pi) = \sum_{a \in A} D(s, a, T; \pi)\pi(a \mid s).$$

Chance-constrained planning in MDP aims to ensure that for a given initial random state S_0, a confidence level $\beta \in (0, 1)$, and cost tolerance α, the policy π satisfies

$$\mathbf{Pr}(D(S_0, T; \pi) \geq \alpha) \leq \beta.$$

Given a path $\rho = s_0 a_0 s_1 a_1 \cdots s_{T-1} a_{T-1} \cdots \in \mathrm{PATH}^\pi(s)$, we define the cost of a path ρ as follows

$$D(\rho, T; \pi) = \sum_{t=0}^{|T-1|} \gamma^t d(s_t, a_t),$$

and the chance constraint is equivalently expressed as

$$\mathbf{Pr}(D(\rho, T; \pi) \geq \alpha) \leq \beta.$$

We introduce the notion of mixing time and use it later to determine a stopping time T for approximately satisfying time-unbounded specifications in PCTL formulas.

Definition 5 (ϵ-return Mixing Time)**.**
Let $\mathbf{M}$ be an MDP and π be an ergodic policy in $\mathbf{M}$. Given $\epsilon \in \mathbf{R}_+$, the ϵ-return mixing time of π is the smallest T such that for all $T' \geq T$, $|D(s, T'; \pi) - D(s, T; \pi)| \leq \epsilon$.

Given a policy π, we can find an upper bound of this mixing time in terms of eigenvalues of the state transition matrix $\mathbf{Pr}^\pi(s' \mid s)$ in the Markov chain $\mathbf{M}^\pi$ using methods in [111].

Next, we show how to translate PCTL formulas into chance constraints. We select the discount factor $\gamma = 1$ unless otherwise specified. The reason is that PCTL considers the probability of satisfying path formulas and total cost without discounting when the total cost is finite. We distinguish three classes of PCTL formulas.

Probabilistic formula $P_{\bowtie p}(\psi)$

We consider a class of probabilistic formula $P_{\bowtie p}(\psi)$ such that ψ is a path formula of one of the following forms: $\neg\phi_1 \,\mathsf{U}\, \phi_2$, $\mathsf{true}\,\mathsf{U}\,\phi$, and $\bigcirc\,\phi$, where ϕ, ϕ_1, and ϕ_2 are propositional logic formulas. Formally,

$$s \models_\pi P_{\bowtie p}(\psi) \iff \mathbf{Pr}(\{\rho \in \mathrm{PATH}^\pi(s) \mid \rho \models \psi\}) \bowtie p.$$

We show that a chance constraint can represent a probabilistic formula with a properly defined cost function.

Lemma 1. Given a formula $P_{\bowtie p}(\bigcirc\,\phi)$, let us define a cost function $d\colon S \times A \to \mathbf{R}$ as

$$d(s,a) = \mathbf{E}_{s' \sim P(\cdot|s,a)}\, \mathbf{1}(s' \models \phi).$$

Given a policy π, the π-induced Markov chain satisfies the formula, denoted by $\mathbf{M}^\pi \models P_{\bowtie p}(\bigcirc\,\phi)$, if and only if $D(S_0, T; \pi) \bowtie p$, and the stopping time $T = 2$.

Lemma 2. Given a formula $P_{\bowtie p}(\mathsf{true}\,\mathsf{U}\,\phi)$, let us define a cost function $d\colon S \times A \to \mathbf{R}$ as

$$d(s,a) = \mathbf{E}_{s' \sim P(\cdot|s,a)}\, \mathbf{1}(s' \models \phi)\, \mathbf{1}(s \not\models \phi).$$

Let all states in the set $\{s \in S \mid s \models \phi\}$ be sink states. Given a policy π, the π-induced Markov chain satisfies the formula, denoted by $\mathbf{M}^\pi \models P_{\bowtie p}(\mathsf{true}\,\mathsf{U}\,\phi)$, if it is one of the following cases:

- $\bowtie \in \{\geq, >\}$: $D(S_0, T_\epsilon; \pi) \bowtie p$;
- $\bowtie \in \{\leq, <\}$: $D(S_0, T_\epsilon; \pi) \bowtie p - \epsilon$,

where $\epsilon \in (0,1)$ is a small constant, and T_ϵ is an upper bound of the ϵ-return mixing time of policy π.

Proof. By the definition of cost function d, $D(S_0; \pi)$ is the probability of eventually reaching a state that satisfies ϕ; $D(S_0, T; \pi)$ is the probability of reaching a state that satisfies ϕ within T steps. Given T_ϵ is the ϵ-return mixing time, if a path of length T_ϵ has not yet visited a state that satisfies ϕ, then the probability of satisfying the path formula as we continue along this path is less than ϵ. Since the cost is non-negative, we have $D(S_0; \pi) - D(S_0, T_\epsilon; \pi) \leq \epsilon$ for a predefined positive real number ϵ. Next, we consider two cases:

Case I $\bowtie \in \{\geq, >\}$, $P_{\geq p}(\mathsf{true}\,\mathsf{U}\,\phi)$ is equivalent to $D(S_0;\pi) \geq p$. Given $D(S_0;\pi) \geq D(S_0, T_\epsilon;\pi)$, a sufficient condition for $D(S_0;\pi) \geq p$ is that $D(S_0, T_\epsilon;\pi) \geq p$. The same argument applies for strictly greater than, *i.e.,* $>$.

Case II $\bowtie \in \{\leq, <\}$, $P_{\leq p}(\mathsf{true}\,\mathsf{U}\,\phi)$ is equivalent to $D(S_0;\pi) \leq p$. Given $D(S_0;\pi) - D(S_0, T_\epsilon;\pi) \leq \epsilon$, we have $D(S_0;\pi) \leq D(S_0, T_\epsilon;\pi) + \epsilon$. A sufficient condition for $D(S_0;\pi) \leq p$ is that $D(S_0, T_\epsilon;\pi) + \epsilon \leq p$, which is equivalent to $D(S_0, T_\epsilon;\pi) \leq p - \epsilon$. The same argument applies for strictly less than, *i.e.,* $<$.

□

Lemma 3. Given a formula $P_{\bowtie p}(\neg\phi_1\,\mathsf{U}\,\phi_2)$, let us define a cost function $d\colon S \times A \to \mathbf{R}$ as

$$d(s,a) = \mathbf{E}_{s'\sim P(\cdot|s,a)}\,\mathbf{1}(s' \models \phi_2)\,\mathbf{1}(s \not\models \phi_2).$$

Let all states in the set $\{s \in S \mid s \models \phi_1 \vee \phi_2\}$ be sink states. Given a policy π, the π-induced Markov chain satisfies the formula, denoted by $\mathbf{M}^\pi \models P_{\bowtie p}(\neg\phi_1\,\mathsf{U}\,\phi_2)$, if it is one of the following cases:

- $\bowtie \in \{\geq, >\}$: $D(S_0, T_\epsilon;\pi) \bowtie p$;
- $\bowtie \in \{\leq, <\}$: $D(S_0, T_\epsilon;\pi) \bowtie p - \epsilon$,

where T_ϵ is an upper bound of the ϵ-return mixing time of policy π.

Proof. The proof is similar to that of Lemma 2. It is noted that when a system reaches a state that satisfies ϕ_1 before reaching a state that satisfies ϕ_2, it receives a cost of 0 onwards, and the path continues to evolve until the time bound T_ϵ. □

Risk neutral cost constraint formula $C_{\bowtie m}\Diamond^{\leq k}\phi$

$C_{\bowtie m}\Diamond^{\leq k}\phi$ is the expected cost of paths satisfying the formulas $\Diamond^{\leq k}\phi$. Based on the definition in [68], a policy π satisfying the constraint given an initial state s, denoted by

$$s \models_\pi C_{\bowtie m}\Diamond^{\leq k}\phi, \iff \mathbf{E}_{\rho\sim\mu^\pi(s)}[X_{\Diamond^{\leq k}\phi}(\rho)] \bowtie m,$$

where $\mathbf{E}_{\rho\sim\mu^\pi(s)}[X_{\Diamond^{\leq k}\phi}(\rho)]$ denotes the expectation of the random variable $X_{\Diamond^{\leq k}\phi}\colon \mathrm{PATH}^\pi(s) \to \mathbf{R}_{\geq 0}$ for the Markov chain $\mathbf{M}^\pi$ given the initial state s. Let $d\colon S \times A \to \mathbf{R}$ be

$$d(s,a) = c(s,a)(1 - \mathbf{1}(s \models \phi)),$$

and any state $s \models \phi$ be a sink state. For any path $\rho = s_0 a_0 s_1 a_1 \cdots \in \mathrm{PATH}^{\pi}(s)$,

$$X_{\Diamond^{\leq k}}(\rho) = D(\rho;\pi) = \left(\sum_{t=0}^{T-1} d(s_t, a_t)\right) + \bar{D}\,\mathbf{1}(s_{T-1} \not\models \phi),$$

where $T = \min(k, \min\{j \mid s_j \models \phi\})$ and $\bar{D} \gg 0$ is a penalty term, which is added when the path does not satisfy the specification $\Diamond^{\leq k}\phi$. Note that instead of assigning a cost of ∞ to a path that fails to satisfy $C_{\bowtie m}\Diamond^{\leq k}\phi$ in [68], we use a large penalty to ensure the planning problem is well-defined.

Risk-sensitive PCTL Formula

We consider a class of PCTL formulas of the following form: $\phi_1 \implies P_{\bowtie_1 p}(\bigcirc C_{\bowtie_2 m}\Diamond^{\leq k}\phi_2)$, where ϕ_1 and ϕ_2 are propositional logic formulas, *i.e.,* state formulas that use only conjunction and negation with atomic propositions in $\mathcal{AP}$. The *risk-sensitive* semantics of the formula means that if a Markov chain satisfies the formula, then starting from a state that satisfies ϕ_1, in the next step, the probability of a sampled path that satisfies $\Diamond^{\leq k}\phi_2$ with a cost $\bowtie_2 m$ is $\bowtie_1 p$, where $\bowtie_1, \bowtie_2 \in \{\geq, >, \leq, >\}$. Formally, it is defined as

$$s \models_{\pi} \phi_1 \implies P_{\bowtie_1 p}(\bigcirc C_{\bowtie_2 m}\Diamond^{\leq k}\phi_2) \iff \begin{cases} s \not\models \phi_1 \\ s \models \phi_1 \text{ and } \mathbf{Pr}(\bigcirc C_{\bowtie_2 m}\Diamond^{\leq k}\phi_2) \bowtie_1 p. \end{cases}$$

Lemma 4. Given a risk-sensitive PCTL formula $\phi_1 \implies P_{\bowtie_1 p}(\bigcirc C_{\bowtie_2 m}\Diamond^{\leq k}\phi_2)$, let us define a cost function $d\colon S \times A \to \mathbf{R}$ as

$$d(s,a) = c(s,a)(1 - \mathbf{1}(s \models \phi_2)),$$

and let any state $s \in \{s \in S \mid s \models \phi_2\}$ be a sink state. Given a policy π, the π-induced Markov chain satisfies the formula, denoted by $\mathbf{M}^{\pi} \models \phi$, if

$$\mathbf{Pr}(D(s, k+1;\pi) \bowtie_2 m) \bowtie_1 p, \text{ for all } s \models \phi_1,$$

where

$$D(s, k+1;\pi) = \mathbf{E}_{\rho\sim\mu^{\pi}(s)}\left[\left(\sum_{t=0}^{T-1} d(s_t, a_t)\right) + \bar{D}\,\mathbf{1}(s_{T-1} \not\models \phi_2)\right],$$

$\rho = s_0 a_0 s_1 a_1 \cdots \in \mathrm{PATH}^{\pi}(s)$, $T = \min(k+1, \min\{j \mid s_j \models \phi_2\})$, and $\bar{D} \gg 0$ is a penalty.

Proof. The proof is by construction and omitted. □

If we let $Y = \{s \in S \mid s \models \phi_1\}$, then the constraint is

$$\mathbf{Pr}(D(s, T; \pi) \bowtie_2 m) \bowtie_1 p, \text{ for all } s \in Y.$$

So far, given an appropriate definition of the cost function $d\colon S \times A \to \mathbf{R}$, we have shown that any PCTL formula in these three subclasses can be represented as a chance constraint of the following form: for some $\alpha \in \mathbf{R}$ and $\beta \in [0, 1]$,

$$\mathbf{Pr}(D(s, k; \pi) \geq \alpha) \leq \beta, \text{ for all } s \in Y,$$

where k is a positive integer, and $Y \subseteq S$.

Remark 1. It is noted that $D(s, k; \pi) < \alpha$ is a special case of chance constraint and can be expressed by $\mathbf{Pr}(D(s, k; \pi) \geq \alpha) \leq 0$. The conjunction of multiple PCTLs can be translated into multiple chance constraints.

Remark 2. We use the mixing time of an ergodic policy only for PCTL with time-unbounded temporal operator U , $\Diamond$ to transform planning with a constraint over an infinite horizon into planning with a constraint over a finite horizon. If the chain induced by a policy is not ergodic, then it is unclear that the chance constraints are still valid to represent PCTL constraints with time-bounded temporal operators $\mathsf{U}^{\leq k}$ and $\Diamond^{\leq k}$.

3.4.2 Formulate Stochastic Programming for PCTL Constrained Optimal Planning

We aim to develop an ADP to solve PCTL constrained optimal policy in an MDP. First, we consider the Bellman equation with the softmax operator [6],

$$\mathcal{T}\,\mathcal{V}(s) = \tau \log \sum_{a \in A} \exp\left\{ \left(R(s, a) + \gamma \sum_{s' \in S} P(s' \mid s, a)\, \mathcal{V}(s') \right) / \tau \right\}, \tag{3.1}$$

where $\tau > 0$ is a predefined temperature parameter. With the τ approaches 0, Equation (3.1) recovers the hardmax Bellman operator. The softmax Bellman operator is contracting [6].

We have the following statement based on the monotonic contraction property of the softmax Bellman operator $\mathcal{T}$.

Lemma 5. For any value function $\mathcal{V}$ that satisfies $\mathcal{T}\,\mathcal{V} \leq \mathcal{V}$, $\mathcal{V}$ is an upper bound of the value function $\mathcal{V}^*$, where $\mathcal{V}^*$ is the fixed point of the softmax operator, *i.e.,* $\mathcal{T}\,\mathcal{V}^* = \mathcal{V}^*$.

Next, we introduce a linear function approximator of $\mathcal{V}$ as follows:

$$\mathcal{V}_\theta(s) \approx \sum_{k=1}^{K} \phi_k(s)\theta_k = \Phi\theta,$$

where $\phi_k(s)\colon S \to \mathbf{R}$, $\Phi = \{\phi_k \mid 1 \le k \le K\}$ is a set of preselected basis functions, and $\theta = [\theta_1, \cdots, \theta_K] \in \mathbf{R}^K$ is a weight vector.

Given the approximate value function $\mathcal{V}_\theta$ parameterized by θ, one can obtain the corresponding state-action value function $\mathcal{Q}_\theta$ and policy π_θ using the Bellman Equation (3.1):

$$\mathcal{Q}_\theta(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a)\, \mathcal{V}_\theta(s'), \tag{3.2}$$

$$\pi_\theta(a \mid s) = \exp((\mathcal{Q}_\theta(s,a) - \mathcal{V}_\theta(s))/\tau). \tag{3.3}$$

Given a value function approximation $\mathcal{V}_\theta(s) = \Phi(s)\theta$, the goal is to search for a function parameter $\theta \in \mathbf{R}^K$ that solves the following optimization problem:

$$\begin{aligned} \min_{\theta} \quad & \sum_{s \in S} c(s)\, \mathcal{V}_\theta(s) \\ \text{subject to:} \quad & \mathcal{T}\,\mathcal{V}_\theta(s) - \mathcal{V}_\theta(s) \le 0, \text{ for all } s \in S && \text{(3.4a)} \\ & \mathbf{Pr}(D_\theta(s,k) \ge \alpha) - \beta \le 0, \text{ for all } s \in Y && \text{(3.4b)} \end{aligned}$$

where the constraint (3.4b) is the chance constraint introduced by the PCTL formula.

Next, we develop an ADP method for the problem (3.4). Our approach is based on randomized optimization [118] that iteratively searches for an optimal parameter minimizing a weighted distance between the upper bound given by the value function approximation and the true softmax value function.

We first introduce a continuous function $h\colon \mathbf{R} \to \mathbf{R}_+$ with support equal to $[0, \infty)$, in the sense that

$$h(x) = 0 \text{ for all } x \in (-\infty, 0], \text{ and } h(x) > 0 \text{ for all } x \in (0, \infty). \tag{3.5}$$

One such function is $h(x) = \max(x, 0)$. Let

$$g_\theta(s) = \mathcal{T}\,\mathcal{V}_\theta(s) - \mathcal{V}_\theta(s), \text{ for all } s \in S.$$

Then, the constraints in (3.4a) become $g_\theta(s) \le 0$ for all $s \in S$. Using randomized optimization [117], an equivalent representation of the set of constraints in (3.4a) is

$$\mathbf{E}_{s \sim \mu_1}\, h(g_\theta(s)) = 0,$$

where s is a random variable with a distribution μ_1 whose support is S.

Similarly, let

$$\ell_\theta(s) = \mathbf{Pr}(D_\theta(s,k) \geq \alpha) - \beta, \text{ for all } s \in Y.$$

The equivalent representation of (3.4b) is

$$\mathbf{E}_{s\sim\mu_2}\, h(\ell_\theta(s)) = 0,$$

where s is a random variable with a distribution μ_2 over Y.

Thus, problem (3.4) is equivalent to:

$$\begin{aligned} \min_\theta \quad & \sum_{s\in S} c(s)\, \mathcal{V}_\theta(s) \\ \text{subject to:} \quad & \mathbf{E}_{s\sim\mu_1}\, h(g_\theta(s)) = 0 \\ & \mathbf{E}_{s\sim\mu_2}\, h(\ell_\theta(s)) = 0 \end{aligned} \tag{3.6}$$

State-relevant weights We select the state-relevant weight $c(s) = c_\theta(s)$ to be the frequency with which different states are expected to be visited in the Markov chain induced by the policy π_θ, which is computed from $\mathcal{V}_\theta$ using Equation (3.3). To justify this choice, we rephrase the following result in [34] with softmax Bellman operator and chance constraints.

Lemma 6 (Extended from Lemma1 in [34])**.** A vector θ^* solves the problem (3.4)

$$\begin{aligned} \min_\theta \quad & \sum_{s\in S} c(s)\Phi(s)\theta \\ \text{subject to:} \quad & \mathbf{E}_{s\sim\mu_1}\, h(g_\theta(s)) = 0 \\ & \mathbf{E}_{s\sim\mu_2}\, h(\ell_\theta(s)) = 0 \end{aligned}$$

if and only if it solves

$$\begin{aligned} \min_\theta \quad & \sum_{s\in S} c(s)|\mathcal{V}^*(s) - \Phi(s)\theta| \\ \text{subject to:} \quad & \mathbf{E}_{s\sim\mu_1}\, h(g_\theta(s)) = 0 \\ & \mathbf{E}_{s\sim\mu_2}\, h(\ell_\theta(s)) = 0 \end{aligned}$$

Proof. It is well known that softmax operator $\mathcal{T}$ is monotonic. From this and the fact that $\mathcal{T}$ is a contraction with fixed point $\mathcal{V}^*$, it follows that for any $\mathcal{V}$ with $\mathcal{V} \geq \mathcal{T}\,\mathcal{V}$, we have

$$\mathcal{V} \geq \mathcal{T}\,\mathcal{V} \geq \mathcal{T}^2\,\mathcal{V} \geq \cdots \geq \mathcal{V}^*\,.$$

Here, any θ that is a feasible solution to the optimization problems of interest satisfies $\Phi\theta \geq \mathcal{V}^*$. It follows that

$$\sum_{s\in S} c(s)|\mathcal{V}^*(s) - \Phi(s)\theta| = \sum_{s\in S} c(s)\Phi(s)\theta - \sum_{s\in S} c(s)\,\mathcal{V}^*(s),$$

and minimizing $\sum_{s\in S} c(s)\Phi(s)\theta$ is therefore equivalent to minimum $\sum_{s\in S} c(s)|\mathcal{V}^*(s) - \Phi(s)\theta|$. □

For any state-relevant weight $c \in \Delta(S)$, it holds that $\min_\theta \sum_{s\in S} c(s)|\mathcal{V}^* - \Phi\theta| \geq \min_\theta \sum_{s\in S} c_\theta(s)|\mathcal{V}^* - \Phi\theta|$ for the $c_\theta(\cdot)$ defined above as the state-relevant weights. According to Theorem 1 of [34], the ideal weight is to choose c that captures the (discounted) frequency with which different states are expected to be visited. For a given parameter θ, the weight function $c_\theta(\cdot)$ can be obtained from the on-policy sampling of trajectories with policy π_θ.

The augmented Lagrangian function of problem (3.6) is

$$\begin{aligned}\mathcal{L}_\theta(\lambda,\xi,\nu) = &\sum_{s\in S} c_\theta(s)\,\mathcal{V}_\theta(s) + \lambda\cdot\mathbf{E}_{s\sim\mu_1}\,h(g_\theta(s)) \\ &+ \frac{\nu}{2}\cdot|\mathbf{E}_{s\sim\mu_1}\,h(g_\theta(s))|^2 + \xi\cdot\mathbf{E}_{s\sim\mu_2}\,h(\ell_\theta(s)) \\ &+ \frac{\nu}{2}\cdot|\mathbf{E}_{s\sim\mu_2}\,h(\ell_\theta(s))|^2,\end{aligned} \tag{3.7}$$

where λ and ξ are the Lagrange multipliers, k denotes the k-th outer iteration, and ν is a large penalty constant. If the MDP is known, then the approximately optimal value function can be solved by the quadratic penalty function method [16], which consists of solving a sequence of *inner* optimization problems of the form:

$$\min_{\theta\in\mathbf{R}^K} \quad \mathcal{L}_\theta(\lambda^k,\xi^k,\nu^k) \tag{3.8}$$

where $\{\lambda^k\}$ and $\{\xi^k\}$ are sequences in $\mathbf{R}$, $\{\nu^k\}$ is a positive penalty parameter sequence, and K is the size of θ. The stopping criterion for each inner optimization problem is $\|\nabla_\theta\mathcal{L}_{\theta^j}(\lambda^k,\nu^k)\| \leq \epsilon^k$, where $\{\epsilon^k\}$ is a positive sequence converging to 0. In practice, we chose a sequence of $\{\epsilon^k\}$ as $\{m^k\epsilon^0 \mid k \geq 0\}$, where $m \in (0,1)$.

After the inner optimization for (3.8) converges, we update multipliers λ and ξ in the *outer* optimization as

$$\lambda^{k+1} = \lambda^k + \nu^k\cdot\mathbf{E}_{s\sim\mu_1}\,h(g_{\theta^k}(s)), \tag{3.9}$$

$$\xi^{k+1} = \xi^k + \nu^k\cdot\mathbf{E}_{s\sim\mu_2}\,h(\ell_{\theta^k}(s)), \tag{3.10}$$

The outer optimization stops when it reaches the maximum number of iterations or $\|\nabla_\theta \mathcal{L}_{\theta^j}(\lambda^k, \xi^k, \nu^k)\| \leq 10^{-3} \times \epsilon^k$, which essentially means the derivative is ignorable. Once the outer iteration stops, the near-optimal value function is learned. We can obtain the corresponding state-action value function and policy by equations (3.2) and (3.3).

We refer the readers to [16, Chap. 5.2] for more details about the quadratic penalty function method, its stopping criteria, and multiplier update methods.

3.4.3 Model-based ADP for PCTL Constrained Planning

We develop a model-based ADP for PCTL constrained planning to tackle the problem (3.6), which shows that the gradient of $\mathcal{L}_{\theta^k}(\lambda^k, \xi^k, \nu^k)$ can be computed from sampled trajectories.

Slightly abusing the notation, let $\mathbf{M}_\theta$ be a Markov chain induced by policy π_θ. By selecting $c_\theta(s) = \sum_{t=0}^{\infty} \mathbf{Pr}(S_t = s)$ as the state visitation frequency in $\mathbf{M}_\theta$, for an arbitrary function $f\colon S \to \mathbf{R}$, it holds that

$$\sum_{s \in S} c_\theta(s) f_\theta(s) = \sum_{\rho \in \mathrm{PATH}^{\pi^\theta}} \mu_\theta(\rho) f_\theta(\rho), \tag{3.11}$$

where $\mu_\theta(\rho)$ is the probability of the path ρ in the Markov chain $\mathbf{M}_\theta$, and we have

$$f_\theta(\rho) = \sum_{t=0}^{|\rho|-1} f_\theta(s_t),$$

where $\rho = s_0 a_0 s_1 a_1 \cdots \in \mathrm{PATH}^{\pi^\theta}$,

Furthermore, by selecting $\mu_1 \propto c_\theta$ and letting

$$f_\theta(s) = \mathcal{V}_\theta(s) + \lambda^k \cdot h(g_\theta(s)) + \frac{\nu^k}{2} \cdot |h(g_\theta(s))|^2,$$

and

$$m_\theta(\mu_2) = \xi^k \cdot \mathbf{E}_{s \sim \mu_2} h(\ell_\theta(s)) + \frac{\nu^k}{2} \cdot |\mathbf{E}_{s \sim \mu_2} h(\ell_\theta(s))|^2,$$

the k-th objective function in problem (3.8) becomes

$$\min_\theta \underbrace{\sum_{\rho \in \mathrm{PATH}^{\pi^\theta}} \mu_\theta(\rho) f(\rho)}_{F(\theta)} + m_\theta(\mu_2)$$

Using the gradient descent, parameter θ is updated by

$$\theta^{j+1} \leftarrow \theta^j - \eta_1 \cdot \nabla_\theta F(\theta) - \eta_2 \cdot \nabla_\theta m_\theta(\mu_2), \tag{3.12}$$

where j represents the j-th inner iteration, and η_1 and η_2 are positive step sizes.

$$\nabla_\theta F(\theta) = \sum_{\rho \in \text{PATH}^{\pi^\theta}} \underbrace{\nabla_\theta \mu_\theta(\rho) f_\theta(\rho)}_{1} + \sum_{\rho \in \text{PATH}^{\pi^\theta}} \underbrace{\mu_\theta(\rho) \nabla_\theta f_\theta(\rho)}_{2}, \tag{3.13}$$

where

$$\begin{aligned}
1 &= \sum_{\rho \in \text{PATH}^{\pi^\theta}} \nabla_\theta \mu_\theta(\rho) f_\theta(\rho) \\
&= \sum_{\rho \in \text{PATH}^{\pi^\theta}} \mu_\theta(\rho) \nabla_\theta \log \mu_\theta(\rho) f_\theta(\rho) \\
&= \sum_{\rho \in \text{PATH}^{\pi^\theta}} \mu_\theta(\rho) \left[\sum_{t=0}^{|\rho|-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right] f_\theta(\rho) \\
&\approx \frac{1}{|N|} \sum_{\rho \in N} \left[\sum_{t=0}^{|\rho|-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right] f_\theta(\rho) \\
&\text{(Monte Carlo approximation)} \\
2 &= \sum_{\rho \in \text{PATH}^{\pi^\theta}} \mu_\theta(\rho) \nabla_\theta f_\theta(\rho) \\
&= \sum_{\rho \in \text{PATH}^{\pi^\theta}} \mu_\theta(\rho) [\sum_{t=0}^{|\rho|-1} \nabla_\theta f_\theta(s_t)] \\
&\approx \frac{1}{|N|} \sum_{\rho \in N} \left[\sum_{t=0}^{|\rho|-1} \nabla_\theta f_\theta(s_t) \right] \\
&\text{(Monte Carlo approximation)}
\end{aligned}$$

and

$$\nabla_\theta f_\theta(s_t) = \nabla_\theta \, \mathcal{V}_\theta(s_t) + \lambda^k \cdot \nabla_g h(g_\theta(s_t)) \nabla_\theta g_\theta(s_t) + \nu^k \cdot h(g_\theta(s_t) \nabla_\theta g_\theta(s_t),$$

where we generate a set N of trajectories starting at a state with the initial distribution μ_1 to estimate the gradient. Note if $h(x) = \max(x, 0)$, then $\nabla_g h(x) = 1$ if $x > 0$ and 0 otherwise. In terms of the derivative of the second term:

$$\begin{aligned}
\nabla_\theta m_\theta(\mu_2) = \xi^k \cdot \mathbf{E}_{s \sim \mu_2} \nabla_\ell h(\ell_\theta(s)) \nabla_\theta \ell_\theta(s) \\
+ \nu^k \, \mathbf{E}_{s \sim \mu_2} \cdot h(\ell_\theta(s)) \nabla_\ell h(\ell_\theta(s)) \nabla_\theta \ell_\theta(s).
\end{aligned} \tag{3.14}$$

To estimate the gradient, we generate a set Z of trajectories starting at a state with the initial distribution μ_2. A simple choice of μ_2 is a uniform distribution over Y, the set of states subject to PCTL state formulas (3.4b).

Let $D(\rho)$ be the total cost along the trajectory ρ; we have

$$\begin{aligned}\nabla_\theta \ell_\theta(s) &\approx \nabla_\theta \left[\sum_{\rho \in \mathrm{PATH}(s)} \mu_\theta(\rho)\, \mathbf{1}(D(\rho) \geq \alpha) - \beta \right] \\ &= \sum_{\rho \in \mathrm{PATH}(s)} \nabla_\theta \mu_\theta(\rho)\, \mathbf{1}(D(\rho) \geq \alpha) \\ &= \sum_{\rho \in \mathrm{PATH}(s)} \mu_\theta(\rho) \nabla_\theta \log \mu_\theta(\rho)\, \mathbf{1}(D(\rho) \geq \alpha) \\ &\approx \frac{1}{|Z(s)|} \sum_{\rho \in Z(s)} \nabla_\theta \log \mu_\theta(\rho)\, \mathbf{1}(D(\rho) \geq \alpha),\end{aligned}$$

where

$$\nabla_\theta \log \mu_\theta(\rho) = \sum_{t=0}^{|\rho|-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t),$$

and

$$\begin{aligned}h(\ell_\theta(s)) &= h(\sum_{\rho \in \mathrm{PATH}(s)} \mu_\theta(\rho)\, \mathbf{1}(D(\rho) \geq \alpha) - \beta) \\ &= \max(\sum_{\rho \in \mathrm{PATH}(s)} \mu_\theta(\rho)\, \mathbf{1}(D(\rho) \geq \alpha) - \beta, 0) \\ &\approx \max(\frac{1}{|Z(s)|} \sum_{\rho \in Z(s)} \mathbf{1}(D(\rho) \geq \alpha) - \beta, 0).\end{aligned}$$

Again, the gradient $\nabla_\theta m_\theta(\mu_2)$ is approximated by the Monte Carlo approximation. The sample bound $|Z|$ is determined by methods in stochastic programming [35].

Using the sampling-based approach to obtain the gradient of the augmented Lagrangian, we can perform the inner and outer optimization using trajectories sampled from the MDP. In the outer optimization, the Monte Carlo approximation estimates the expectations for updating the multipliers in (3.9). Furthermore, the update of the penalty parameter does not require any new samples since both $\mathbf{E}_{s\sim\mu_1}\, h(g_{\theta^k}(s))$, and $\mathbf{E}_{s\sim\mu_2}\, h(\ell_{\theta^k}(s))$ can be evaluated based on sampled trajectories.

In the analysis of the convergence, we assume:

A1 For $y \in \{\eta_1, \eta_2\}$, $y^k > 0$ for any $k \geq 1$, $\sum_{k=1}^{\infty} y^k = \infty$, and $\sum_{n=1}^{\infty} (y^k)^2 < \infty$.

A2 The value function approximation is continuously differentiable in θ, and $\nabla_\theta \mathcal{V}_\theta$ is locally Lipschitz continuous.

A3 There exists a feasible solution for the problem (3.6).

A4 For time-unbounded PCTL, the length of a sampled trajectory is lower bounded by either the time bound in PCTL formulas or the mixing time of the Markov chain with policy parameterized by θ.

A1 and A3 are standard requirements for the augmented Lagrangian method to converge. A2 satisfies the linear function approximator and other differentiable value function approximations. A4 is to validate the PCTL constraint satisfaction using sampling-based statistical model checking.

Theorem 1. Assuming A1-A4, the sequence of value function updates converges almost surely (with probability 1) to a locally optimal solution θ^* for the problem (3.4).

Proof. The convergence proof is standard for stochastic programming and omitted for space limitation. □

The following establishes the probabilistic completeness of the proposed ADP algorithm.

Theorem 2 (Probabilistic Completeness)**.** If there exists an optimal value function, then the probability that our ADP algorithm will a solution approaches 1 as the number of paths approaches ∞.

Proof. When the number of paths approaches ∞, then derivatives (3.13) and (3.14) are approximately exactly. Then the gradient descent step (3.12) guarantees to converge following the theorem 1. □

Complexity In MDP planning problem, the number of decision variables grows linearly in the state space and action space of the MDP, where the state space is the product of the states of the system states and the DFA states. Further, this optimal planning problem can be solved in polynomial time in the size of the decision variables [81]. However, because we use a linear function to approximate the value function, the number of decision variables becomes independent of the size of the MDP. Thus, ADP methods [17] allow one to address the problem of scalability in planning for stochastic systems.

3.5 Case Study

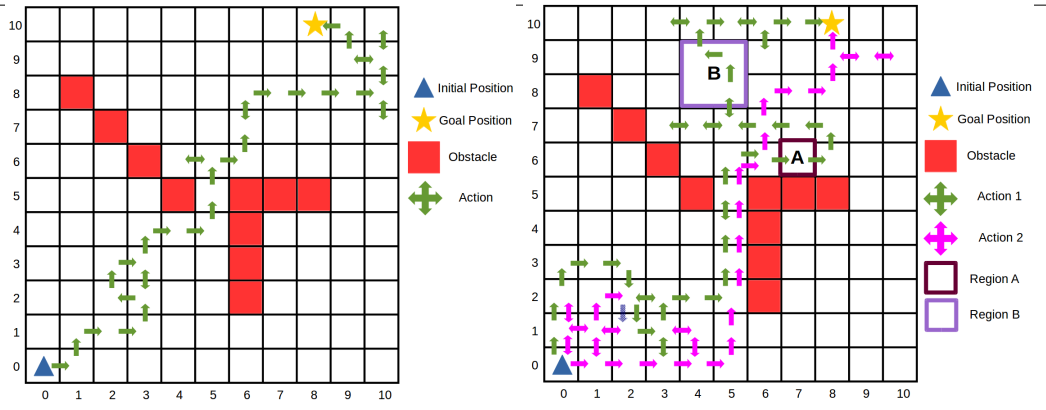
We validate the efficacy of the proposed algorithm with two robotic motion planning examples in stochastic grid worlds illustrated in Figure 3.1. The first example is a reach-avoid task shown in Figure 3.11, and the second is an optimal planning problem with a PCTL constraint shown in Figure 3.12. In both examples, for each state $s \in S$, the robot has 4 different actions: head up ("U"), head down ("D"), head left ("L"), head right ("R"). The probability of arriving at the correct cell is $1 - 0.1 \times N$, where N is the number of the neighbors of the current state, including itself. If the system hits the wall, it will be bounced back to its original cell.

3.5.1 Planning without PCTL Constraints

The first experiment is designed to observe the relation between the state-relevance weights and the approximation error and to justify the choice of state-relevance weights in the problem (3.4). The planning objective is to find an approximately optimal policy that drives the robot from the initial position $s_0 = [0, 0]$ to the goal goal $= [8, 10]$ while avoiding obstacles. The reward is defined as the following: the robot receives a positive reward of 100 when executing an action $a \in A$ at the state $s \in S$ only if $P(s_{goal} \mid s, a) > 0.5$.

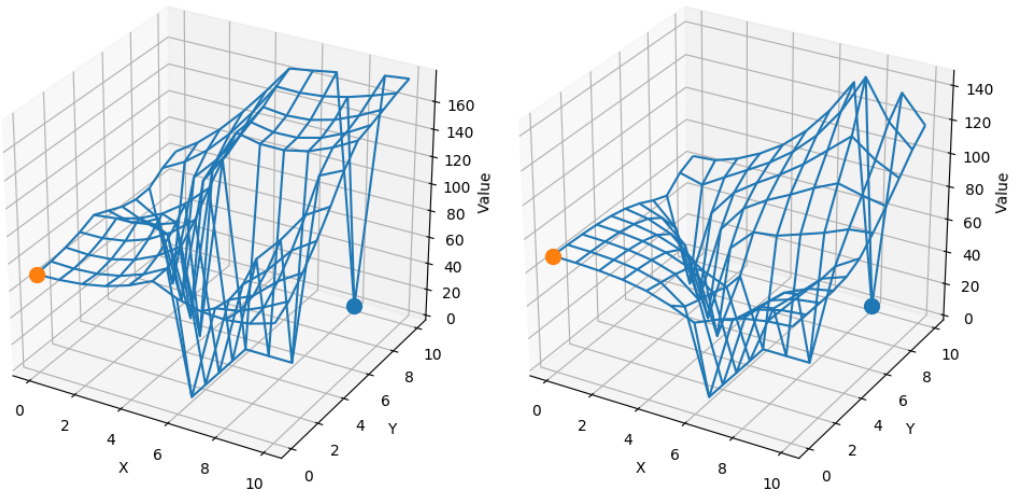
The value function approximation is $\mathcal{V}_\theta(s) = \Phi\theta$, where the basis functions $\Phi = [\phi_1, \phi_2, \cdots, \phi_K]$ are geodesic Gaussian kernels [115] defined as the following: $\Phi_j(s) = k(s, c_j)$ and $k(s, s') = \exp(-\frac{SP(s,s')^2}{2\sigma^2})$, where $\{c_j \mid 1 \leq j \leq K\}$ is a set of preselected centers. In this example, we select the centers to be $\{(x, y) \mid x, y \in \{0, 5, 10\}\}$ and the variance σ to be 5. The term $SP(s, s')$ refers to the shortest path from state $s \in S$ to state $s' \in S$ in the graph, assuming deterministic transitions.

The parameters used in the algorithm are the following (with respect to Section 3.4.3): the temperature parameter $\tau = 5$, the growth rate $b = 1.1$, the learning rate $\eta_1 = 0.1$, the initial penalty parameter $\nu^0 = 10.0$, the initial Lagrangian multipliers $\lambda^0 = 0$. During each iteration, 30 trajectories of length ≤ 6 are sampled. The algorithm converges after 4 outer iterations, with $164, 24, 8, 1$ iterations for each outer iteration. Figure 3.3 shows the result of 100 independent experiments for solving the same planning problem starting with the same initial parameter θ, which is a zero vector. The black line represents the mean, the shaded area is limited by the maximum and minimum values over iterations, and the red line is the ground truth. The black line is above the red line after the convergence. This is expected as the value function approximation is an upper bound of the optimal value



(1) (2)

Figure 3.1: (1) The grid world with initial (triangle), goal (star), obstacles (solid squares), and no PCTL constraints. (2) The grid world with PCTL constraint $a \implies \mathbf{Pr}_{\geq\delta}(C_{\leq 13} \Diamond^{\leq 14} b)$, where A and B are regions marked in the graph.



(1) (2)

Figure 3.2: (1) Approximate value function obtained with the proposed ADP method. (2) The ground truth value function obtained with softmax value iteration.

function.

To illustrate the approximation error, we compare the optimal and approximate-optimal value functions depicted in Figure 3.41, which plots the error $\mathcal{V}_\theta(s) - \mathcal{V}^*(s)$ on each state $s \in S$. Figure 3.42 shows state visitation frequency under the computed policy. It shows that the error tends to be very small for a state with a high visitation frequency under the optimal policy. This result is expected due to our choice of the state-relevance weights

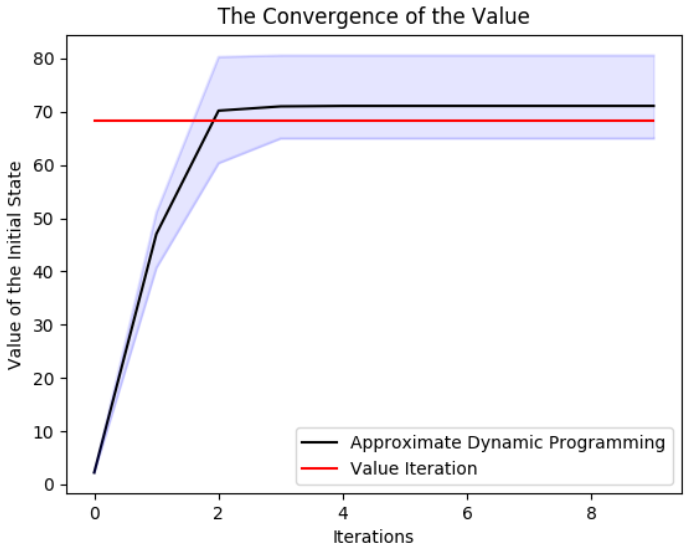
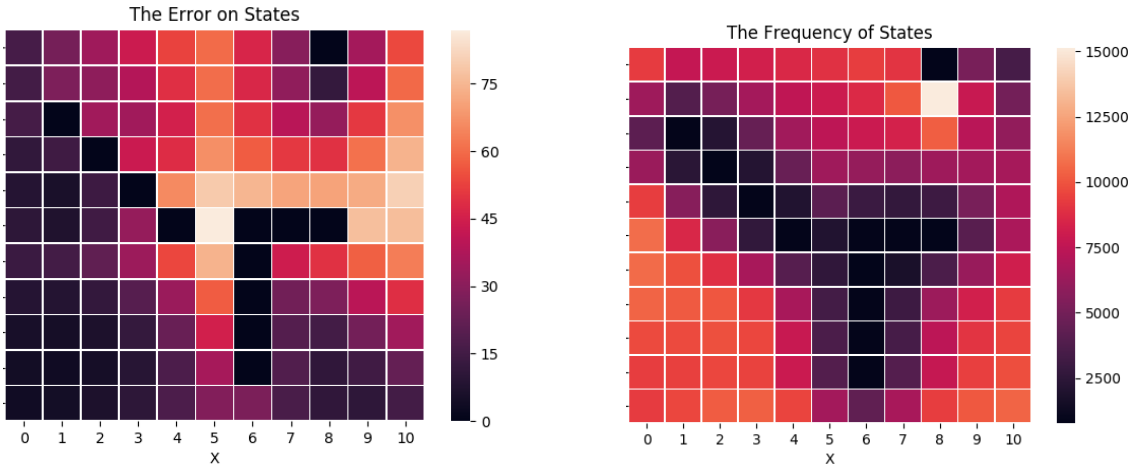


Figure 3.3: The learning curve for the stochastic grid world with no PCTL formulas, averaged across 100 runs of the ADP algorithm with the same initialization.

that have larger weights for states with high visitation frequencies. Figure 3.11 plots one simulation generated by following the computed policy.



(1) (2)

Figure 3.4: (1) The error heatmap $\mathcal{V}_{\theta^*}(s) - \mathcal{V}^*(s)$. (2) The state visitation frequency heatmap under policy π_{θ^*}.

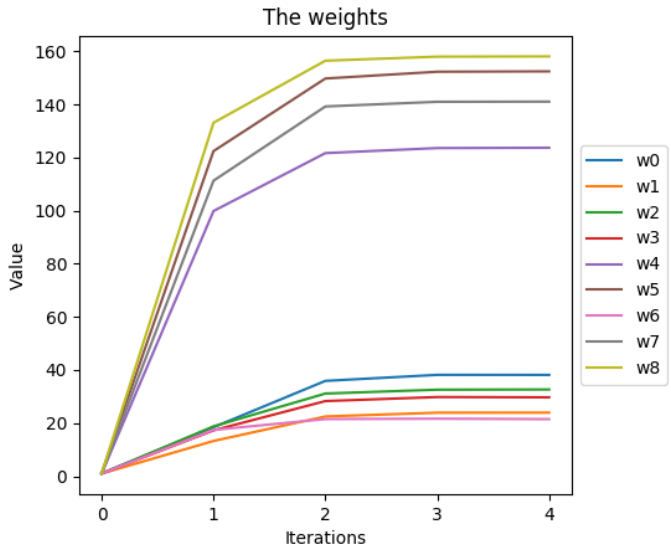


Figure 3.5: The Convergence of Parameters.

3.5.2 Planning with PCTL Constraints

Let us consider a set $\mathcal{AP}$ of atomic propositions:

- a: robot reaches region A.
- b: robot reaches region B.

Given such a set $\mathcal{AP}$ of atomic propositions, we have a PCTL constraint $a \implies \mathbf{Pr}_{\geq\delta}(\bigcirc C_{\leq 13} \Diamond^{\leq 14} b)$: When the agent visits region A, then it will ensure, starting from the next state, with a probability of at least 0.2, to eventually visit region B in less than 14 steps with a cost less than 13. Region A and B are shown in Figure 3.12. Let $d\colon S \times A \to \mathbf{R}$ be defined by $d(s,a) = 1, \forall (s \times a) \in S \times A$, *i.e.,* receiving a cost of 1 if an action is taken.

We use the same value function approximation, the same stopping criterion for the inner optimization, and the same set of parameters with different initial penalty parameters $\nu_1^0 = 10.0$ and $\nu_2^0 = 500$ for constraints (3.4a) and (3.4b), respectively. For each iteration, there are 30 trajectories (of lengths ≤ 6) sampled and another 100 trajectories (of lengths $= 15$) starting from region A for the chance constraints. Given $\delta = 0.2$, Figure 3.5 shows the convergence of the parameters. Even by adding PCTL constraints, the algorithm converges after 4 outer iterations. Figure 3.12 shows two trajectories simulated by following the computed policy. In one sampled trajectory, the system reaches A and then B with a cost

Table 3.1: Frequencies of satisfying paths under different δ.

The value of δ	0.0	0.1	0.2	0.3
Number of satisfying paths	3130	2227	4581	7410
Percentage of satisfying paths	0.15	0.11	0.23	0.37

less than the given threshold. In another sampled trajectory, the system does not visit region A and directly goes to the goal.

We tested the algorithm with different values for δ. Table 3.1 shows the frequencies of trajectories satisfying the cost constraint $\bigcirc C_{\leq 13} \Diamond^{\leq 14} b$ under different values for δ after the empirical evaluation of 2×10^4 trajectories starting from A. In all experiments, the PCTL constraint is satisfied. The result shows that as δ increases, the probability of satisfying the constraint increases, but not monotonically.

3.6 Conclusion

We have presented an approximate dynamic programming method for MDP with PCTL constraints. We proposed a method that first translates PCTL constraints into chance constraints and then uses stochastic programming for solving an upper bound of the optimal value function subject to constraints where PCTL formulas are encoded. We provide the almost sure convergence of the proposed algorithm under several assumptions. In the next chapter, we are interested in extending this method to a large class of temporal logic formulas, for which finite memory is needed for optimality.

Chapter 4

Topological Approximate Dynamic Programming with Temporal Logic Specifications

4.1 Overview

This chapter extends the ADP method for a stochastic system modeled as an MDP. The planning objective is to maximize the (discounted) probability of satisfying constraints expressed in a subclass of temporal logic—scLTL formulas [12].

This chapter proposes a different approach to mitigate the challenges in RL with sparse reward signals from LTL other than reward shaping. Our approach is inspired by the efficient value iteration [30]: In an acyclic MDP, an optimal backup order exists, such that each state in the MDP only needs to perform a one-step backup operation in value iteration [18]. Authors [30] generalize the optimal backup order for an acyclic MDP to a general MDP. They develop a topological value iteration method that divides an MDP into several strongly connected components (SCC). Then, they solve the values of states for each component sequentially in the computed topological order. Although it seems straightforward to apply topological value iteration to the product MDP, obtained by augmenting the original MDP with a finite set of memory states related to the task, the solution suffers from the scalability issue. However, the ADP method in section 3.4 can mitigate this. The ADP method uses value function approximations to approximate the optimal solutions for large-scale problems. Hence, we propose a topological approximate dynamic programming (TADP) method that includes two stages. Firstly, we translate the

task formula into a DFA referred to as the *task DFA*. We exploit the graphical structure in the task automaton to determine a topological optimal backup order for *a set of value functions*—one for each discrete state in the task DFA. Transitions in the task DFA relate value functions that jointly determine the optimal policy based on the Bellman equation. Secondly, we introduce function approximations for the set of value functions to reduce the number N of decision variables—the number of states in the product MDP—to a number M of weights in function approximations, where $M \ll N$. Finally, we integrate an ADP method with the backup ordering to solve the set of value function approximations for each task state in an optimal order. By doing this, the learning algorithm receives meaningful gradient information that gets propagated back to earlier stages of task completion when the system receives a reward when it completes a task in later stages.

Exploiting the structure of a task DFA for planning has been considered in [106], where the authors partition the task DFA into SCC and then define progress levels towards satisfaction of the specification. This chapter formally defines a topological backup order based on the causal dependency among states in a task DFA. We prove the optimality of this backup order. Further, this backup order potentially can be integrated with the actor-critic method for LTL-constrained planning in [126] or other ADP methods that solve value function approximations to address the sparse reward problem.

The rest of the chapter is structured as follows. Section 4.3 provides the necessary background on modeling stochastic systems under temporal objectives in LTL formulas. Section 4.4 contains the main results, including computing the topological order, proof of optimality in this order, and the TADP algorithm. The correctness and effectiveness of the proposed method are experimentally validated in Section 4.5 with a robotic motion planning example. Section 4.6 summarizes.

4.2 Related Work

When a high-level specification is in an LTL formula, the problem becomes synthesizing a policy that maximizes the probability of satisfying an LTL formula, which was first introduced by Ding *et al.* [39] in 2011. The two major ways of approaching this satisfaction problem are automaton-based and constraint-based approaches.

Constraint-based approaches [70, 110, 129] formulate constrained optimizations, where LTL formulas are encoded in constraints. These constrained optimization approaches leverage the modern solvers and achieve substantial accomplishments in theoretical results [129]

and applications [66]. Especially, Wolff *et al.* [129] formulated a mixed-integer linear programming (MILP) problem for optimal control of nonlinear systems. They encoded LTL specifications as mixed-integer linear constraints to avoid the construction of the task automaton. However, MILP suffers from mathematical difficulties when the problem size becomes large because solving a MILP is an NP-complete problem. Shoukry *et al.* [110] investigated a multi-robot motion planning problem with tasks expressed in a subset of LTL formulas. They first formulated a feasibility problem using Boolean and convex constraints. They then followed a satisfiability modulo convex programming approach to decompose the problem into efficiently solvable smaller problems. However, constraints of these constraint-based methods generally only tackle a subclass to LTL formulas.

Alternatively, works following automaton-based approaches [46, 56, 57, 73] either learn an approximate model and then use model-based methods or directly use model-free methods to address the lack of system model knowledge. For a model-based approach, Fu *et al.* [46] first proposed an algorithm that learns a probably approximately correct MDP. Then they applied value iteration (model-based) to synthesize a policy. However, learning a probably approximately correct MDP takes extra computational time. On the other side, Hasanbeig *et al.* [57] proposed a model-free RL algorithm (*i.e.,* Q-learning) to produce a policy for an product MDP with a synchronous reward function dependent on the acceptance condition. This approach's theoretical soundness opened a vast research area for applying state-of-art model-free reinforcement learning algorithms to learn optimal policy following the automaton-based approach. However, their work is still only demonstrated on discrete systems.

To improve learning efficiency, hierarchical planning is developed [38, 64, 105]. Jothimurugan *et al.* [64] developed a compositional learning approach that interleaves high-level planning and RL algorithm. First, this learning approach encoded the specification as an abstract graph; intuitively, the vertices and edges of the graph correspond to regions of the state space and simpler sub-tasks, respectively. Their approach then incorporated RL to learn neural network policies for each edge (sub-task) within a Dijkstra-style planning algorithm to compute a high-level plan in the graph. However, this work still requires a dynamic system model in high-level planning.

Despite the theoretical success and impressive experimental demonstration of following the automaton-based method and adopting model-free reinforcement learning algorithms [48, 75], the sparse reward remains a challenge when a system only receives a few reward signals in a gigantic/continuous space, leading to no gradient information.

Icarte *et al.* [61] proposed reward machines, a finite state machine that supports the specification of rewarding function while exposing reward function structure. In doing so, the agent could exploit the function's internal structure to learn optimal policies more sample-efficiently. They introduced different methodologies to exploit this structure to support learning, including automated reward shaping, task decomposition, and counterfactual reasoning with off-policy learning. Further, they showed that the reward machines have the expressive power for regular language and support loops, sequences, and conditionals, as well as the expression of temporally extended properties typical of linear temporal logic and non-Markovian reward specification. Jiang *et al.* [63] presented the first reward shaping framework for average reward learning that proves that, under standard assumptions, the optimal policy under the original reward function can be recovered. They used a temporal logic formula to construct the shaping function that provides additional rewards. However, it is difficult to find a reward shaping function to keep policy invariant.

4.3 Preliminaries: Product MDP

Recall that optimal planning for MDPs with high-level specifications in LTL formulas falls into two categories: automaton-based and nonautomaton-based. This section provides the necessary background for optimal planning following the automaton-based approach. We introduce the product MDP to model the stochastic system with a high-level specification in scLTL, a subclass of LTL.

Definition 6 (Product MDP)**.**
Given a labeled MDP $\mathbf{M} = \langle S, A, P, s_0, L, \mathcal{AP}\rangle$ and a DFA $\mathcal{A} = \langle Q, \Sigma, \delta, \iota, F\rangle$ associated with an scLTL formula φ, a product MDP is a tuple

$$\mathcal{M} = \mathbf{M} \otimes \mathcal{A} = \langle Z, A, \Delta, z_0, \mathcal{F}\rangle,$$

where the components of $\mathcal{M}$ are defined as follows:

- $Z\colon S \times Q$ is a set of product states. Every product state $z = (s, q) \in Z$ has two components:
  - s is a state in MDP.
  - q is an automaton state keeping track of the progress towards satisfying the specification.

- A is a set of actions inherited from MDP.
- $\Delta\colon Z \times A \times Z \to [0,1]$ is a new transition probability function: for each product state $z = (s,q) \in Z$, action $a \in A$, and next product state $z' = (s',q') \in Z$,

$$\Delta((s',q') \mid (s,q),a) = P(s' \mid s,a)\,\mathbf{1}(q' = \delta(q, L(s'))).$$

- $z_0 = (s_0, q_0)$ is the initial state that includes the initial state s_0 in MDP and $q_0 = \Delta(\iota, L(s_0))$, where ι is the initial state of the DFA $\mathcal{A}$.
- $\mathcal{F} = S \times F$ is the set of final states, where F is the set of accepting states in DFA $\mathcal{A}$. Any product state $(s,q) \in \mathcal{F}$ is a sink/absorbing state. By entering these states, the system satisfies the specification and will never come out again.

Recall from Chapter 2, a finite path $\rho = s_0 a_0 s_1 a_1 \cdots \in \mathrm{PATH}$ satisfies the formula φ; we denote it as $\rho \models \varphi$. We are interested in solving a *MaxProb* problem defined as follows.

Problem 2 (MaxProb Problem)**.** Given an MDP and a high-level specification expressed in an scLTL formula φ, the *MaxProb* is to learn an optimal policy π^* that maximizes the probability of satisfying the formula φ. Formally,

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}}\, \mathbf{E}_{\rho_t \sim \mu^\pi}[\sum_{t=0}^{\infty} \mathbf{1}(\rho_t \models \varphi)], \tag{4.1}$$

where $\rho_t = s_0 a_0 s_1 a_1 \ldots s_t \in \mathrm{PATH}^\pi$.

To solve the MaxProb problem, we introduce the reward function defined over the product MDP. The reward function $R\colon Z \times A \to \mathbf{R}$ maps the current product state and action into a real value, where $R(z,a)$ is the reward for executing action $a \in A$ at product state $z \in Z$. Formally, for each product state $z = (s,q) \in Z$, action $a \in A$, and next product state $z' = (s',q') \in Z$,

$$R(z,a) = \mathbf{1}(z \in Z \setminus \mathcal{F}) \sum_{z' \in Z} \Delta(z' \mid z,a)\,\mathbf{1}(z' \in \mathcal{F}). \tag{4.2}$$

Equation (4.2) describes that a reward is only received transiting from a product state not in final states $\mathcal{F}$ to a product state in final states $\mathcal{F}$. Given the defined reward function (4.2), the MaxProb problem becomes an optimal planning problem on product MDP whose objective is to maximize the excepted sum of rewards. We term the excepted

sum of rewards as value function $\mathcal{V}$. A value function $\mathcal{V}$ starting at a product state $z = (s, q) \in Z$ following policy π is defined as follows:

$$\mathcal{V}^{\pi}(z) = \mathbf{E}_{\rho \sim \mu^{\pi}(z)}[\sum_{t=0}^{\infty} \gamma^t R(z_t, a_t)], \tag{4.3}$$

where $\rho = z_0 a_0 z_1 a_1 \dots z_t a_t \cdots \in \mathrm{PATH}^{\pi}(z)$, and $\gamma \in [0, 1)$ is a discounting factor. The goal of the optimal planning problem is to learn an optimal policy that maximizes that expected sum of rewards. Slightly abusing the notation, we introduce the randomized policy $\pi \colon Z \times A \times Z \to [0, 1]$ over product MDP. The optimal policy π^* is achieved only if: for all $z = (s, q) \in Z$,

$$\pi^*(z) = \operatorname*{argmax}_{\pi \in \Pi} \mathcal{V}^{\pi}(z). \tag{4.4}$$

4.4 Main Result

We are interested in developing ADP algorithms for solving the *MaxProb* problem. However, suppose we directly solve for approximately optimal policies in the product MDP using the method in Section 3.4, as the reward is sparse. In that case, sampling a path satisfying the specification becomes a rare event. As a consequence, the estimate of the gradient in [76] has a high variance with finite samples. To address this problem, we develop TADP that leverages the structural property in the task automaton to improve the convergence due to sparse and temporally extended rewards with LTL specifications.

4.4.1 Hierarchical Decomposition and Causal Dependency

Reward function (4.2) introduces the sparse reward issue. That is because if the state space is large or continuous, the system has difficulty transiting into a state in final states $\mathcal{F}$ and receiving a reward. To address the reward sparsity, this section introduces the topological order to generalize the optimal backup order in MDP [18] to that in product MDP. The generalized optimal backup uses the causal dependence to direct the value backups, resulting in a more efficient value learning process. Optimal planning over product MDP with the generalized optimal backup order shall guarantee the optimality of the value function.

Causal dependence was first introduced in [30] to describe a state's value depending on its successors' values.

Definition 7 (Causal Dependence in MDP [30])**.**
If there exists an action $a \in A$ such that $P(s' \mid s, a) > 0$, and the Bellman equation[1] indicates $\mathcal{V}(s)$ is dependent on $\mathcal{V}(s')$, then state s causally depends on state s'.

Causal dependency suggests it is more efficient to perform backup on state s' before state s. This observation leads to the optimal backup order in MDP.

Theorem 3 (Optimal Backup Order [18])**.** If an MDP is acyclic, then there exists an optimal backup order. The optimal value function can be found with the optimal backup order, where each state needs only one backup.

We can generalize the causal dependence in Definition 7 to product MDP.

Definition 8 (Causal Dependence on Z)**.**
In product MDP $\mathcal{M}$, a state (s, q) is causally dependent on state (s', q'), the causal dependence on Z is a subset $\rightarrow$ of the set $\{((s, q), (s', q) \mid (s, q), (s', q') \in Z)\}$. If there exists an action $a \in A$ such that $\Delta((s', q') \mid (s, q), a) > 0$, then $((s, q), (s', q')) \in \rightarrow$, and we write it as $(s, q) \rightarrow (s', q')$.

But, we observe that on DFA, if there exists a symbol $\sigma \in \Sigma$ such that $q' = L(q, \sigma)$, and the automaton state q' makes more satisfaction progress than automaton state q, then we should perform backup on product state (s', q') before product state (s, q) for any $s, s' \in S$.

Given such observation, we define the *invariant set* and *guard set* in product MDP, which are generalizations of similar definitions in Markov chains [45].

Definition 9 (Invariant Set)**.**
Given an automaton state $q \in Q$ and an MDP $\mathbf{M}$, the invariant set of q with respect to $\mathbf{M}$, denoted by $\mathbf{Inv}(q, \mathbf{M})$, is a set of MDP states such that no matter which action is selected, the system has probability one to stay within the state q. Formally,

$$\mathbf{Inv}(q, \mathbf{M}) = \{s \in S \mid \forall a \in A, \forall s' \in S, P(s' \mid s, a) > 0 \implies \delta(q, L(s')) = q\}. \quad (4.5)$$

Definition 10 (Guard Set)**.**
Given automaton states $q, q' \in Q$ and an MDP $\mathbf{M}$, the *guard* set of q and q' with respect to $\mathbf{M}$, denoted by $\mathbf{Guard}(q, q', \mathbf{M})$, is a set of MDP states where there exists an action

[1] Mellowmax operator $\mathcal{T}$ defined in Equation (4.7) is adopted in this work.

$a \in A$, and the system can transit from q to q' with a positive probability by taking such an action $a \in A$. Formally,

$$\mathbf{Guard}(q, q', \mathbf{M}) = \{s \in S \mid \exists a \in A, \exists s' \in S, P(s' \mid s, a) > 0 \wedge \delta(q, L(s')) = q'\}. \tag{4.6}$$

We provide the following example to help understand the defined invariant and guard sets.

Example 4. Going back to Example 2, given a labeled MDP $\mathbf{M} = \langle S = \{s_0, s_1, s_2\}, A = \{a_1\}, P, s_0, L, \mathcal{AP} = \{s_2\}\rangle$, the reachability objective of the MDP can be described by the formula $\Diamond\, s_2$, where the associated DFA is in Figure 4.12. By Definition 9 and 10, we

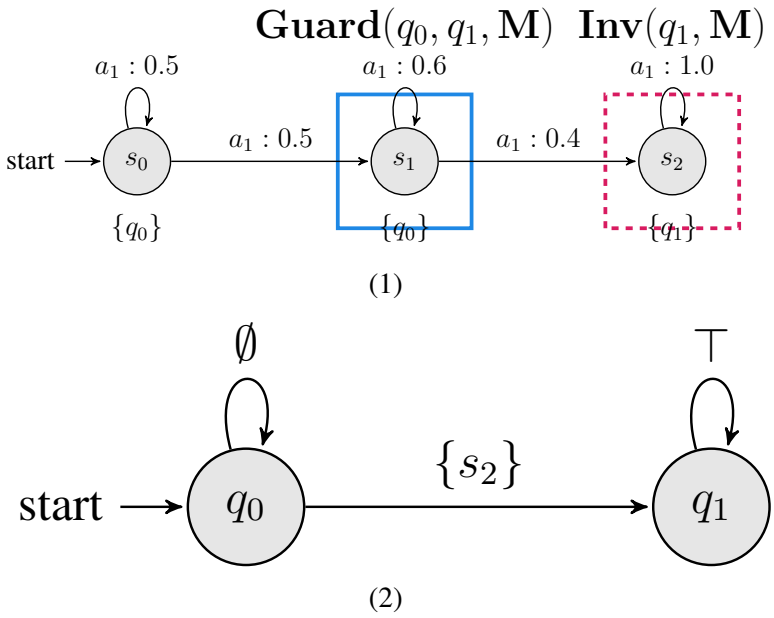


Figure 4.1: (1) $\mathbf{M} = \langle S = \{s_0, s_1, s_2\}, A = \{a_1\}, P, s_0, L, \mathcal{AP} = \{s_2\}\rangle$, where the satisfaction progress is labeled beneath the state. (2) The DFA accepting the formula $\Diamond\, s_2$.

have $\mathbf{Inv}(q_1, \mathbf{M}) = \{s_2\}$ and $\mathbf{Guard}(q_0, q_1, \mathbf{M}) = \{s_1\}$.

With the definition of the guard set in Definition 10, we define causal dependence on automaton state space Q.

Definition 11 (Causal Dependence on Q)**.**
Given a DFA $\mathcal{A}$, the causal dependence on Q is a subset $\rightarrow$ of the set $\{(q, q') \mid q, q' \in Q\}$. If $\mathbf{Guard}(q, q', \mathbf{M}) \neq \emptyset$, then $(q, q') \in \rightarrow$, and we write it as $q \rightarrow q'$.

If $q_1 \to q_2$ and $q_2 \to q_1$, then we say q_1 and q_2 are mutually causally dependent, and we denote it as $q_1 \leftrightarrow q_2$. However, when $q_1 \leftrightarrow q_2$, it becomes unclear that in which order product state (s_1, q_1) and product state (s_2, q_2) should be updated when there exists actions $a_1, a_2 \in A$ such that $\Delta((s_2, q_2) \mid (s_1, q_1), a_1) > 0$ and $\Delta((s_1, q_1) \mid (s_2, q_2), a_2) > 0$. It is natural to group these automaton states that are mutually causally dependent and update these corresponding product states together.

A *meta-mode* $X \subseteq Q$ is a subset of automaton states mutually causally dependent on each other. If an automaton state q is not mutually causally dependent on any other automaton state, then the set $\{q\}$ itself is a meta-mode.

Definition 12 (Maximal Meta-Mode)**.**
Given a DFA $\mathcal{A}$, a maximal meta-mode X is a subset of Q such that:

- For every state $q, q' \in X$, $q \leftrightarrow q'$.
- For every state $q \in X$, for every state $q' \in Q \setminus X$, $q \not\leftrightarrow q'$.

We denote the set of all maximal meta-modes as $\mathcal{X}$.

Lemma 7. The set $\mathcal{X}$ of all maximal meta-modes is a partition of automaton state space Q, *i.e.,* $Q = \cup_{X \in \mathcal{X}} X$.

Proof. By way of contradiction, if $\mathcal{X}$ is not a partition of automaton state space Q, then there exists an automaton state q such that $q \in X \cap X'$. Because q is mutually causally dependent on all states in X as well as X', then any pair $(q_1, q_2) \in X \times X'$ will be mutually causally dependent—a contradiction to the definition of $\mathcal{X}$. □

We next define the causal dependence on the set $\mathcal{X}$ of all maximal meta-modes.

Definition 13 (Causal Dependence on $\mathcal{X}$)**.**
Given Definition 12, the causal dependence on $\mathcal{X}$ is a subset $\to$ of the set $\{(X, X') \mid X, X' \in \mathcal{X}\}$. If there exists $q \in X, q' \in X'$, and $q \to q'$, then $(X, X') \in \to$, and we write it as $X \to X'$.

If $X_1 \to X_2$ and $X_2 \to X_3$, for simplicity, we write it as $X_1 \to^+ X_3$. The following lemma suggests that two product states in the product MDP can be causally dependent if their discrete dependent modes are causally dependent.

Lemma 8. Given two meta-modes $X, X' \in \mathcal{X}$, if $X \to^+ X'$ but not $X' \to^+ X$, then for any state $(s, q) \in S \times X$ and $(s', q') \in S \times X'$, one of the following holds:

- $(s, q) \to^+ (s', q')$ and $(s', q') \not\to^+ (s, q)$.
- (s, q) and (s', q') are causally independent.

Proof. We prove the first case by way of contradiction. If $(s, q) \to^+ (s', q')$ and $(s', q') \to^+ (s, q)$, then there must exist a product state (s'', q'') such that $(s', q') \to^+ (s'', q'')$ and $(s'', q'') \to (s, q)$. Relating the causal dependence on product states in Definition 8 and the definition of the guard set in Definition 10, we have $s'' \in \mathbf{Guard}(q'', q, \mathbf{M})$ and $q'' \to q$. This implies $q' \to^+ q'' \to q$, and we have $X' \to^+ X$, which is a contradiction to $X' \not\to^+ X$. The second case is obvious, and the proof is omitted. □

Lemma 8 provides structural information about backup order on product states; that is, if $X \to^+ X'$ and $X' \not\to^+ X$, then we should first update the product states $\{(s, q) \mid S \times X'\}$ then the product states $\{(s, q) \mid S \times X\}$.

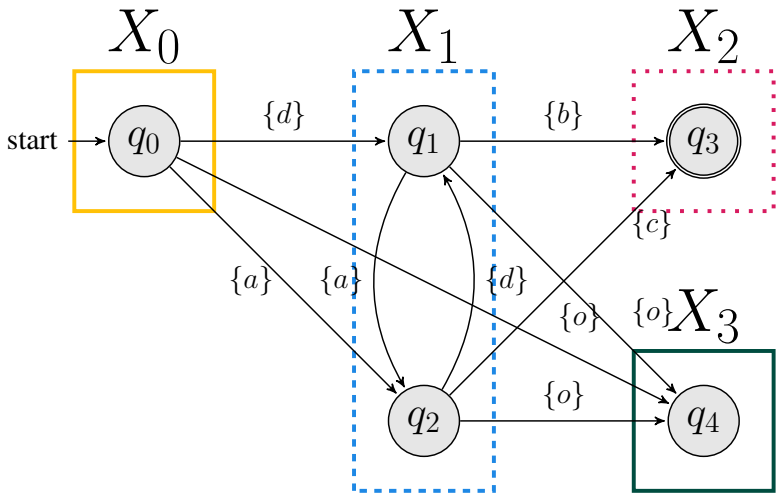


Figure 4.2: The set of maximal meta-modes $\mathcal{X} = \{X_0, X_1, X_2, X_3\}$ on the DFA accepting the formula φ.

Example 5. Continue on Example 1. We use the Kosaraju-Sharir's algorithm [3] to obtain the set of maximal meta-modes $\mathcal{X} = \{X_0, X_1, X_2, X_3\}$, where $X_0 = \{q_0\}, X_1 = \{q_1, q_2\}, X_2 = \{q_3\}$, and $X_3 = \{q_4\}$. We draw the set of maximal meta-modes in Figure 4.2. However, Lemma 8 does not provide a *total* order over $\mathcal{X}$. That is because two maximal meta-modes can be causally independent. In this example, $X_2 = \{q_3\}$ and $X_3 = \{q_4\}$ are causally independent.

Two causally independent maximal meta-modes exist in Example 5, and we cannot decide the backup order. To address this, we provide Algorithm 2 to obtain a set $\{\mathcal{L}_i \mid 0 \leq i \leq n\}$, termed a set of *level sets* over meta-modes.

Algorithm 2: Computation of Level Sets over Meta-Modes

Input: Set of maximal meta-modes $\mathcal{X}$.
Output: Set of level sets over meta-modes $\{\mathcal{L}_j\}$.
Initialization: $\mathcal{L}_0 = \{X \in \mathcal{X} \mid F \cap X \neq \emptyset \vee \{q_{\mathsf{sink}}\} \cap X \neq \emptyset\}$, $i = 1$.
1: **while** $\mathcal{L}_{i-1} \neq \emptyset$ **do**
2: $\quad \bar{\mathcal{X}} = \cup_{k=0}^{i-1} \mathcal{L}_k$
3: $\quad \mathcal{L}_i = \{X \in \mathcal{X} \setminus \bar{\mathcal{X}} \mid \exists X' \in \mathcal{L}_{i-1}, X \to X', \forall X'' \in \mathcal{X} \setminus (\bar{\mathcal{X}} \cup \{X\}), X \not\to X''\}$
4: $\quad i = i + 1$
5: **end while**
6: **return** $\{\mathcal{L}_j \mid j < i - 1\}$

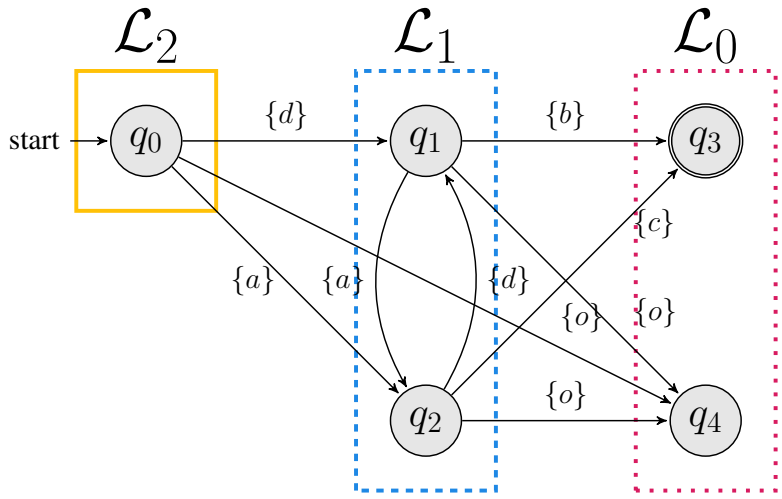


Figure 4.3: The set of level sets over meta-modes $\{\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2\}$ on the DFA accepting the formula φ.

Example 6. Continue on Example 5, we use Algorithm 2 to obtain a set of level sets over meta-modes $\{\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2\}$, where $\mathcal{L}_0 = \{X_2, X_3\}$, $\mathcal{L}_1 = \{X_1\}$, and $\mathcal{L}_2 = \{X_0\}$. We visualize the set of level sets over meta-modes $\{\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2\}$ in Figure 4.3, where $\mathcal{L}_0 = \{q_3, q_4\}$[2], $\mathcal{L}_1 = \{q_1, q_2\}$, and $\mathcal{L}_2 = \{q_0\}$.

[2]We are supposed to write $\mathcal{L}_0 = \{\{q_3\}, \{q_4\}\}$. For notional convenience, we denote an automaton state q at level $\mathcal{L}_i$ as $q \in \mathcal{L}_i$.

By way of construction, the set of level sets groups two causally independent meta-modes that depend on the same level together; We introduce *topological order* on the set of level sets.

Definition 14 (Topological Order)**.**
Given a set of level sets over meta-modes $\{\mathcal{L}_i \mid 0 \leq i \leq n\}$, a topological order is a subset $\rightsquigarrow$ of the set $\{(\mathcal{L}_j, \mathcal{L}_k) \mid \mathcal{L}_j, \mathcal{L}_k \in \{\mathcal{L}_i \mid 0 \leq i \leq n\}\}$. If $j = k + 1$, then $(\mathcal{L}_j, \mathcal{L}_k) \in \rightsquigarrow$, and we write is as $\mathcal{L}_j \rightsquigarrow \mathcal{L}_k$. We denote the topological order for the set of level sets over meta-modes $\{\mathcal{L}_i \mid 0 \leq i \leq n\}$ as follows:

$$\mathcal{L}_n \rightsquigarrow \mathcal{L}_{n-1} \rightsquigarrow \cdots \rightsquigarrow \mathcal{L}_1 \rightsquigarrow \mathcal{L}_0 \,.$$

Given Definition 14, we define the generalized optimal backup order in reverse to the topological order.

Theorem 4 (Generalized Optimal Backup Order)**.** Given a probabilistic planning problem for product MDP and the topological order, if we update the value function of each level set in reverse to the topological order $\rightsquigarrow$, then the optimal value function for each level set can be found with only one backup operation.

Proof. We show this by induction. Suppose we have a set of level sets $\{\mathcal{L}_0\}$, the problem is reduced to optimal planning in a product MDP that performs only one update for the value function in $\mathcal{L}_0$. When we have $\{\mathcal{L}_i \mid i \geq 0\}$, for $1 \leq i \leq n$, Line 2 in Algorithm 3 performs a value update for level set $\mathcal{L}_i$, where value $\mathcal{V}(s, q)$ only depends on the values of its descent states; that is, the value $\mathcal{V}(s, q)$ depends on the values of the set $\{\mathcal{V}(s', q') \mid (s, q) \to (s', q')\}$. It is noted that any descendant automaton state q' of the state (s, q) must belong to $\mathcal{L}_k$ for some $k \leq i$. It means the value of any descendant $\mathcal{V}(s', q')$ for (s, q) is either updated in level $\mathcal{L}_k$, $k < i$, or along with the value $\mathcal{V}(s, q)$ when $k = i$. As a result, when the value function $\{\mathcal{V}(s, q) \mid s \in S, q \in \mathcal{L}_i\}$ converges, it remains unchanged. Value function in higher level sets updates without affecting level i. Thus, each level set only needs to be updated once. □

Given the set of level sets over meta-modes $\{\mathcal{L}_i \mid 0 \leq i \leq n\}$, we propose Algorithm 3 for solving the planning problem over product MDP optimally. The Algorithm starts with all values being 0 for any states $(s, q) \in S \times \mathcal{L}_0$. That is because level set $\mathcal{L}_0$ only contains the final states or the sink state. Then for level i, Line 2 in Algorithm 3 can call any optimal algorithm to solve the values for any states $(s, q) \in S \times \mathcal{L}_i$ given the values for any state $(s', q') \in S \times \mathcal{L}_k$, where $i < k$, have learned.

Algorithm 3: Topological Guided Value Learning

Input: Set of level sets over meta-modes $\{\mathcal{L}_i \mid 0 \leq i \leq n\}$ and product MDP $\mathcal{M}$.
Output: Optimal value function $\mathcal{V}$.
Initialization: Let $\mathcal{V}_0(s,q) = 0$, for all $(s,q) \in S \times \mathcal{L}_0$.
1: **for** $i = 1$ to n **do**
2: Call any optimal algorithm to solve the value function $\mathcal{V}(s,q)$ for all $(s,q) \in S \times \mathcal{L}_i$.
3: **end for**

4.4.2 ADP for Planning with Temporal Logic Constraints

This section first formulates a constrained optimization problem to solve the optimal value function for level i (Line 2 in Algorithm 3) over product MDP.

The value function for level i can be solved optimally in a constrained optimization problem. First, we introduce the optimal mellowmax operation $\mathcal{T}$ as follows: for all $(s,q) \in S \times Q$,

$$\mathcal{T}\,\mathcal{V}(s,q) = \tau \log \sum_{a \in A} \exp\{\mathcal{Q}((s,q),a)/\tau\}, \tag{4.7}$$

where the state-action value function $\mathcal{Q}$ is defined as follows:

$$\mathcal{Q}((s,q),a) = R((s,q),a) + \gamma\, \mathbf{E}_{(s',q') \sim \Delta(\cdot|(s,q),a)}[\mathcal{V}(s',q')],$$

and $\tau > 0$ is a user-specified temperature. If $\tau \to 0$, then mellowmax operator $\mathcal{T}$ recovers the operator $\max$.

Given optimal values $\mathcal{V}^*(s,q)$ for all $(s,q) \in S \times \mathcal{L}_k$, where $k < i$, we formulate the problem for level i as follows (similar to the linear programming formulation [34]):

$$\begin{aligned} \min_{\mathcal{V}} \quad & \sum_{(s,q) \in S \times \mathcal{L}_i} c(s,q)\, \mathcal{V}(s,q) \\ \text{s.t.} \quad & \mathcal{T}\,\mathcal{V}(s,q) - \mathcal{V}(s,q) \leq 0, \forall (s,q) \in S \times \mathcal{L}_i \end{aligned} \tag{4.8}$$

Recall that the set $\{c(s,q) \mid (s,q) \in S \times \mathcal{L}_i, c(s,q) > 0\}$ is termed as *state-relevance* weights. In problem (4.8), $\mathcal{V}(s',q')$ is to be solved if $q' \in \mathcal{L}_i$ or has been solved if $q' \in \mathcal{L}_k$ for some $k < i$.

The solution to the problem (4.8) is the optimal value function $\mathcal{V}^*(s,q)$ for all $(s,q) \in S \times \mathcal{L}_i$. We have the optimal state-action value function $\mathcal{Q}^*$ is defined as follows: for all $(s,q) \in S \times \mathcal{L}_i$,

$$\mathcal{Q}^*((s,q),a) = R((s,q),a) + \gamma\, \mathbf{E}_{(s',q') \sim \Delta(\cdot|((s,q),a)}[\mathcal{V}^*(s',q')],$$

and the corresponding optimal policy is defined as follows: for all $(s, q) \in S \times \mathcal{L}_i$, $a \in A$,

$$\pi^*(a \mid (s, q)) = \exp\{(\mathcal{Q}^*((s, q), a) - \mathcal{V}^*(s, q))/\tau\}.$$

We introduce a value function approximation for each $q \in X$ as follows: For each $q \in X$, the value function is approximated by $\mathcal{V}_{\theta_q} : S \to \mathbf{R}$, where $\theta_q \in \mathbf{R}^{|\theta_q|}$ is a parameter vector of length $|\theta_q|$. We use a linear function approximation of $V_{\theta_q}(s) = \sum_{k=1}^{|\theta_q|} \phi_{k,q}(s)\theta_{k,q} = \Phi_q\theta_q, \forall s \in S$, where $\phi_{k,q} \colon S \to \mathbf{R}$, and $1 \leq k \leq |\theta_q|$ are pre-selected basis functions.

Remark 3. A value iteration using the softmax Bellman operator finds a policy that maximizes a weighted sum of total rewards and the entropy of the policy [93]. When the value/reward is small, the total entropy of policies accumulated with the softmax Bellman operator overshadows the value given by the reward function. Thus, when the value $\mathcal{V}((s', q'); \theta_{q'})$ of the state to be reached is small, we scale this value by a constant α to avoid the value overshadowed problem. Given the nature of the *MaxProb* problem, with a reward of 1 being assigned when the LTL constraint is satisfied, we almost always need to amplify the reward to avoid the value overshadowed problem.

4.5 Case Study

We validate the algorithm in a motion planning problem under an scLTL specification in a grid world. In this example, we consider the set $\mathcal{AP}$ of the following atomic propositions:

- a: robot reaches region A.
- b: robot reaches region B.
- c: robot reaches region C.
- d: robot reaches region D.
- o: robot reaches obstacles.
- goal: robot reaches goal region.

This specification describes that the system needs to avoid obstacles and satisfy one of the following:

- System visits A, C, and goal sequentially.

- System visits B, D, and goal sequentially.

Regions A, B, C, D, obstacles, initial, and goal are shown in Figure 4.4. Given the set $\mathcal{AP}$ of atomic propositions, this task can be captured by the following specification: $\Diamond(((a \wedge (\neg b \,\mathsf{U}\, c)) \vee (b \wedge (\neg a \,\mathsf{U}\, d))) \wedge \Diamond \,\mathsf{goal} \wedge \Box \neg o)$, and the corresponding DFA is plotted in Figure 4.5. The partitions of meta-modes are shown in Figure 4.5, with different meta-modes being boxed in different styled rectangles. The task automaton is partitioned into four meta-modes $\{X_i \mid 0 \leq i \leq 3\}$, and each level set $\mathcal{L}_i$, for $0 \leq i \leq 3$, contains one meta-mode with the same index. The reward is defined as the following: the robot receives a reward of 60 (an amplified reward for avoiding value diminishing) if the trajectory satisfies the specification. In each state $s \in S$ and for the robot's different actions: heading up, heading down, heading left, heading right, the probability of arriving at the "correct" cell is $1 - 0.03 \times |N|$, and the probability of arriving a "wrong" cell is 0.03, where 0.03 is the randomness in the system and $|N|$ is the number of possible succeeding states. We surround the gird world with walls. If the system hits the wall, it will be bounced back and stay in its original cell. All the obstacles are sink states; that is, it stays there forever when a robot goes to an obstacle.

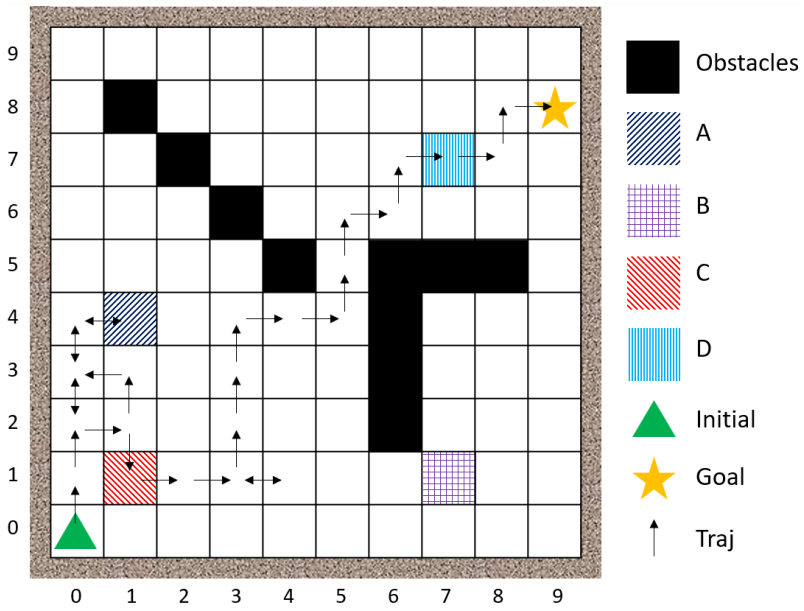


Figure 4.4: One simulation on the grid world.

The planning objective is to find an approximately optimal policy for satisfying the specification with a maximal probability. We compare the TADP with value iteration and topological value iteration to show the correctness and efficiency.

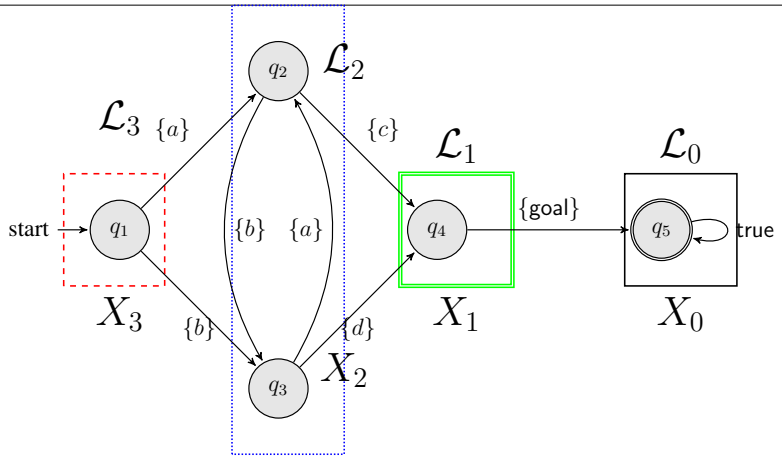


Figure 4.5: Automaton $\Diamond\left(((a \wedge (\neg b \,\mathsf{U}\, c)) \vee (b \wedge (\neg a \,\mathsf{U}\, d))) \wedge \Diamond \mathsf{goal} \wedge \Box \neg o\right)$, where meta-modes and the level sets are marked.

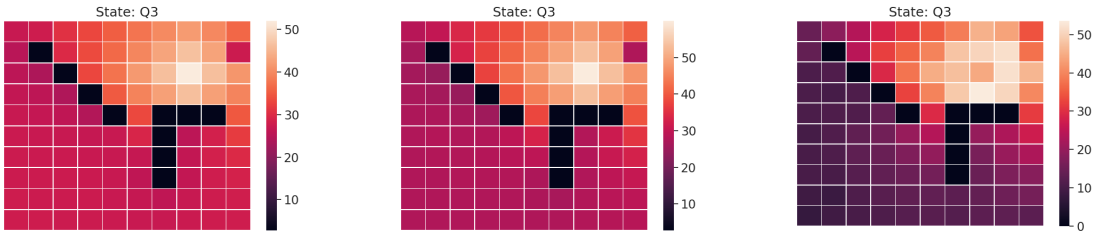
We use the following parameters: the user-specified temperature $\tau = 2$, discounting factor $\gamma = 0.9$, and error tolerance $\epsilon = 10^{-3}$. The tolerance is shared by topological value iteration, value iteration, and TADP, where the stopping criterion is $\max|\mathcal{V}^j - \mathcal{V}^{j-1}| \leq \epsilon$ for j-th iteration of in topological value iteration and value iteration and for each inner j-th iteration of TADP, respectively.

We adopt the following initial parameters in the TADP algorithm for the k-th problem: the coefficient of the penalty $b = 1.5$, the learning rate $\eta = 0.1$, the penalty parameter $\nu = 2.0$, and the Lagrangian multipliers $\lambda = 0$. During each inner iteration, we sample 30 trajectories of length ≤ 3. For each $s \in S$, the value function $\mathcal{V}_{\theta_q}(s)$ is approximated by a weighted sum of Gaussian Kernels: $\mathcal{V}_{\theta_q}(s) = \Phi_q \theta_q$, where basis functions $\Phi_q = [\phi_1, \phi_2, \ldots, \phi_{|\theta_q|}]$ are defined as the following: $\Phi_j(s) = K(s, c_j)$ and $K(s, s') = \exp(-\frac{SP(s,s')^2}{2\sigma^2})$, where $\{c_j \mid 1 \leq j \leq |\theta_q|\}$ is a set of pre-selected centers and $\sigma = 1$. In this example, we select the centers to be uniformly selected points with interval 1 within the grid world.

After the TADP converges, we obtain the policy from the converged value functions computed by TADP and simulate the system. We plot one system simulation in Figure 4.4. The system starts at the initial state s_0; then it visits region A, then region C, and eventually it visits the goal state goal.

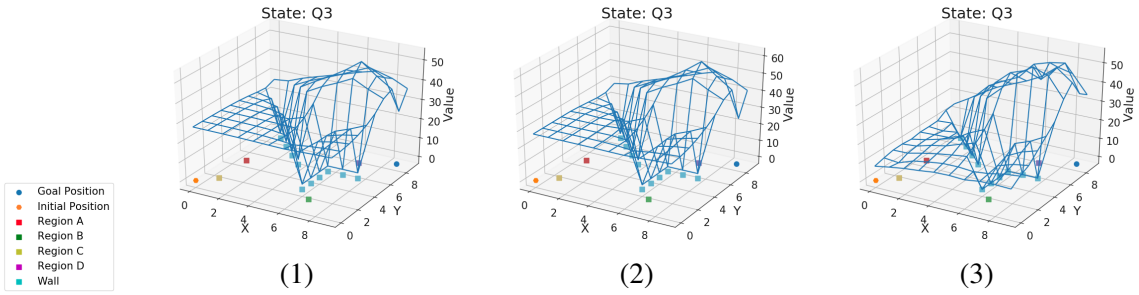
Figure 4.6 plots the heatmaps and values for different states at q_3 obtained by value iteration, topological value iteration, and TADP. In heatmaps, the brighter the area is, the higher value of that area is. Figures 4.61 and 4.62 show that value iteration and topological value iteration both show that the most bright area is at $[7, 7]$. In Figure 4.63, the area

around $[7, 7]$ obtained has a relatively bright color. The heatmap of TADP is not the same as the other two. This is due to the approximation error. Comparing three value surfs in Figures 4.71, 4.72, and 4.73, we can see the similarity between these three value surfs.



(1) (2) (3)

Figure 4.6: (1) The heatmap of $\mathcal{V}(\cdot, q_3)$ obtained by value iteration, topological value iteration, and TADP. (2) The heatmap of $\mathcal{V}(\cdot, q_3)$ obtained by topological value iteration. (3) The heatmap of $\mathcal{V}(\cdot, q_3)$ obtained by TADP.



(1) (2) (3)

Figure 4.7: (1) The value surf of $\mathcal{V}(\cdot, q_3)$ obtained by value iteration, topological value iteration, and TADP. (2) The value surf of $\mathcal{V}(\cdot, q_3)$ obtained by topological value iteration. (3) The value surf of $\mathcal{V}(\cdot, q_3)$ obtained by TADP.

We conduct two experiments for different sizes of grid worlds, *i.e.,* 10×10 and 20×20. We show the results in Table 4.1. In different sizes of gird worlds, comparing value iteration and topological value iteration, the runtime is reduced by 38.45% and 53.62% by exploiting the topological structure. The total numbers of Bellman Backup Operations are reduced by 7.71% and 7.76%. The decomposition occupies major CPU time in simple specifications, but exploiting the topological structure will be leveraged if more complex specifications are associated. The TADP converges after 135.96 seconds and 1117.91 seconds, respectively, for different sizes of grid worlds. The runtime of the topological value iteration and value iteration in a 20×20 grid world are 14-20 times their runtime in a 10×10 grid world. However, the runtime of TADP in a 20×20 grid world is only 8

times the runtime of TADP in the 10×10 grid world. TADP is more beneficial in large MDP problems or with more complex specifications. It is noted that though TADP takes, in general, a longer time to converge, it is scalable. Topological value iteration and value iteration do not scale.

	Algorithms	value iteration	topological value iteration	TADP
10×10	Bellman Backup Operations (times)	64620	59636	N/A
	Runtime (Seconds)	11.52	7.09	135.96
20×20	Bellman Backup Operations (times)	280620	258836	N/A
	Runtime (Seconds)	222.56	103.21	1117.59

Table 4.1: Bellman Backup Operations and runtime between value iteration, topological value iteration, and TADP. Note that topological value iteration has a significantly shorter runtime.

In Figure 4.8, we plot the convergence of values for different states in the 10×10 grid and states in automaton against epochs, which is the number of inner iterations in TADP. It indicates that the values initially oscillate, but all values converge after 250 iterations.

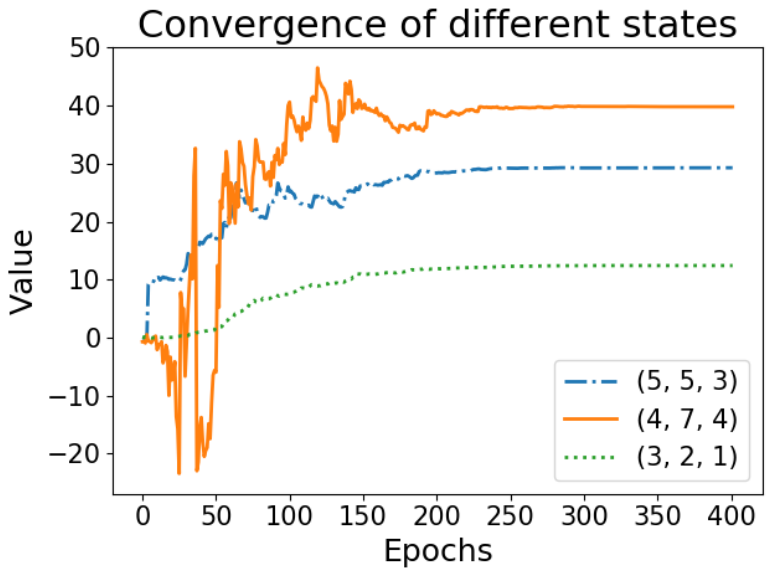


Figure 4.8: The convergence of values in TADP in the 10×10 stochastic grid world for different states in the product MDP. A product state $[5, 5, 3]$ means the grid cell $[5, 5]$ and the DFA state q_3.

We want to quantify and compare the performance of different methods. We update the policies from converged value functions computed by TADP and topological value iteration. We simulate trajectories 500 times and compare the percentage of trajectories

reaching the goal out of the total. We limit the max time-step to 500; if the system cannot reach the goal within 500 steps, then the system fails to reach the goal. Moreover, if the system reaches any sink states, then the system fails to reach the goal. Otherwise, the system successfully reaches the goal. We conduct two statistical experiments under the same setting with different starting potions, *i.e.,* $[1, 2, 2]$ and $[2, 2, 4]$. Starting with $[1, 2, 2]$, the percentages of reaching the goal under TADP and topological value iteration are 66.2% and 86.8%. Starting with $[2, 2, 4]$, the percentages of reaching the goal under TADP and topological value iteration are 80% and 88.6%. The results indicate that the policy computed by TADP is suboptimal due to the nature of ADP, but the performance gap between the two policies is insignificant.

4.6 Conclusion

We present a topological approximate dynamic programming method to maximize the probability of satisfying high-level system specifications in LTL. We decompose the product MDP and define the topological order for updating value functions at different task states to mitigate the sparse reward problems in RL with LTL objectives. The correctness of the algorithm is demonstrated on a robotic motion planning problem under LTL constraints. In the next chapter, we will consider using a neural network instead of linear function approximation and extend this method for solving continuous-state continuous-action stochastic systems under temporal logic constraints.

Chapter 5

Topological Order Guided Actor-Critic Modular Learning of Continuous Systems with Temporal Objectives

5.1 Overview

Chapter 4 has demonstrated that high-level specifications can define more sophisticated system behaviors, distinguished from a traditional A-to-B motion planning task. Providing probabilistic guarantees of high-level specifications brings significant value to a broad range of safety-critical applications, including robotics [78], military defense [15], cybersecurity [21], and other cyber-psychical systems [13]. One most encountered safety-critical example in robotics is that a robot avoids obstacles. Suppose we can quantitatively evaluate the probability of a robot running into obstacles. In that case, we can trade this probability for other decisive factors, such as cost. Despite the theoretical successes and state-of-art experimental results on formal policy synthesis, there are challenges when implemented in real-world applications: (1) availability of system models; *i.e.,* the system model is often unknown. (2) power of handling the continuous-state space; *i.e.,* we encounter continuous systems almost all the time, while most published results are demonstrated on discrete systems.

This chapter investigates the formal policy synthesis for continuous-state stochastic systems with high-level specifications expressed in LTL. Recall that LTL can succinctly

specify a collection of desired system properties, such as safety, liveness, persistence, and stability [83]. To learn an optimal policy that maximizes the probability of satisfying an LTL formula, we follow an automaton-based approach described in Chapter 4. We take a product between the stochastic system and the task automaton, where we translate the task automaton via existing LTL-automaton conversion [40]. When the system visits the final states, it satisfies the specification and receives positive rewards; However, since this system has a hybrid state space, entering the final states requires extremely high sampling complexity, raising the sparse reward issue. To mitigate this issue, we leverage the topological order proposed in Chapter 4 to guide the value backups and help accelerate the learning process. Topological order provides the structural information on the order of updating values of product states. Specifically, the system receives reward signals when it transits from the current level to the previous level whose values have converged and been optimal.

Further, we propose a model-free RL algorithm where topological order is applicable. This algorithm only requires paths of the stochastic process but not the system model. Our proposed RL algorithm is a variant of the actor-critic algorithm. It differs from the soft actor-critic (SAC) algorithm by the distinctive way of policy evaluation. We evaluate our policy by solving a sequential optimization problem and leveraging the augmented Lagrangian method for hyperparameter self-tuning. The proposed actor-critic algorithm alternates between: (a) policy evaluation and (b) policy improvement. We prove that our algorithm achieves optimality and convergence in a tabular case. We approximate value and policy functions with neural networks to tackle storing value functions and policy functions for hybrid product states. However, by assigning integers to denote automaton states, it ranks values of different automaton states by integers [133]. Instead, we approximate each automaton state's value (policy) functions by individual neural networks, termed modular learning. Specifically, if there are N automaton states at the current level, we use $2N$ neural networks to approximate values and policy functions.

Compared to minimizing the temporal error like temporal difference learning [119], our policy evaluation is inspired by the linear programming solution for MDP [34]. By using the mellowmax operator, we formulate a similar constrained optimization problem. However, as the system is continuous/hybrid, we transform the original constrained optimization problem into an equivalent stochastic constrained optimization. We only require the constraints to be satisfied on trajectories sampled in an off-policy manner. Further, to remove the system model in constraints, we adopt an unbiased estimate to approximate the

value of the current state by using only the current action, the next state, and the reward of execution for the trajectories. Similar to Chapter 4, we transform the inequality constraints into equality constraints by adopting the augmented Lagrangian method. We solve this unconstrained optimization problem sequentially, where sequential solving means solving a sequence of subproblems with a fixed set of hyperparameters. At the end of each subproblem, we update the hyperparameters. As the number of subproblems solved increases, solutions of value and policy functions improve. Note that we incorporate policy improvement into the sequential optimization.

This chapter's central theoretical contribution is to present a comprehensive, efficient formal policy synthesis framework for continuous-state stochastic systems with high-level specifications. Learning a control policy that maximizes the satisfaction probability of high-level specifications is genuinely intractable when the system model is continuous and unavailable. We present a sequential, actor-critic RL algorithm only requiring sampled trajectories interacting with the environment. The algorithm converges and achieves optimality in a tabular case. Using neural networks is the standard practice to save the trouble of storing values/policies in continuous/hybrid space. However, we approximate each value/policy function at each task state by one neural network to break the ordinal relationship between automaton states denoted by integers. We illustrate the empirical performance benefited from advanced mathematical methods by comparing our proposed RL algorithm with baselines. Further, we demonstrate the efficacy of our formal policy synthesis framework on motion planning of a Dubins car with a temporal specification.

5.2 Related Work

As motioned researchers enjoy [57]'s theoretical soundness for formal policy synthesis, a general RL algorithms for MDP mainly fall into two categories: (a) model-based methods (*e.g.,* value iteration, policy iteration, and linear programming [116]) and (b) model-free methods (*e.g.,* Q-learning [128], deep Q-learning (DQN) [90], deep deterministic policy gradient (DDPG) [80], and SAC [54]). An RL algorithm incorporates neural networks to approximate functions to avoid intractable system models and has made tremendous progress in planning for complex environments [53, 54, 90, 122]. Lillicrap *et al.* [80] first extended Q-learning to the continuous action domain. They adopted an off-policy update to minimize correlations between samples and a target Q network to produce a consistent target during temporal difference backups along with batch normalization [62]. Experiment

results demonstrate that the DDPG algorithm is competitive with algorithms with full knowledge of models. They also showed that DDPG could learn policies directly from raw pixel inputs. Haarnoja *et al.* [54] proposed an off-policy actor-critic deep refinement learning algorithm to maximize expected reward while maximizing entropy. By combing off-policy updates with a stable stochastic actor-citric formulation, this method conquered two major challenges: very high sample complexity and brittle convergence properties, which requires very fine tuning of hyperparameters.

5.3 Main Result

Recall from Section 4.4.2, given optimal values $\mathcal{V}^*(s,q)$ for all $(s,q) \in S \times \mathcal{L}_k$, where $k < i$, we formulate the problem for level i as follows:

$$\begin{aligned} \min_{\mathcal{V}} \quad & \sum_{(s,q)\in S\times\mathcal{L}_i} c(s,q)\,\mathcal{V}(s,q) \\ \text{s.t.} \quad & \mathcal{T}\,\mathcal{V}(s,q) - \mathcal{V}(s,q) \leq 0, \forall (s,q) \in S \times \mathcal{L}_i \end{aligned} \tag{5.1}$$

The corresponding optimal state-action value function $\mathcal{Q}^*$ is defined as follows: for all $(s,q) \in S \times \mathcal{L}_i$,

$$\mathcal{Q}^*((s,q),a) = R((s,q),a) + \gamma\, \mathbf{E}_{(s',q')\sim\Delta(\cdot|((s,q),a)}[\mathcal{V}^*(s',q')],$$

and the corresponding optimal policy is defined as follows: for all $(s,q) \in S \times \mathcal{L}_i$, $a \in A$,

$$\pi^*(a \mid (s,q)) = \exp\{(\mathcal{Q}^*((s,q),a) - \mathcal{V}^*(s,q))/\tau\}.$$

However, problem (5.1) poses constraints on every state in level i, which makes the computation intractable when the product state space is hybrid.

5.3.1 Sequential actor-critic RL

We use neural networks to approximate value function and policy function for level i to tackle with hybrid product state space. That is because maintaining value and policy for each product state at level i is intractable. Specifically, for all state $(s,q) \in S \times \mathcal{L}_i$, the approximate value function and policy function are denoted by $\mathcal{V}_\theta(s,q)$ and $\pi_\phi(\cdot \mid s,q)$, respectively, where θ and ϕ are the corresponding parameters to search. A product MDP in Definition 6 can be treated as an MDP augmented with an automaton state space.

Given such observation, for simplicity, we present our algorithm in the conventional MDP context.

Compared to a classic actor-critic algorithm, where the critic network and actor network share a common objective function, our proposed actor-critic algorithm provides a novel policy evaluation mechanism—a constrained optimization. By adopting such a formulation, we can leverage advanced mathematical techniques for hyperparameter self-tuning. Similar to the classic actor-critic algorithm, our proposed algorithm consists of two components: policy evaluation and policy improvement, and alternates between these two. We provide proof of the convergence and optimality of our algorithm.

Policy Evaluation

We define a mellowmax operator $\mathcal{T}^\pi$ for policy π: for any $s \in S$,

$$\mathcal{T}^\pi \mathcal{V}(s) = \sum_{a \in A} \pi(a \mid s)(\mathcal{Q}(s,a) - \tau \log \pi(a \mid s)).$$

It can be shown that the value function $\mathcal{V}^\pi$ is the solution for the following optimization problem:

$$\begin{aligned} \min_{\mathcal{V}} \quad & \sum_{s \in S} c(s)\, \mathcal{V}(s) \\ \text{s.t.} \quad & \mathcal{V}(s) \geq \mathcal{T}^\pi \mathcal{V}(s), \forall s \in S \end{aligned} \tag{5.2}$$

Lemma 9 (Convergence of Policy Evaluation (Extended from Lemma 1 in [34]))**.**

A value function $\mathcal{V}$ solves

$$\begin{aligned} \min_{\mathcal{V}} \quad & \sum_{s \in S} c(s)\, \mathcal{V}(s) \\ \text{s.t.} \quad & \mathcal{V}(s) \geq \mathcal{T}^\pi \mathcal{V}(s), \forall s \in S \end{aligned}$$

if and only if it solves

$$\begin{aligned} \min_{\mathcal{V}} \quad & \|\mathcal{V}^\pi - \mathcal{V}\|_{1,c} \\ \text{s.t.} \quad & \mathcal{V}(s) \geq \mathcal{T}^\pi \mathcal{V}(s), \forall s \in S \end{aligned}$$

Proof. The proof is similar to Lemma 1 in [34] with the replacement of operator $\mathcal{T}^\pi$. It is known that the mellowmax operator $\mathcal{T}^\pi$ is monotonic and contractive [113]. Given this and the fact that there is a fixed point $\mathcal{V}^\pi$ such that $\mathcal{V}^\pi(s) = \mathcal{T}^\pi\,\mathcal{V}^\pi(s), \forall s \in S$, it follows that for any $\mathcal{V}(s)$ with $\mathcal{V}(s) \geq \mathcal{T}^\pi\,\mathcal{V}(s)$, we have

$$\mathcal{V}(s) \geq \mathcal{T}^\pi\,\mathcal{V}(s) \geq \mathcal{T}^\pi\,\mathcal{T}^\pi\,\mathcal{V}(s) \geq \cdots \geq \mathcal{V}^\pi(s). \tag{5.3}$$

Hence, if solution $\mathcal{V}$ is a feasible solution to the optimization problem (5.2), then solution $\mathcal{V}$ satisfies $\mathcal{V}(s) \geq \mathcal{V}^\pi(s), \forall s \in S$. Further, we have

$$\min\|\mathcal{V}^\pi - \mathcal{V}\|_{1,c} = \min \sum_{s\in S} c(s)|\mathcal{V}(s) - \mathcal{V}^\pi(s)| \tag{5.4}$$

$$= \min \sum_{s\in S} c(s)\,\mathcal{V}(s) - \sum_{s\in S} c(s)\,\mathcal{V}^\pi(s) \tag{5.5}$$

Hence minimizing $\sum_{s\in S} c(s)\,\mathcal{V}(s)$ is equivalent to minimizing $\min\|\mathcal{V}^\pi - \mathcal{V}\|_{1,c}$. □

If we let $c(s)$ = $\mathbf{Pr}^\pi(s)$, for all $s \in S$, where $\mathbf{Pr}^\pi$ denotes the state marginals of the trajectory distribution induced by a policy π, then for any function $f\colon S \to \mathbf{R}$, the following holds:

$$\sum_{s\in S} \mathbf{Pr}^\pi(s) f(s) = \sum_{\rho\in \mathrm{PATH}^\pi} \mu^\pi(\rho) \sum_{t\geq 0} f(s_t) \tag{5.6}$$

$$= \mathbf{E}_{\rho\sim\mu^\pi}[\sum_{t\geq 0} f(s_t)],$$

where $\rho = s_0 a_0 s_1 a_1 \ldots \in \mathrm{PATH}^\pi$, and $\mu^\pi(\rho)$ is the probability of a path $\rho \in \mathrm{PATH}^\pi$. Intuitively, Equation (5.6) states that the expected sum of value on states visited by following policy π is equal to the expected sum of value over paths following policy π.

Given Equation (5.6) and replacing f with $\mathcal{V}$, the problem (5.2) becomes as follows:

$$\begin{aligned} \min_{\mathcal{V}} \quad & \mathbf{E}_{\rho\sim\mu^\pi}[\sum_{t\geq 0} \mathcal{V}(s_t)] \\ \text{s.t.} \quad & \mathcal{V}(s) \geq \mathcal{T}^\pi\,\mathcal{V}(s), \forall s \in S \end{aligned} \tag{5.7}$$

where $\rho = s_0 a_0 s_1 a_1 \ldots \in \mathrm{PATH}^\pi$.

We define a function $g\colon S \to \mathbf{R}$ such that

$$\begin{aligned} g(s) &= \mathcal{T}^\pi\,\mathcal{V}(s) - \mathcal{V}(s) \\ &= \sum_{a\in A} \pi(a \mid s)(\mathcal{Q}(s,a) - \tau \log \pi(a \mid s)) - \mathcal{V}(s) \\ &= \sum_{a\in A} \pi(a \mid s)(R(s,a) + \gamma\,\mathbf{E}_{s'\sim P(\cdot|s,a)}[\mathcal{V}(s')] - \tau \log \pi(a \mid s)) - \mathcal{V}(s). \end{aligned} \tag{5.8}$$

Given Equations g (5.8) and h (3.5), the problem (5.7) becomes as follows:

$$\begin{aligned} \min_{\mathcal{V}} \quad & \mathbf{E}_{\rho\sim\mu^\pi}[\sum_{t\geq 0}\mathcal{V}(s_t)] \\ \text{s.t.} \quad & h(g(s)) = 0, \forall s \in S \end{aligned} \tag{5.9}$$

where $\rho = s_0a_0s_1a_1\ldots \in \text{PATH}^\pi$.

Note that $g(s)$, for any $s \in S$, requires the knowledge of the transition probabilities for the expected value of the next state $\mathbf{E}_{s'\sim P(\cdot|s,a)}[\mathcal{V}(s')]$. However, we can replace $R(s,a) + \gamma\,\mathbf{E}_{s'\sim P(\cdot|s,a)}[\mathcal{V}(s')]$ with $R(s,a) + \gamma\,\mathcal{V}(s')$ that is an unbiased estimate [116]. We define the unbiased estimate $\tilde{g}\colon S \to \mathbf{R}$ as follows: for any state $s \in S$,

$$\begin{aligned} \tilde{g}(s) = & \sum_{a\in A}\pi(a \mid s)(R(s,a) + \gamma\,\mathcal{V}(s') - \tau\log\pi(a \mid s)) - \mathcal{V}(s), \\ & \text{where } s' \sim P(\cdot \mid s,a). \end{aligned} \tag{5.10}$$

For a continuous-state MDP, there are an infinite number of constraints in the problem (5.7). To relieve this, we only enforce on states visited by policy π and obtain a new equivalent problem as follows:

$$\begin{aligned} \min_{\mathcal{V}} \quad & \mathbf{E}_{\rho\sim\mu^\pi}[\sum_{t\geq 0}\mathcal{V}(s_t)] \\ \text{s.t.} \quad & \mathbf{E}_{\rho\sim\mu^\pi}[h(\tilde{g}(s_t))] = 0 \end{aligned} \tag{5.11}$$

where $\rho = s_0a_0s_1a_1\ldots \in \text{PATH}^\pi$.

The augmented Lagrange function of a given path $\rho = s_0a_0s_1a_1\cdots \in \text{PATH}^\pi$ is defined as follows:

$$\begin{aligned} \mathcal{L}(\rho,\vec{\lambda},\vec{\nu}) &= \sum_{t\geq 0}[\mathcal{V}(s_t)] + \sum_{t\geq 0}[\lambda_t(h(\tilde{g}(s_t)))] + \sum_{t\geq 0}[\frac{\nu_t}{2}h(\tilde{g}(s_t))^2] \\ &= \sum_{t\geq 0}[\mathcal{V}(s_t) + \lambda_t(h(\tilde{g}(s_t))) + \frac{\nu_t}{2}h(\tilde{g}(s_t))^2]. \end{aligned} \tag{5.12}$$

where $\rho = s_0a_0s_1a_1\ldots \in \text{PATH}^\pi$, and we let $\vec{\lambda} = [\lambda_i \mid i \geq 0]$ and $\vec{\nu} = [\nu_i \mid i \geq 0]$.

Given the augmented Lagrange function, problem (5.9) becomes as follows:

$$\min_{\mathcal{V}} \quad \mathbf{E}_{\rho\sim\mu^\pi}[\mathcal{L}(\rho,\vec{\lambda},\vec{\nu})] \tag{5.13}$$

We solve the problem (5.13) with sequential optimization techniques; sequential optimization solves a sequence of subproblems with corresponding fixed hyperparameters

$\{\vec{\lambda}_m \mid m \geq 0\}$ and $\{\vec{\nu}_m \mid m \geq 0\}$. For m-th subproblem, we aim to solve the following subproblem:

$$\min_{\mathcal{V}} \quad \mathbf{E}_{\rho\sim\mu^\pi}[\mathcal{L}(\rho, \vec{\lambda}_m, \vec{\nu}_m)] \tag{5.14}$$

Policy Improvement

Policy improvement is to minimize soft consistency error [93]. Formally:

$$\min_{\pi} \quad \mathbf{E}_{\rho\sim\mu^\pi}[\frac{1}{2}C(\rho)^2] \tag{5.15}$$

We define the soft consistency error of a finite path $\rho = s_0a_0s_1a_1\cdots s_T \in \mathrm{PATH}^\pi$ as follows:

$$C(\rho) = -\mathcal{V}^\pi(s_0) + \gamma^T \mathcal{V}^\pi(s_T) + \sum_{t=0}^{T-1} \gamma^t (R(s_t, a_t) - \tau \log \pi(a_t \mid s_t)). \tag{5.16}$$

It can be shown that $\pi = \pi^*$ when $J^\pi = 0$.

Lemma 10 (Consistency Implies Optimality [93])**.** If $\mathcal{V}$ and π satisfy, for all $(s, a) \in S \times A$:

$$\mathcal{V}(s) = R(s, a) + \gamma\, \mathbf{E}_{s'\sim P(\cdot|s,a)}[\mathcal{V}(s')] - \tau \log \pi(a \mid s),$$

then $\mathcal{V} = \mathcal{V}^*$ and $\pi = \pi^*$.

Policy Iteration

The actor-citric algorithm alternates between policy evaluation and improvement, and it will converge to the optimal policy in the tabular case [36]. Likewise, it can be shown that our proposed algorithm converges to the optimal value function and policy function.

Theorem 5 (Policy Iteration extended from [54])**.** Repeated application of police evaluation (5.9) and policy improvement (5.15) to any π converges to a policy π^* such that $\mathcal{V}^*(s, a) > \mathcal{V}^\pi(s, a)$ for all $\pi \in \Pi$ and all $(s, a) \in S \times A$.

Proof. The proof is similar to Theorem 1 in [54]. Let π_i be the policy at iteration i. By nature of the gradient descent with the appropriate step size, the sequence $\mathbf{E}_{\rho\sim\mu^{\pi_i}}[\frac{1}{2}C(\rho)^2]$ is monotonically decreasing. Since $\mathbf{E}_{\rho\sim\mu^{\pi_i}}[\frac{1}{2}C(\rho)^2]$ is lower bounded by zero, the sequence goes to zero. By Lemma 10, the consistency implies optimality; we get an optimal policy π^* such that $\mathcal{V}^*(s, a) > \mathcal{V}^\pi(s, a)$ for all $\pi \in \Pi$ and all $(s, a) \in S \times A$. □

However, we can only perform such an exact algorithm in the tabular case. We approximate the exact algorithm and propose a practical approximation of Algorithm 4 for continuous-state MDP.

Given an approximate value function $\mathcal{V}_\theta$ and approximate policy function π_ϕ, we rewrite Equation (5.10) as follows:

$$\tilde{g}_\theta^\phi(s) = \sum_{a \in A} \pi_\phi(a \mid s)(R(s,a) + \gamma\, \mathcal{V}_\theta(s') - \tau \log \pi_\phi(a \mid s)) - \mathcal{V}_\theta(s_t), \quad \text{where } s' \sim P(\cdot \mid s, a), \tag{5.17}$$

Plugging Equation (5.17) in Equation (5.12), the augmented Lagrange function becomes as follows:

$$\mathcal{L}_\theta^\phi(\rho, \vec{\lambda}, \vec{\nu}) = \sum_{t \geq 0} [\mathcal{V}_\theta(s_t) + \lambda_t(h(\tilde{g}_\theta^\phi(s_t))) + \frac{\nu_t}{2} h(\tilde{g}_\theta^\phi(s_t))^2]. \tag{5.18}$$

Correspondingly, m-th subproblem (5.14) becomes as follows:

$$\min_\theta \quad \mathbf{E}_{\rho \sim \mu^{\pi_\phi}}[\mathcal{L}_\theta^\phi(\rho, \vec{\lambda}_m, \vec{\nu}_m)] \tag{5.19}$$

We use gradient descent to update parameter θ as follows:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \, \mathbf{E}_{\rho \sim \mu^{\pi_{\phi_n}}}[\mathcal{L}_{\theta_n}^{\phi_n}(\rho, \vec{\lambda}_m, \vec{\nu}_m)], \tag{5.20}$$

where η is a user-specified learning rate.

Similarly, we approximate the soft consistency error of a finite path $\rho = s_0 a_0 s_1 \cdots s_T \in \mathrm{PATH}^{\pi_\phi}$ defined in Equation (5.16) as follows:

$$C_\theta^\phi(\rho) = -\mathcal{V}_\theta(s_0) + \gamma^T \mathcal{V}_\theta(s_T) + \sum_{t=0}^{T-1} \gamma^t (R(s_t, a_t) - \tau \log \pi_\phi(a_t \mid s_t)). \tag{5.21}$$

The problem (5.15) becomes as follows:

$$\min_\phi \quad \mathbf{E}_{\rho \sim \mu^{\pi_\phi}}[\frac{1}{2} C_\theta^\phi(\rho)^2] \tag{5.22}$$

For rotational connivance, we let $J_\theta^\phi = \mathbf{E}_{\rho \sim \mu^{\pi_\phi}}[\frac{1}{2} C_\theta^\phi(\rho)^2]$, and the updating rule for parameter ϕ is as follows:

$$\phi_{n+1} = \phi_n - \eta \nabla_\phi J_{\theta_n}^{\phi_n}, \tag{5.23}$$

where policy gradient for $J_{\theta_n}^{\phi_n}$ has the following form:

$$\nabla_\phi J_{\theta_n}^{\phi_n} = \mathbf{E}_{\rho\sim\mu^{\pi_{\phi_n}}}[C_{\phi_n}^{\theta_n}(\rho)\sum_{t=0}^{|\rho|-2}\gamma^t\nabla_\phi \log \pi_{\phi_n}(a_t \mid s_t)], \tag{5.24}$$

It is impossible to perform gradient descent in Equation (5.20) and (5.23) due to the expectation is over all trajectories; we approximate Equation (5.20) and (5.23) with a set of K trajectories $\{\rho_k \in \mathrm{PATH}^{\pi_{\phi_n}} \mid 1 \le k \le K\}$ as follows:

$$\theta_{n+1} = \theta_n - \eta\nabla_\theta \sum_{k=1}^{K}\frac{1}{K}\mathcal{L}_{\theta_n}^{\phi_n}(\rho_k, \vec{\lambda}_m, \vec{\nu}_m),$$
$$\phi_{n+1} = \phi_n - \eta \sum_{k=1}^{K}\frac{1}{K}C_{\phi_n}^{\theta_n}(\rho_k)\sum_{t=0}^{|\rho_k|-2}\gamma^t\nabla_\phi \log \pi_{\phi_n}(a_t \mid s_t).$$

Algorithm 4: Sequential Actor-Critic Algorithm

Output: Parameters θ_m, ϕ_m.

Initialization: $m = 0$, randomly initialize θ_0, ϕ_0, initialize a replay buffer.

while $m < M$ **do**
  Sample K trajectories $\{\rho_k \in \mathrm{PATH} \mid 1 \le k \le K\}$ from the replay buffer.
  $\mathsf{violation}_m = \sum_{k=1}^{K}\frac{1}{K}\sum_{t=0}^{|\rho_k|-1} h(\tilde{g}_{\theta_m}^{\phi_m}(s_t))$
  $\theta_{m+1}, \phi_{m+1} \leftarrow$ Solve subproblem $(\theta_m, \phi_m, \vec{\lambda}_m, \vec{\nu}_m)$
  Sample K trajectories $\{\rho_k \in \mathrm{PATH} \mid 1 \le k \le K\}$ from the replay buffer.
  $\mathsf{violation}_{m+1} = \sum_{k=1}^{K}\frac{1}{K}\sum_{t=0}^{|\rho_k|-1} h(\tilde{g}_{\theta_{m+1}}^{\phi_{m+1}}(s_t))$
  $\vec{\lambda}_{m+1} \leftarrow \vec{\lambda}_m + \vec{\nu}_m \cdot \sum_{t\ge 0} h(\tilde{g}_{\theta_{m+1}}^{\phi_{m+1}}(s_t))$
  $\vec{\nu}_{m+1} \leftarrow \begin{cases}\beta\vec{\nu}_m & \text{if } \mathsf{violation}_{m+1} > \epsilon\cdot\mathsf{violation}_m \\ \vec{\nu}_m & \text{otherwise.}\end{cases}$
  $m = m + 1$
end while
return $\mathcal{V}_{\theta_m}, \pi_{\phi_m}$

Let us briefly summarize the proposed actor-critic algorithm in Algorithms 4 and 5: Recall that we use sequential optimization to solve the policy evaluation, and the policy improvement adapts to that mechanism. The difference is that no dual variables for policy improvement need updating for the next subproblem. We start with any random parameters θ_0 and ϕ_0 for critic and actor networks. For m-th subproblem, we solve this problem

Algorithm 5: Solve subproblem $(\theta_0, \phi_0, \lambda, \nu)$

Input: Initial solution θ_0 and ϕ_0, dual variables $\vec{\lambda}$ and $\vec{\nu}$.
Output: Parameters θ_n and ϕ_n.
Initialization: $n = 0$.
1: **while** $n < N$ **do**
2: Sample a path ρ following the policy π_{θ_n} from env.
3: Add ρ into the replay buffer.
4: Sample K trajectories $\{\rho_k \in \text{PATH} \mid 1 \leq k \leq K\}$ from the replay buffer.
5: $\theta_{n+1} = \theta_n - \eta \nabla_\theta \sum_{k=1}^{K} \frac{1}{K} \mathcal{L}_{\theta_n}^{\phi_n}(\rho_k, \vec{\lambda}_m, \vec{\nu}_m)$
6: $\phi_{n+1} = \phi_n - \eta \sum_{k=1}^{K} \frac{1}{K} C_{\phi_n}^{\theta_n}(\rho_k) \sum_{t=0}^{|\rho_k|-2} \gamma^t \nabla_\phi \log \pi_{\phi_n}(a_t \mid s_t)$
7: $n = n + 1$
8: **end while**
9: **return** θ_n, ϕ_n

with fixed dual variables $\vec{\lambda}_m$ and $\vec{\nu}_m$. Then we update dual variables in Line 7 and 8 in Algorithm 4, where β is the growth rate of the penalty term $\vec{\nu}_m$, and ϵ is the performance threshold. Inside m-th subproblem, for n-th iteration, we sample a trajectory by following the current policy, then perform one policy evaluation and improvement.

5.3.2 Modular Learning: One Neural Network Per Task State

However, we observe that it is empirically challenging to train value function $\mathcal{V}_\theta$ and policy function π_ϕ since a neural network can assume an ordinal relationship between automaton states. By assigning integer numbers to automaton states, value or policy function approximated by a single neural network can be ranked by integer numbers.

To break this ordinal relationship, we approximate value function and policy function in one automaton state per neural network, termed modular learning. That is, instead of using one single neural network to approximate value function $\mathcal{V}(s, q)$ (resp. policy function $\pi(\cdot \mid s, q)$) denoted by $\mathcal{V}_\theta(s, q)$ (resp. $\pi_\phi(\cdot \mid s, q)$) for all $(s, q) \in S \times \mathcal{L}_i$, we use $|\mathcal{L}_i|$ neural networks to approximate, where $|\mathcal{L}_i|$ denotes the number of the automaton states in level $\mathcal{L}_i$. For each automaton state $q \in \mathcal{L}_i$, we denote the corresponding approximate value function as $\mathcal{V}_{\theta_q}(s)$ (reps. policy function as $\pi_{\phi_q}(s)$). Given $2|\mathcal{L}_i|$ neural networks ($|\mathcal{L}_i|$ for value functions and $|\mathcal{L}_i|$ for policy functions), we switch neural networks for different automaton states. We observe that adopting modular learning does not affect the usage of

the topological order.

5.4 Case Study

We evaluate our proposed RL algorithm on a classic control task, CartPole, and compare the performance with different baselines, including Proximal Policy Optimization (PPO) [108], DQN [89], and Advantage Actor Critic (A2C) [88]. We find that our RL algorithm matches or beats the performance of these baselines. Further, we demonstrate the efficacy of our proposed policy synthesis framework on a robot motion planning example with a high-level specification, where the robotic platform is a Traxxas, the Slash 4×4 Platinum Edition in Figure 5.1. For more details about the RC car platform, readers are referred to [7].

Figure 5.1: The RC car platform.

5.4.1 Performance Benchmark: CartPole-v1

We leverage OpenAI gym [22] for providing the classic control example, CartPole-v1 [9]. In CartPole, the lower end of the pole is mounted to a passive joint of a cart that moves along a frictionless track. The pole can only swing in a vertical plane parallel to the direction of the cart. Two actions: push back and push forward, can be applied to the cart to balance the pole. An episode starts with the pendulum being upright and ends with one of the following situations:

- Pole is more than 15 degrees from vertical;
- Cart moves more than 2.4 units from the center;
- The length of the episode reaches a maximum length of 500.

A reward of 1 is received for every time step that the pole remains upright. CartPole example aims to design a controller that prevents the pole from falling. After finding the best hyperparameters (see Table 5.1 and 5.2), we run our proposed algorithm, PPO, DQN, and A2C, independently five times (with randomly selected seeds). The average length of episodes versus training steps is plotted in Figure 5.2. Figure 5.2 induces that in CartPole, our proposed algorithm matches or defeats the performance of PPO, A2C, and DQN.

Table 5.1: Shared hyperparameters.

Parameter	Symbols	Value
Learning rate	η	$3 \cdot 10^{-4}$
Discounting factor	γ	0.99
Number of layers		2
Number of hidden units per layer		256

Table 5.2: Environment-specific hyperparameters.

Parameter	Symbols	CartPole-v1	Sequential Visiting
User-specified temperature	τ	1	0.5
Dual variable	λ	10^4	10^3
Penalty term	ν	10^5	10^5
Penalty growth rate	β	2	2
Max outer iteration	M	4	3
Max inner iteration	N	2500	1500
Performance threshold	ϵ	0.9	0.9
Length of sampled trajectory	T	10	10
Number of trajectories	K	10	5
Replay buffer size		10^4	10^4
Decay of learning rate		1	0.5
Decay steps			10^3

Note that in practice, there is no need for $|\vec{\lambda}|$ (resp. $|\vec{\nu}|$) dual variables; Instead, all λ_t (resp. ν_t) can be the same for all $t \geq 0$, and we can use λ (resp. ν) to denote $\vec{\lambda}$ (reps. $\vec{\nu}$).

To demonstrate the convergence of our proposed algorithm, we plot the value function of the initial state, loss of critic network, loss of actor network, and evaluation of the constraint $\mathbf{E}_{\rho\sim\mu^\pi}[h(\tilde{g}(s_t))]$ versus training steps in Figure 5.3. Figure 5.31 suggests our value function of the initial state converges after 6×10^4 steps. Figure. 5.32 and Figure. 5.33 suggest 5×10^4 steps is a critical training step point, where losses of actor and critic networks are close to zero. Near-zero losses of actor and critic networks match the

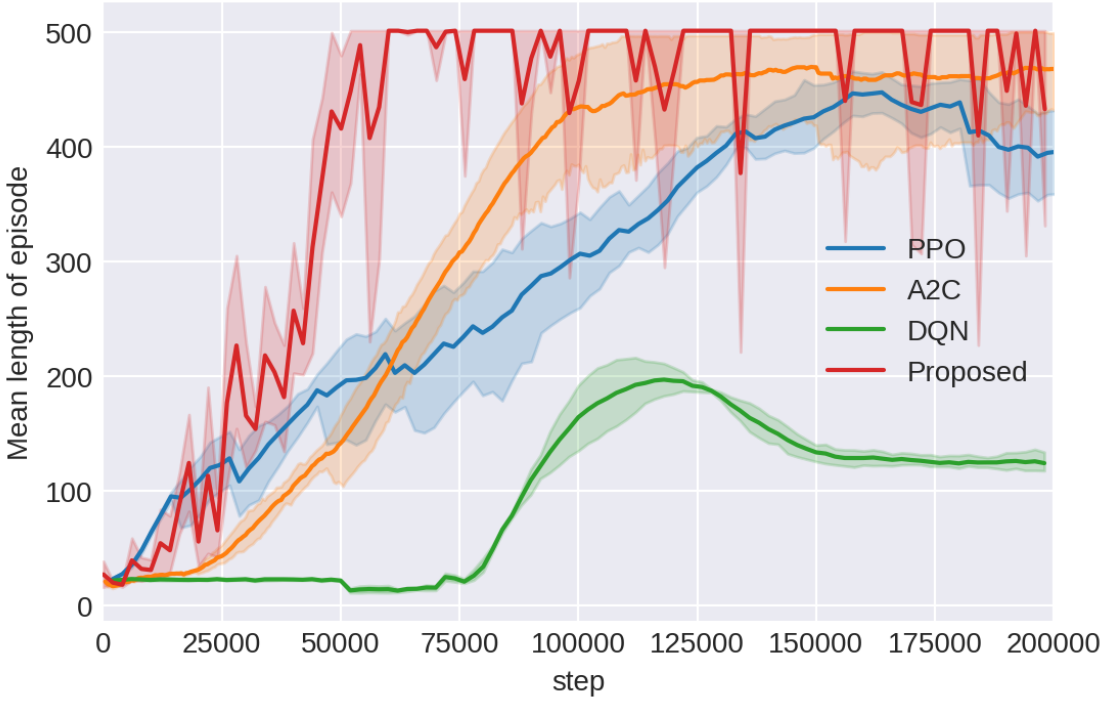


Figure 5.2: Performance of different baselines and our proposed algorithm on the CartPole-v1 benchmark.

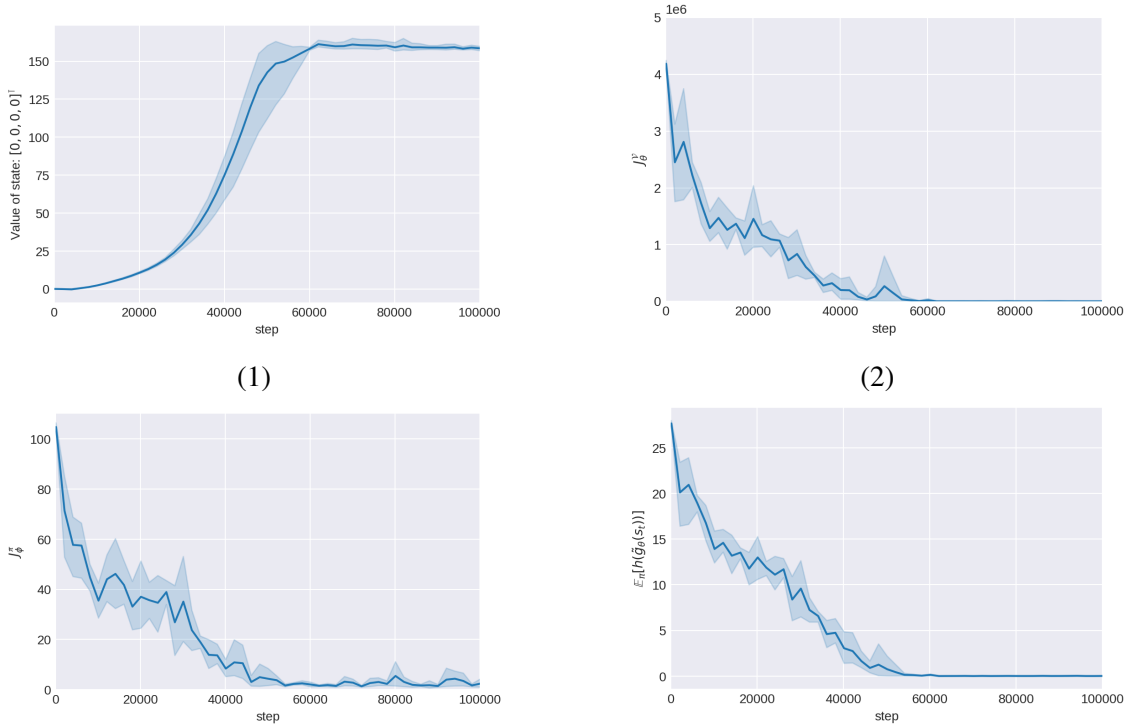
step point, where the length of episodes is close to the maximum length of 500, and the evaluation of the constraint decreases to zero, shown in Figure 5.2 and Figure 5.34, respectively.

5.4.2 Robot Motion Planning with a High-level Specification

We are interested in using the proposed algorithm to learn a policy in robot motion planning example with a high-level specification from Example 1, sequential visiting task. Recall that the goal is to maximize the probability that a car avoids obstacles and completes one of the following:

- car visits A and does not visit D or obstacles until C is visited;
- car visits D and does not visit A or obstacles until B is visited.

The RC car travels within a workspace in Figure 5.4, where A, B, C, and D are regions of interest, and black rectangles are obstacles.



(1) (2)

(3) (4)

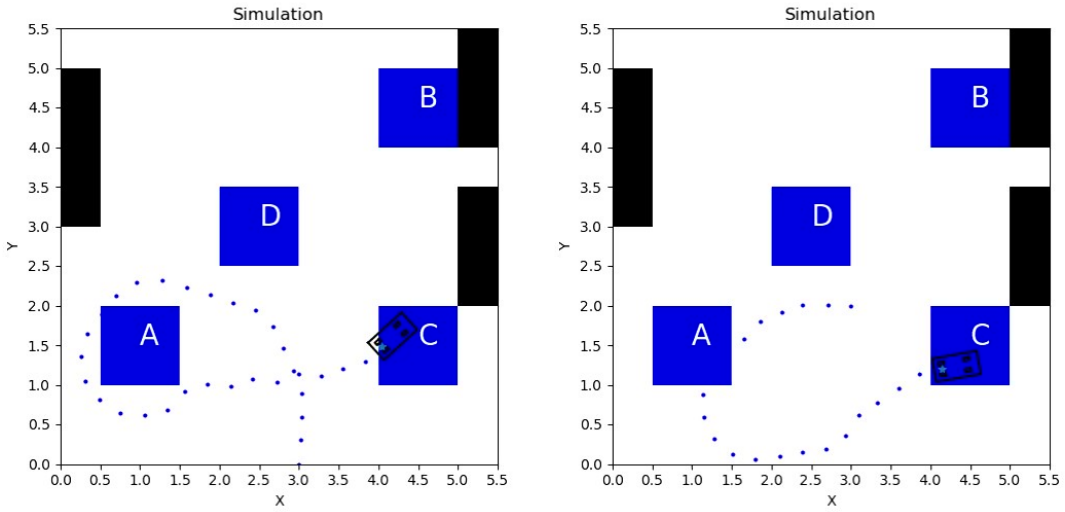
Figure 5.3: (1) Value of the initial state $\mathcal{V}(s_0)$ versus training steps, where $s_0 = [0, 0, 0, 0]$. (2) Loss of critic network $J^{\mathcal{V}} = \mathbf{E}_{\rho\sim\mu^\pi}[\mathcal{L}(\rho, \vec{\lambda}, \vec{\nu})]$ versus training steps. (3) Loss of actor network $J^\pi = \mathbf{E}_{\rho\sim\mu^\pi}[\frac{1}{2}C(\rho)^2]$ versus training steps. (4) Evaluation of constraint $\mathbf{E}_{\rho\sim\mu^\pi}[h(\tilde{g}(s_t))]$ versus training steps.

We model the RC car as a Dubins car model. Let us briefly recall the dynamics of the Dubins car defined as follows:

$$\begin{aligned}\dot{x} &= v \cdot \cos\theta,\\ \dot{y} &= v \cdot \sin\theta,\\ \dot{\theta} &= u,\end{aligned}$$

where $[x, y]$ denotes the car's position, θ is the heading, the car is moving at a constant speed $v = 0.3$ m/s, and the turn rate control $u \in \{-2\pi/15, 0, 2\pi/15\}$. The car starts at the initial state $[3, 0, \pi/2]$. For notational convenience, we use $\vec{z} = [x, y, \theta]^\intercal$ and $\dot{\vec{z}} = [\dot{x}, \dot{y}, \dot{\theta}]^\intercal$ to denote the car's state and velocity. Further, we let $\vec{z}_t$ and $\dot{\vec{z}}_t$ be the car's state and velocity at time t, respectively. We capture the dynamic evolution of the car as follows:

$$\vec{z}_{t+\Delta t} = \vec{z}_t + \Delta\vec{z}_{noise} + \dot{\vec{z}} \cdot \Delta t,$$



(1) (2)

Figure 5.4: (1) A simulated trajectory starting from an initial state $[3, 0, \pi/2]$. (2) A simulated trajectory starting from a different initial state $[3, 2, -\pi]$ that is different from the initial state $[3, 0, \pi/2]$ during training.

where Δt is the user-specified time unit, and $\Delta\vec{z}_{noise}$ is the noise. In our example, we have $\Delta t = 1$ s, and $\Delta\vec{z}_{noise}$ is the white noise with a standard deviation 10^{-2}.

Typically, we reward 1 when the car completes the specification and 0 otherwise. However, small rewards can be overshadowed when the entropy of policies is too large using the mellowmax operators. We amplify rewards as follows: A reward of 10 is given when the car completes the specification; A reward of -1 is given whenever the car goes out of the workspace or hits obstacles. Furthermore, to address the sparse reward issue, we define the reward signal as follows:

$$r(\vec{z}, \vec{z}_{\text{subgoal}}) = \vec{z} \cdot \frac{5\vec{d}}{|\vec{d}|}, \tag{5.25}$$

where $\vec{z}$ is the current state, $\vec{z}_{\text{subgoal}}$ is the current subgoal for the current automaton state, and $\vec{d} = [x_{\text{subgoal}} - x, y_{\text{subgoal}} - y]$. We define the current subgoal for each automaton state as follows:

$$\vec{z}_{\text{subgoal}} = \begin{cases} [1.25, 1.25], \text{ if } q = q_0, \\ [4.25, 4.25], \text{ if } q = q_1, \\ [4.25, 1.25], \text{ if } q = q_2. \end{cases} \tag{5.26}$$

Note that we do not define the subgoal for q_3 since the car has satisfied the specification. To demonstrate our trained policy can handle different initial states, we sample two trajectories from two initial states: (a) $[3, 0, \pi/2]$ is the same as the one during training, and (b) $[3, 2, -\pi]$ is different from the one during training, and plot them in Figure 5.4.

Table 5.3: Success Rates of Sequential Visiting Task.

Description	Success Rate
Single neural network	26%
Modular learning	49%
Modular learning + topological order	71.5%

We propose topological order to address the reward sparsity and modular learning to break down the ordinal relation between automaton states. To demonstrate the efficacy of both techniques, we compare our proposed method with a single neural network, modular learning, and modular learning with topological order. We plot corresponding values of the initial state in Figure 5.5 and list the success rates of the sequential visiting task (over 200 simulations) in Table 5.3. In the case of a single neural network for the entire product MDP, the input of the neural network is a 4-dimension vector, where the first 3 elements are the car's state, and the last element is the automaton state. For modular learning, the input of the neural network is only the car's state.

As listed in Tab 5.3, implementing modular learning increases the success rate dramatically, indicating the ordinal relationship is disrupted and provides better approximations for value and policy function. Given the same training steps, the success rate with the topological order is much higher than the one without the topological order. The above observation from Tab 5.3 matches the result in Figure 5.5, where the value of modular learning with topological order has the highest value and fastest convergence. Theoretically, the one with the topological order shall at least perform as well as the one without the topological order. Such improvement in the performance demonstrates that the topological order can guide value backups and accelerate the learning process.

We provide a video [1]demonstrating the success of learning a controller that satisfies the specification. The car uses AprilTags in the Tag36h11 set as fiducial markers for localization [95].

We run our algorithms on an Ubuntu 20.04 machine with AMD Ryzen 9 5900X CPU, 32 GB RAM, and NVIDIA GeForce RTX 3060. The computational time of computing a policy in CartPole is about 4 hours, and the computational times of computing a policy in

[1]`https://tinyurl.com/3dcysrux`

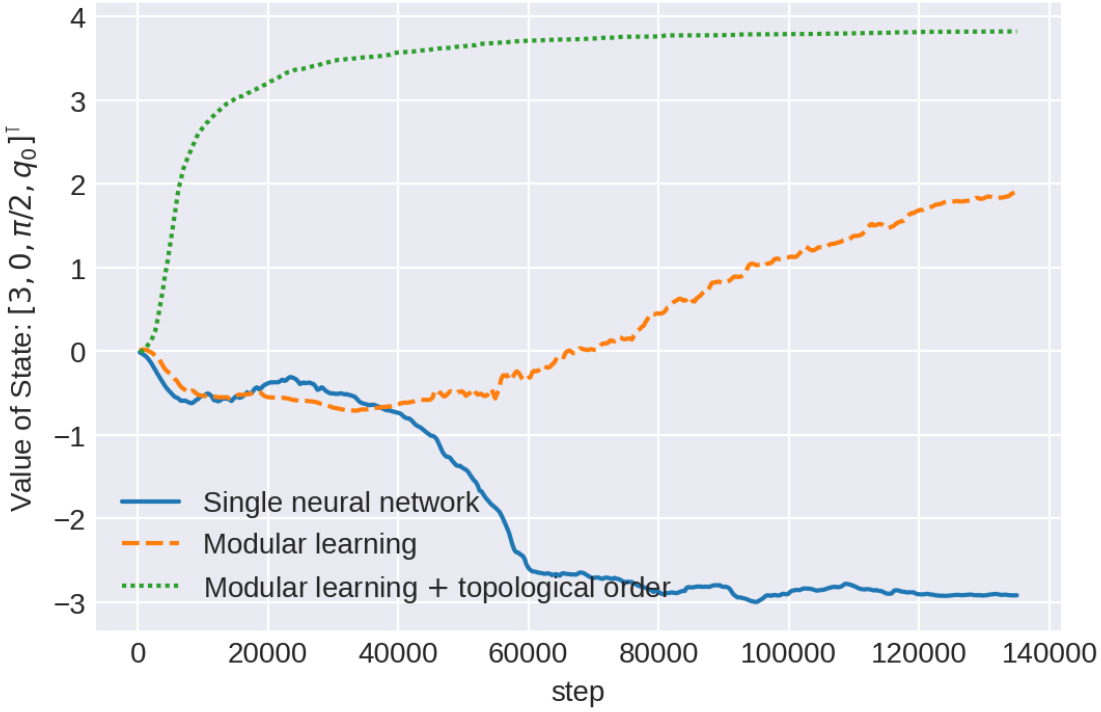


Figure 5.5: Values of initial state $\mathcal{V}(z_0)$ versus training steps, where $z_0 = [3, 0, \pi/2, q_0]$.

Dubins car's environment are about 17 min, 23 min, and 53 min for single neural network, modular learning, and modular learning with topological order, respectively. We find that the long computational time for CartPole is because of trajectory simulations for computing the mean length of the episode. [2] The reason for the computational time of modular learning with topological order is 2 times these of modular learning and single neural network is because, in our example, we have the set of level sets $\{\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2\}$, where the value of any state $(s, q) \in S \times \mathcal{L}_0$ equals to 0. So we need to learn values for $\mathcal{L}_1$ and $\mathcal{L}_2$.

5.5 Conclusion

This chapter proposes a comprehensive formal policy synthesis framework for continuous-state stochastic systems with high-level specifications. We apply the topological order to overcome the reward sparsity. We present a sequential, actor-critic RL algorithm to overcome the continuous/hybrid state space, where topological order still applies. We

[2] As the policy converges to the optimal policy, the length of the episode converges to maximum length 500, although we only consider T steps of a trajectory.

provide proof of optimality and convergence of this RL algorithm in a tabular case. We further use modular learning to prevent the approximate value/policy function from being ranked by assigning integer numbers to automaton states. Our proposed algorithm matches or beats baselines in CartPole. We demonstrate the efficacy of our policy synthesis framework on a Dubins car with a high-level specification, where a video validates the success of learning a controller that satisfies the specification. The results suggest that the topological order can relieve the sparse reward issue, and modular learning can break the ordinal relationship between the automaton states.

Chapter 6

Dynamic Hypergames for Synthesis of Deceptive Strategies with Temporal Logic Objectives

6.1 Overview

Planning in adversarial environments is commonly encountered in security and defense applications. In many such applications, the objective of the agent is partially known to its opponent. Thus, using deception to exploit this incomplete knowledge of the opponent becomes inseparable from strategic planning. Deception has been investigated in economics [51], military operations [55], cybersecurity [2, 58, 60, 114], and planning of robotics and other cyber-physical systems [41, 49, 85, 96, 125].

This chapter investigates the synthesis of deceptive strategies for a class of adversarial interactions with asymmetric information and Boolean payoffs in temporal logic [83]. Specifically, we consider an adversarial two-player interaction in which an agent (player 1/P1) aims to satisfy a task specified in a subclass of temporal logic [67] in a stochastic environment, and its adversary (player 2/P2), who has incomplete knowledge about P1's task, aims to prevent P1 from achieving the task. The temporal logic formulas express P1's complex and temporally extended goals. Thus, the key question of interest is, how can player 1 exploit player 2's incomplete information about the task to improve player 1's performance?

We address this question using a game-theoretic approach. In literature, deceptive planning has been mainly studied using two models of games with asymmetrical infor-

mation: Bayesian games [60, 134] and hypergames [44, 59]. Bayesian games model a player's incomplete information as a probability distribution over a set of types for its opponent, and the ground truth type is the private information of the opponent. The authors [60] utilize dynamic Bayesian games for defensive planning in cyber security disciplines, where the type of opponent, *i.e.,* a legitimate user or an attacker, is the private information. Another class of incomplete information games, called hypergames [14, 127], models multi-player interaction where each player holds a subjective viewpoint of the game and makes decisions based on this viewpoint. The hypergame exploits its opponent's unawareness of a player to deceive, unlike Bayesian games, which assume that all players have the same knowledge about all types. However, solution concepts exist only for normal-form hypergames, where players move simultaneously and receive real-valued payoffs for the outcomes of their actions. In [49], the authors investigate how a player's belief in other players' preferences evolves by observing other players' decisions and analyzing the inconsistency in the equilibrium. They develop stealthy deception [50] to restrict the deceiver's actions so as not to contradict the belief of the deceivee. Besides Bayesian games and hypergames, there have been investigations on deception in cyber security [107, 120] using security games [112], where the defender uses decoys to "mask" the network configuration and creates disinformation of payoffs. Additional to game-theoretic approaches, deceptive planning is developed in [86, 96], where the deceiver hides its objective from the observer to achieve the goal. However, these cases have no dynamic interaction between the deceiver and the observer.

Different from existing work, we study a class of games where players' payoffs are temporal logic formulas, instead of real-valued utility functions. This study is motivated by the need to express complex, temporal objectives in multi-stage interaction. Games with temporal logic payoffs, also known as ω-regular games, have been studied to synthesize reactive programs. ω-regular games are played on a finite state space for an infinite number of rounds. At each round, players move simultaneously or in a turn-based manner, and the next state (or a distribution of the next state) is determined by the current state and their joint actions. The ω-regular winning conditions of players are expressed in the resulting infinite state sequences. Existing solution concepts for ω-regular games studied games with complete information and perfect/partial observations [19, 25, 33, 98] but not for games with asymmetric, incomplete information.

To this end, we present a modeling framework and synthesis methods for deceptive planning in a subclass of ω-regular games with a hierarchical information pattern; that is,

P1 knows that P2 does not have complete information about P1's task. We introduce a class of hypergames with temporal logic objectives. The deceptive planning algorithm is based on a solution concept of hypergames, called subjectively rationalizable strategies. Based on this solution, two key modules, namely, *opponent modeling* and *deceptive planning*, are developed. Since a temporal logic formula describes a sequence of temporally extended subgoals, we assume that the adversary can infer the current agent's subgoal based on observations. In opponent modeling, the agent maintains a model of the adversary's subjectively rationalizable strategy and subgoal inference. Using the opponent model, the agent can predict how its action will influence the perception and strategy of the adversary. Then, it integrates the predictive opponent model into deceptive planning that computes a strategy to maximize the probability of satisfying its temporal logic objective. The agent's strategy is deceptive because it intentionally steers the adversary's perception of its objective and thus influences the adversary's response strategy in a way beneficial to the agent's task performance. We show the effectiveness of the proposed deceptive planning algorithm using robot motion planning examples. Finally, noting that the effectiveness of the deceptive strategy hinges on the matching between the agent's opponent model and the true opponent. To ensure the effectiveness of deception, we also design an online detection algorithm to identify potential errors in the opponent model.

The remainder of this chapter is organized as follows. Section 6.3 provides some necessary background on game theory. Section 6.4 presents the main theory and algorithms for deceptive planning. Section 6.5 presents a case study to demonstrate the effectiveness of the deception. Finally, Section 6.6 concludes and discusses future work.

6.2 Related Work

In literature, deceptive planning has been mainly studied using two models of games with asymmetrical information: Bayesian games [60, 134] and hypergames [44, 59]. Bayesian games model a player's incomplete information as a probability distribution over a set of types for its opponent, and the true type is the private information of the opponent. The authors [60] adopted dynamic Bayesian games to defensive planning for cyber security, where the type of the opponent, *i.e.,* , a legitimate user or an attacker, was the private information. Another class of incomplete information games, called hypergames [14, 127], models multi-player interaction where each player holds a subjective viewpoint of the game and makes decisions based on this viewpoint. In contrast to Bayesian games, hypergames

do not assume that all players share common knowledge about all possible types; this allows hypergames to explicitly capture the unawareness of a player, which could be exploited by its adversary using deception. However, existing solution concepts for hypergames are developed for normal-form hypergames, in which players move simultaneously and receive real-valued payoffs for the outcomes resulted from the combination of actions played. In [49], the authors studied how a player's belief of other players' preferences evolves by observing other players' decisions and analyzing the inconsistency in the equilibrium. They developed stealthy deception [50] to restrict the deceiver's actions so as not to contradict the belief of the deceivee. Besides Bayesian games and hypergames, deception in cyber security [107, 120] has been investigated for security games [112] where the defender uses decoys to "mask" the network configuration and creates disinformation of payoffs. Besides game-theoretic approaches, deceptive planning was developed in [86, 96] when the deceiver hides its objective from the observer for achieving the goal. However, in these cases there is no dynamic interaction between the deceiver and the observer.

6.3 Preliminaries: Omega-regular Games

We consider an adversarial encounter between two players: a controllable player P1 and an uncontrollable player P2. Both players choose their moves simultaneously. The dynamics of their interaction can be captured as a transition system with simultaneous moves.

Definition 15 (Two-player Transition System with Simultaneous Moves)**.**
A *two-player transition system with simultaneous moves* is a tuple

$$TS = \langle S, A, P, s_0, \mathcal{AP}, L \rangle$$

consisting of the following components:

- S is a finite set of states;
- $A = A_1 \times A_2$ is a finite set of actions, where A_1 is the set of actions that P1 can perform, and A_2 is the set of actions that P2 can perform;
- $P\colon S \times A \times S \to [0, 1]$ is a probabilistic transition function. At every state $s \in S$, P1 chooses an action $a^1 \in A_1$, and P2 chooses an action $a^2 \in A_2$, simultaneously. Then, a successor state s' is determined by the probability distribution $P(\cdot \mid s, a)$, where $a = (a^1, a^2) \in A$;

- s_0 is an initial state;
- $\mathcal{AP}$ is a set of atomic propositions;
- $L\colon S \to 2^{\mathcal{AP}}$ is a labeling function. For every state $s \in S$, the label $L(s)$ of the state s represents a set of atomic propositions evaluated true at the state s.

In this section, unless otherwise noted, we will refer to a two-player transition system with simultaneous moves simply as a transition system.

We use LTL formulas to represent the players' objectives/payoffs in the game. Similarly, a path ρ in a transition system TS is said to satisfy an LTL formula φ, if the labeling sequence $L(\rho)$ satisfies the formula φ, *i.e.,* $L(\rho) \models \varphi$. Given this relation, we define a zero-sum game with players' payoffs expressed in temporal logic, also known as a zero-sum ω-regular game.

Definition 16 (Zero-sum ω-regular Game)**.**
Given an LTL formula φ_1 and a two-player transition system with simultaneous move TS, a zero-sum ω-regular game, where P1's objective is φ_1, and P2's objective is $\varphi_2 = \neg\varphi_1$, is the tuple

$$\mathcal{G}(\varphi_1, \neg\varphi_1) = \langle TS, \varphi_1, \neg\varphi_1 \rangle.$$

In the interaction between P1 and P2, P1 maximizes the probability of a path satisfying the temporal logic formula φ_1; P2 minimizes the probability of a path satisfying the formula φ_1. Thus, P2's objective is the logical negation of P1's objective. We omit P2's objective $\neg\varphi_1$ from $\mathcal{G}(\varphi_1, \neg\varphi_1)$ and simply denote the game as $\mathcal{G}(\varphi_1)$.

A *play* in the game is constructed as follows: The players start in the initial game state s_0, simultaneously select a pair of actions $a = (a^1, a^2) \in A$, move to the next state s_1, and repeat. The game ends when one of the players satisfies its objective. Thus, a play is a sequence of states and actions, denoted by $\rho = s_0 a_0 s_1 a_1 \cdots$ such that $P(s_{i+1} \mid s_i, a_i) > 0$ for any $i \geq 0$. The set of all possible plays in the game is denoted by Plays. Slightly abusing the notation, given a play $\rho \in \mathsf{Plays}$, we say that $\rho \models \varphi$ for an LTL formula φ if $L(\rho) \models \varphi$, that is, the sequence of state labels satisfies the LTL formula φ. A play $\rho \in \mathsf{Plays}$ of TS is winning for player i if and only if $L(\rho) \models \varphi_i$ — that is, the labeling of that path satisfies player i's Boolean objective in temporal logic. The set of prefixes of plays is denoted by $\mathsf{PrefPlays}$. We refer to $h \in \mathsf{PrefPlays}$ as a history in the game.

A stochastic strategy $\pi_i\colon \mathsf{PrefPlays} \times A_i \to [0, 1]$, for player $i \in \{1, 2\}$, is a function that assigns a probability distribution over all actions given a history. Let Π_i denote the

stochastic strategy space of player i. A strategy profile $\langle \pi_1, \pi_2 \rangle$ is a pair of strategies, one for each player. A strategy profile $\langle \pi_1, \pi_2 \rangle$ induces a probability measure $\mathbf{Pr}^{\langle \pi_1, \pi_2 \rangle}$ over $\mathsf{PrefPlays}$.

Given player i's Boolean objective φ_i, we define the utility function for player i as $u_i \colon \mathsf{PrefPlays} \times \Pi_i \times \Pi_j \times \Phi \to \mathbf{R}$, such that for $(i,j) \in \{(1,2),(2,1)\}$, $u_i(h, \pi_i, \pi_j, \varphi_i) = \mathbf{Pr}^{\langle \pi_i, \pi_j \rangle}(hh' \models \varphi_i)$ is the probability of satisfying the specification φ_i, where $h \in \mathsf{PrefPlays}$ is the initial history, and h' is the stochastic process induced by the strategy profile $\langle \pi_i, \pi_j \rangle$ after the initial history h.

We present the definition of Nash equilibrium for ω-regular games with complete information.

Definition 17 (Nash equilibrium [20, 97])**.**
A Nash equilibrium of a ω-regular game $\mathcal{G}(\varphi_1)$ is a strategy profile $\langle \pi_1^*, \pi_2^* \rangle$ with the property that for $(i,j) \in \{(1,2),(2,1)\}$ we have

$$u_i(h, \pi_i^*, \pi_j^*, \varphi_i) \geq u_i(h, \pi_i, \pi_j^*, \varphi_i), \text{ for all } h \in \mathsf{PrefPlays}.$$

In a zero-sum ω-regular game, the Nash equilibrium $\langle \pi_1^*, \pi_2^* \rangle$ can be obtained as follows: Given $h \in \mathsf{PrefPlays}$,

$$\langle \pi_1^*, \pi_2^* \rangle = \arg \max_{\pi_1 \in \Pi_1} \min_{\pi_2 \in \Pi_2} \mathbf{Pr}^{\langle \pi_1, \pi_2 \rangle}(hh' \models \varphi_1).$$

To solve the Nash equilibrium in a zero-sum ω-regular game, we can use the software PRISM-games [26]. Qualitative solutions of zero-sum ω-regular games with simultaneous moves have been investigated in [31, 32].

In Definition 16, the game is common knowledge to both players. We now consider the case when the information about the game (*e.g.,* dynamics, payoffs) between two players is asymmetrical. Specifically, we consider the case when P2 has incomplete information about P1's temporal logic objective. Given that P2 has incomplete information, we introduce a *hypothesis space* for P2, denoted by X. The set X can be discrete and finite. For example, the set X can be a finite set of scLTL formulas that P2 believes that P1's true objective is one of these. The hypothesis space X can also be continuous. For example, each $x \in X$ is a distribution over a subset of scLTL formulas Φ so that $x(\varphi)$ is the probability that P2 believes $\varphi \in \Phi$ to be P1's true objective. For the time being, we do not restrict set X. In practice, a hypothesis space can be constructed from observations of any previous interactions or from the threat modeling [92], given P2's understanding of their interaction and potential objectives of an adversary.

Assumption 1. The asymmetrical information between players is introduced as follows:

- P1's objective is φ_1.
- P2 does not know φ_1 but has an initial hypothesis x_0 and a hypothesis space X about P1's objective.

The assumption describes scenarios commonly encountered in practice for both cooperative and adversarial interactions. For example, in a contested search and rescue mission, a search team has a sequence of waypoints that need to be visited according to a temporal order. The opponent may know the set of waypoints but is unclear about the team's temporal objective. The problem we aim to solve is stated informally as follows.

Problem 3. Given an adversarial encounter between P1 and P2 under information asymmetry as defined by Assumption 1, how to compute a strategy for P1 that maximizes the probability of satisfying φ_1 while a rational P2 responds optimally given P2's knowledge of the game?

Next, we introduce the modeling framework of hypergames and present a solution concept for a class of hypergames to solve P1's strategy.

6.4 Main Result

Hypergame, introduced in [14], can model strategic interactions when players have asymmetrical information. Intuitively, a hypergame is a game of games, each associated with a player's subjective view of its interaction with other players based on its own information and information about others' subjective views.

6.4.1 Static hypergames on graphs

We formally introduce a hypergame, which extends the hypergames from normal-form games [14, 123] to ω-regular games.

Definition 18 (Static ω-regular Hypergames)**.**
A static ω-regular hypergame of level-1 is defined as

$$\mathcal{HG}^1(x) = \langle \mathcal{G}(\varphi_1), \mathcal{G}(x) \rangle,$$

where $\mathcal{G}(\varphi_1)$ is a zero-sum ω-regular game given P1's objective φ_1, and $\mathcal{G}(x)$ is the game constructed by P2 given P2's hypothesis $x \in X$. If P1 is aware of P2's game $\mathcal{G}(x)$, then the resulting hypergame is said to be of level-2 and defined as

$$\mathcal{HG}^2(x) = \langle \mathcal{HG}^1(x), \mathcal{G}(x) \rangle,$$

where $\mathcal{HG}^1(x)$ is the level-1 hypergame constructed by P1, given P1's knowledge about the game constructed by P2, P1 computes its strategy by solving the level-1 hypergame $\mathcal{HG}^1(x)$ while P2 computes its strategy by solving the game $\mathcal{G}(x)$.

The game constructed by a player given its information and higher-order information is called the player's perceptual game. In level-2 hypergame $\mathcal{HG}^2(x)$, P1's perceptual game is $\mathcal{HG}^1(x)$, and P2's perceptual game is $\mathcal{G}(x)$. The game is *static* if neither player's perceptual game changes during their interaction.

Higher levels of hypergames can be defined through recursive reasoning about higher-order information (*e.g.,* what I know that you know that I know ...). In this work, level-2 hypergames suffice to capture the interaction because the highest order of information is that P1 knows what P2 knows. To construct level-3 hypergame, higher-order information is needed; for instance, P2 knows that P1 knows that P2 knows... This is not the case, given the class of asymmetric information considered herein.

For simplicity, we refer to level-2 ω-regular hypergames as hypergames in this paper whenever it is clear from the context.

We now discuss the solution concepts of hypergames. Given that different players may have different perceptions (*i.e.,* subjective views) of the utility functions in a hypergame, we denote u_i^j as the *utility function of player i perceived by player j*. Next, we generalize the related notions of subjective rationalizability and best-response equilibrium in hypergames from normal-form games in [104] to ω-regular hypergames.

Definition 19 (Subjective Rationalizability)**.**
Given a level-2 hypergame $\mathcal{HG}^2(x) = \langle \mathcal{HG}^1(x), \mathcal{G}(x) \rangle$, strategy $\pi_i^{*,2}$ is subjectively rationalizable (SR) for player 2 if and only if it satisfies, for any $h \in \mathsf{PrefPlays}$, for any $\pi_i \in \Pi_i$,

$$u_i^2(h, \pi_i^{*,2}, \pi_j^{*,2}, x) \geq u_i^2(h, \pi_i, \pi_j^{*,2}, x),$$

where $(i, j) \in \{(1, 2), (2, 1)\}$. Note that in the case that P2's hypothesis x is a distribution over Φ, the utility is calculated based on the expectation, that is, $u_i^2(h, \pi_i, \pi_j, x) = \sum_{\varphi \in \Phi} x(\varphi) u_i^2(h, \pi_i, \pi_j, \varphi)$.

The strategy $\pi_1^{*,1}$ is SR for P1 if and only if it satisfies, for any $h \in \mathsf{PrefPlays}$, for any $\pi_1 \in \Pi_1$,

$$u_1^1(h, \pi_1^{*,1}, \pi_2^{*,2}, \varphi_1) \geq u_1^1(h, \pi_1, \pi_2^{*,2}, \varphi_1),$$

where $\pi_2^{*,2}$ is SR for player 2.

In words, a strategy $\pi_i^{*,i}$ is called SR for player i if in player i's subjective view, it is the best response to player j's best response $\pi_j^{*,i}$, which is computed from player i's perceptual game. A pair of SR strategies $\langle \pi_1^{*,1}, \pi_2^{*,2} \rangle$ is called the best-response equilibrium of the hypergame $\mathcal{HG}^2(x)$.

In level-2 hypergame, P2's strategy is SR if it is rationalizable in P2's perceptual game $\mathcal{G}(x)$. P1's strategy is SR if it is the best response to P2's SR strategy. However, P1's SR strategy may not be consistent with P1's SR strategy predicated by P2. This inconsistency can be recognized by P2. When P2 notices the mismatch in the perceptual games, P2's perceptual game may evolve given new information; P2's current hypothesis $x \in X$ may change to a new hypothesis $x' \in X$ given new information.

6.4.2 Dynamic Hypergames on Graphs

To characterize P2's evolving perceptual game, we introduce an inference function.

Definition 20 (Inference)**.**
Assuming P2 has a complete observation of the game plays, a *perfect recall inference* function $\eta \colon X \times \mathsf{PrefPlays} \to X$ maps a hypothesis $x \in X$ and an observation (a history) $h \in \mathsf{PrefPlays}$ to a new hypothesis $x' = \eta(x, h) \in X$.

Anticipating that P2 will respond with an evolving hypothesis, P1 must calculate its moves to steer P2's inferred hypothesis and the resulting strategy. For the time being, we assume that P1 knows P2's inference mechanism and initial hypothesis and study how P1 can exploit P2's incomplete knowledge and inference mechanism for strategic advantage. In Section 6.4.4, we introduce a method that allows P1 to validate if its knowledge about P2's inference mechanism is correct or not.

We introduce a *transition system of P1's level-1 hypergame* to simultaneously capture the changes in game states given players' actions and the evolving perceptual game of P2.

Definition 21 (Transition System of P1's Level-1 Hypergame)**.**
Given the transition system $TS = \langle S, A, P, s_0, \mathcal{AP}, L \rangle$, the DFA $\mathcal{A} = \langle Q, \Sigma, \delta, \iota, F \rangle$ that

corresponds to P1's scLTL specification φ_1, and P2's hypothesis space X, the transition system of P1's level-1 hypergame is a tuple

$$\mathcal{H} = \langle V, A, \Delta, (s_0, h_0, q_0, x_0), \mathcal{F} \rangle,$$

where the components of hypergame transition system are defined as follows.

- $V = S \times \mathsf{PrefPlays} \times Q \times X$ is the set of states. Every state $v = (s, h, q, x) \in V$ has four components:
  - $s \in S$ is the state.
  - $h \in \mathsf{PrefPlays}$ is a history terminating in state $s \in S$.
  - $q \in Q$ is the automaton state for keeping track of P1's progress towards satisfying φ_1.
  - $x \in X$ represents the hypothesis of P2 given the history h.
- A is the set of joint actions.
- $\Delta \colon V \times A \times V \to [0, 1]$ is a probabilistic transition function defined as follows. Consider $v = (s, h, q, x)$ and $v' = (s', has', q', x')$, where has' is the history h appended with the new action a and state s',

$$\Delta(v' \mid v, a) = P(s' \mid s, a)\, \mathbf{1}(\delta(q, L(s')) = q') \cdot \mathbf{1}(\eta(x, has') = x').$$

- (s_0, h_0, q_0, x_0) is the initial state that includes the initial state in the transition system TS, the current history that consists of the initial state only, *i.e.,* $h_0 = s_0$, $q_0 = \delta(\iota, L(s_0))$, and P2's initial hypothesis x_0.
- $\mathcal{F} = S \times \mathsf{PrefPlays} \times F \times X$ is the set of final states for P1.

The transition function is understood as follows: Given a history h ending in the current state s and a joint action $a \in A$, the probability of reaching the next state s' is determined by $P(s' \mid s, a)$ in the transition system. Upon reaching s', P2 updates its hypothesis to $x' = \eta(x, has')$ (here, we assume the entire history is used for this update). Also, the transition in the specification automaton is triggered to reach state q' from state q given the label of the new state s'.

It is observed that the hypergame transition system in Definition 21 captures the dynamic evolution of P2's viewpoint. The history has time indices implicitly encoded. For example, a history $h = s_0 a_0 s_1 a_1 \cdots s_t$ is a history up to time step t.

Given P2's perceptual game evolving given the history and the inference function, P2 employs a *behaviorally subjectively rationalizable (BSR)* strategy, defined as follows.

Definition 22 (Behaviorally Subjectively Rationalizable Strategy)**.**
A strategy $\pi_2^{B,2}\colon \mathsf{PrefPlays} \times A_2 \to [0,1]$ is behaviorally subjectively rationalizable for P2 if, for any $h \in \mathsf{PrefPlays}$,

$$\pi_2^{B,2}(h) = \pi_2^{*,x,2}(h),$$

where $x = \eta(x_0, h)$, and $\pi_2^{*,x,2}\colon \mathsf{PrefPlays} \times A_2 \to [0,1]$ is a SR strategy for P2 in the hypergame $\mathcal{HG}^2(x)$.

Intuitively, playing a BSR strategy means that for any history h, P2 plays the SR strategy corresponding to its hypothesis constructed from the history h and its initial hypothesis. It is noted that the BSR strategy for P2 always exists in the class of hypergames studied herein. In [103], the author states the condition for the existence of the SR strategy as follows: P1 never excludes an action from P2's action set in P1's own perceptual game, where P1 thinks in P2's perceptual game, P2 believes this action is rationalizable [47] to P2. In the class of hypergames considered, P2's SR strategy exists as P2's perceptual game is a zero-sum game.

The hypergame transition system has a countably infinite set of states when X is finite. This is because a history can be of a finite but unbounded length. The entire history is maintained as a part of the state due to the general definition of the inference mechanism. In the next section, we show for some special cases of interactions, a state aggregation can be performed in the hypergame transition system to reduce the infinite state space to a finite state space.

6.4.3 Synthesizing P1's Deceptive Strategy

Given that P2 uses a BSR strategy, P1 can play deceptively by influencing P2's hypothesis so that P2's actions given P2's hypothesis can be advantageous for P1. To make P1's planning problem tractable, we introduce inference-equivalent histories to aggregate the countably infinite states of the transition system $\mathcal{H}$ of P1's level-1 hypergame into a finite state set.

Definition 23 (Inference-equivalent Histories)**.**
Given an inference function $\eta\colon X \times \mathsf{PrefPlays} \to X$ and a hypothesis x, two histories h_1 and h_2 are said to be (η, x)-equivalent if $\eta(x, h_1) = \eta(x, h_2)$ and for any $h' \in (A \times S)^+$,

$\eta(x, h_1 h') = \eta(x, h_2 h')$. The set of histories equivalent to $h \in \mathsf{PrefPlays}$ given hypothesis x is denoted by $[\![h]\!]_x$. If the equivalence between histories can be defined to be independent of the current hypothesis, that is, *for any* pair of hypotheses $x, x' \in X$, if h_1, h_2 are (η, x)-equivalent, then h_1, h_2 are also (η, x')-equivalent, then we say that the two histories h_1 and h_2 are η-*equivalent*. The set of histories η-equivalent to $h \in \mathsf{PrefPlays}$ is denoted by $[\![h]\!]$.

We consider a subset of dynamic hypergames, which satisfies the following assumption.

Assumption 2.

1. The hypothesis space X is discrete and finite.

2. The inference function η has a finite domain. The set of histories is grouped into a finite set of *inference-equivalent classes* (see Definition 23).

3. For any $x \in X$, P2 selects a quantal response strategy in the zero-sum game $\mathcal{G}(x)$ with a response parameter known to P1[1].

4. For any $x \in X$, P2's strategy in game $\mathcal{G}(x)$ is memoryless.

Assumptions 2-1 and 2-2 ensure the planning state space in $\mathcal{H}$ can be aggregated into a finite set. Assumption 2-3 enables us only to need to consider one SR strategy for P2 in game $\mathcal{G}(x)$, for each $x \in X$. If P2 takes the deterministic SR strategy instead of the quantal response, there may be multiple strategies. There are two possible approaches to dealing with multiple equilibria. The first one is that P1 must learn from online interaction about which SR strategy is employed by P2 and adapt P1's deceptive strategy. However, this adaptive deception requires further study of online optimization. The second one is that the deceptive planning algorithm should be robust for a range of possible equilibria strategies used by P2. Adaptive and robust deceptive planning are future extensions for this work.

Next, we formally state the deceptive planning problem for a subclass of dynamic hypergames.

[1] At each state, the quantal response strategy selects an action that is proportional to the exponential of λ-times the expected future payoffs from that state given the chosen action. The parameter λ is called the response parameter [52].

Problem 4. Given Assumptions 1 and 2, compute the optimal deceptive strategy for P1 in the dynamic hypergame $\mathcal{H}$, provided that P2 follows a BSR strategy.

We leverage the hierarchy of reasoning in level-2 hypergames and develop a two-step approach: Firstly, we construct P2's BSR strategy according to Definition 22: for each $x \in X$, we solve P2's SR strategy $\pi_2^{*,x,2}$ in the static hypergame $\mathcal{HG}^2(x)$. P2's BSR strategy is computed from the set of SR strategies given P2's evolving hypothesis (see Definition 22). Secondly, we incorporate P2's BSR strategies into the transition system in Definition 21 to reduce P1's planning problem into an MDP with a reachability objective, as stated next.

Definition 24 (Hypergame Reduced MDP)**.**
Under Assumption 2, the dynamic hypergame $\mathcal{H} = \langle V, A, \Delta, (s_0, h_0, q_0, x_0), \mathcal{F} \rangle$ reduces to a finite-state MDP with a reachability objective for P1,

$$\tilde{\mathcal{H}} = \langle \tilde{V}, A_1, \tilde{\Delta}, (s_0, [\![h_0]\!]_{x_0}, q_0, x_0), \tilde{\mathcal{F}} \rangle,$$

where

- $\tilde{V}$ is a finite and discrete set of states. Each state $\tilde{v} = (s, [\![h]\!]_x, q, x)$ consists of a state s, an inference-equivalent class given the (η, x)-equivalent relation, a state q in the DFA, and a hypothesis x of P2.

- $\tilde{\Delta} \colon \tilde{V} \times A_1 \times \tilde{V} \to [0, 1]$ is defined as follows: For any state $\tilde{v} = (s, [\![h]\!]_x, q, x)$, if $q = q_{\mathsf{sink}}$ — the sink state in the DFA $\mathcal{A}$, then state $\tilde{v}$ is a sink state. Given $\tilde{v}_1 = (s_1, [\![h_1]\!]_{x_1}, q_1, x_1)$ with $q_1 \neq q_{\mathsf{sink}}$, $a^1 \in A_1$, and $\tilde{v}_2 = (s_2, [\![h_2]\!]_{x_2}, q_2, x_2)$ and $h_1(a^1, a^2)s_2 \in [\![h_2]\!]_{x_2}$, then

  $$\tilde{\Delta}(\tilde{v}_2 \mid \tilde{v}_1, a^1) = \sum_{a^2 \in A_2} \pi_2^{*,x_1,2}(a^2 \mid s_1) \cdot P(s_2 \mid s_1, (a^1, a^2))\, \mathbf{1}(\delta(q_1, L(s_2)) = q_2),$$

  where $\pi_2^{*,x_1,2}(a^2 \mid s_1)$ is the probability of P2 selecting action a^2 given its current hypothesis x_1 and the current state s_1. That is, P2 uses a BSR strategy.

- $(s_0, [\![h_0]\!]_{x_0}, q_0, x_0) \in \tilde{V}$ is the initial state, given (s_0, h_0, q_0, x_0) is the initial state in the transition system $\mathcal{H}$.

- $\tilde{\mathcal{F}} = \{(s, [\![h]\!]_x, q, x) \in \tilde{V} \mid q \in F\}$ is the set of final states for P1, where F is the set of final states of DFA $\mathcal{A}$. P1's goal is to maximize the probability of reaching $\tilde{\mathcal{F}}$.

By construction, if a path in the MDP visits $\tilde{\mathcal{F}}$, P1 satisfies the scLTL formula φ_1. Thus, maximizing the probability of satisfying P1's specification is equivalent to maximizing the probability of reaching the set $\tilde{\mathcal{F}}$. The optimal policy for P1 in $\tilde{\mathcal{H}}$ is deceptive because by optimally planning in this MDP, P1 will select actions to influence P2's belief so that P2 takes actions that are advantageous for P1 to achieve its goal. We can employ dynamic programming to solve the optimal value function $\mathcal{V} : \tilde{V} \to \mathbf{R}$, which satisfies the Bellman optimality condition:

$$\mathcal{V}(\tilde{v}) = \max_{a \in A_1} \sum_{\tilde{v}' \in \tilde{V}} \tilde{\Delta}(\tilde{v}' \mid \tilde{v}, a)\mathcal{V}(\tilde{v}'), \text{ for all } \tilde{v} \notin \tilde{\mathcal{F}}, \tag{6.1}$$

and

$$\mathcal{V}(\tilde{v}) = 1, \text{ for all } \tilde{v} \in \tilde{\mathcal{F}},$$

where $\{\mathcal{V}(\tilde{v}) \mid \tilde{v} \in \tilde{V}\}$ is the set of decision variables. The optimal policy $\tilde{\pi}_1^*$ is computed from the optimal value function:

$$\tilde{\pi}_1^*(\tilde{v}) = \arg\max_{a \in A_1} \sum_{\tilde{v}' \in \tilde{V}} \tilde{\Delta}(\tilde{v}' \mid \tilde{v}, a)\mathcal{V}(\tilde{v}'), \text{ for all } \tilde{v} \notin \tilde{\mathcal{F}}.$$

The time complexity for solving MDPs with reachability objectives is polynomial in the size of state space and action space. Here, the size of state space in the MDP is $O(|S| \times N \times |Q| \times |X|)$, where N is the number of (η, x)-equivalent classes of histories in the game. The size of the action space in the MDP is $|A_1|$. Besides using dynamic programming, an MDP with a reachability objective can be solved using probabilistic model checking algorithms ([8, Chapter 10.1.1], [68]) and the existing PRISM toolbox [69].

Remark 4. Given that the problem can be large, ADP solutions of MDP can be used to reduce the number of decision variables [77]. For example, value function approximation in ADP uses a function approximator (such as a neural network) to approximate the value function, where the decision variables are coefficients of the value function. In the problem of large scale, it is often the case that the number of coefficients of the value function approximator is much smaller than the number of states.

To this end, we include Algorithm 6 to describe how to compute P1's SR strategy in the dynamic hypergames with temporal logic objectives.

Theorem 6. Assuming P1's knowledge about η is correct, the optimal strategy $\tilde{\pi}_1^* : \tilde{V} \times A_1 \to [0, 1]$ in the MDP $\tilde{\mathcal{H}}$ is P1's SR strategy in the dynamic hypergame given P2's evolving knowledge.

Algorithm 6: Computation of P1's SR Strategy

Input: Transition system TS, DFA $\mathcal{A}$, P2's inference function η, and hypothesis space X.

Output: P1's SR strategy $\tilde{\pi}_1^*$ in the dynamic hypergame.

1: Construct P1's level-1 hypergame $\mathcal{H}$ with TS, $\mathcal{A}$, X, and η. {Using Definition 21.}
2: **for** $x \in X$ **do**
3: Compute P2's SR strategy $\pi_2^{*,x,2}$ from game $\mathcal{G}(x)$. {Using Definition 18 and equilibrium solutions of games.}
4: **end for**
5: Construct $\tilde{\mathcal{H}}$ with $\{\pi_2^{*,x,2} \mid x \in X\}$ and $\mathcal{H}$. {Using Definition 24.}
6: $\tilde{\pi}_1^*, \mathcal{V} \leftarrow$ Solve MDP $\tilde{\mathcal{H}}$.
7: **return** $\tilde{\pi}_1^*$.

Proof. The construction of $\tilde{\mathcal{H}}$ is achieved through marginalizing P2's actions, given that P2 follows the BSR strategy in the dynamic hypergame $\mathcal{H}$. Thus, optimal planning in $\tilde{\mathcal{H}}$ computes the best response strategy for P1 against P2's BSR strategy. Any deviation from this best response strategy will not gain P1 a better outcome. □

Remark 5. Assumption 2-4 is not necessary. If P2's SR strategy $\pi_2^{*,x,2}$ is not memoryless in the game $\mathcal{G}(x)$ but represented using a finite-state controller (also known as a finite-memory policy), then we can augment the states in the hypergame transition system in Definition 21 with the states in the finite-state controller and planning in the augmented state space.

Definition 25 (Value of Deceit)**.**
Given the dynamic hypergame $\mathcal{H} = \langle V, A, \Delta, (s_0, h_0, q_0, x_0), \mathcal{F} \rangle$, the value of deceit is defined by

$$\mathsf{VoD} = \frac{\mathbf{Pr}^{\tilde{\mathcal{H}}, \tilde{\pi}_1^*}(s_0 h' \models \varphi_1)}{u_1(s_0, \pi_1^*, \pi_2^*, \varphi_1)},$$

where $\mathbf{Pr}^{\tilde{\mathcal{H}}, \tilde{\pi}_1^*}(s_0 h' \models \varphi_1)$ is the probability of satisfying P1's task φ_1 in the Markov chain induced from $\tilde{\mathcal{H}}$ under the optimal policy $\tilde{\pi}_1^*$, and $u_1(s_0, \pi_1^*, \pi_2^*, \varphi_1)$ is the value of the zero-sum game with complete information given P1's task φ_1.

Note that we have $\mathbf{Pr}^{\tilde{\mathcal{H}}, \tilde{\pi}_1^*}(s_0 h' \models \varphi_1) = \mathcal{V}(s_0, [\![h_0]\!]_{x_0}, q_0, x_0)$. In words, the value of deceit is the ratio between P1's probability of satisfying the scLTL objective using the solution of the dynamic hypergame and P1's probability of satisfying the same objective when both players have complete information. Based on the definition, P1 will only gain advantages with deception when the value of deceit is greater than one.

6.4.4 Detecting the Mismatch for Opponent Modeling

The effectiveness of P1's deceptive strategy hinges on the accuracy in modeling P2's inference and predicting P2's BSR strategy. In this section, we develop a method to detect if there is any inconsistency between the actual behavior of P2 observed during online interaction and P1's model of P2's behavior used in the MDP $\tilde{\mathcal{H}}$. If the method identifies the inconsistency, it alerts P1 that the optimal policy $\tilde{\pi}_1^*$ in $\tilde{\mathcal{H}}$ may not be effective.

Consider a finite history $h = s_0 a_0 s_1 a_1 \cdots s_n$, we denote the action pairs for P1 and P2 as $a_i = (a_i^1, a_i^2)$ for any $0 \leq i \leq n-1$. Under the assumption of complete observations, both players can observe the history h. By employing the DFA $\mathcal{A}$ corresponding to P1's scLTL specification φ_1 and the inference function η in the MDP $\tilde{\mathcal{H}}$, we can transform the history h to an augmented state-action sequence denoted by $\tilde{h} = \tilde{v}_0 a_0 \tilde{v}_1 a_1 \cdots \tilde{v}_n$, where $\tilde{v}_i = (s_i, [\![h_i]\!]_{x_i}, q_i, x_i)$ and $\tilde{v}_{i+1} = (s_{i+1}, [\![h_{i+1}]\!]_{x_{i+1}}, q_{i+1}, x_{i+1})$, $q_{i+1} = \delta(q_i, L(s_{i+1}))$, $h_{i+1} = h_i a_i s_{i+1}$, $x_{i+1} = \eta(x_i, h_{i+1})$. It is worth noting that for a unique history h, there exists a unique augmented state-action sequence $\tilde{h}$ as the transitions in the DFA and inference functions are deterministic, and the entire history up to step $i+1$ is used to compute the $(\eta, x_{i+1})-$equivalent histories at step $i+1$. The detection problem reduces to: Given the transition system TS, a predicted inference function η for P2 and P2's BSR strategies for a set of hypotheses X, how likely is the observation $\tilde{h}$ generated by our predicted model of opponent? If the likelihood of the observation $\tilde{h}$ is low (below a predefined threshold), then there is a mismatch in the model; otherwise, there is no mismatch.

We employ the likelihood ratio test [109] to answer this question. We have two hypotheses as follows:

H_0 the data is generated by our predicted model of P2.

H_1 the data is not generated by our prediction.

The goal is to test which hypothesis is a good fit for the data. Given P1's action sequences, the state sequences, and the MDP $\tilde{\mathcal{H}}$ from Definition 24, the predicted P2's policy, the likelihood of P2's action sequences is computed by

$$\mathsf{L}_0 = \mathsf{L}(a_0^2 a_1^2 \cdots a_{n-1}^2 \mid \{\pi_2^{*,x,2} \mid x \in X\}) = \prod_{i=0}^{n-1} \pi_2^{*,x_i,2}(a_i^2 \mid s_i),$$

where $s_i = \tilde{v}_i[1]$ and $x_i = \tilde{v}_i[4]$ are the first and last components of $\tilde{v}_i$.

At the same time, we obtain an estimate of P2's strategy from the history: For each $\tilde{v} = (s, [\![h]\!]_x, q, x)$,

$$\hat{\pi}_2^x(a^2 \mid s) = \begin{cases} \frac{\#\,\mathbf{1}(\tilde{v}, a^2)}{\#\,\mathbf{1}(\tilde{v})} & \text{if } \#\,\mathbf{1}(\tilde{v}) \neq 0, \\ \uparrow & \text{undefined otherwise,} \end{cases}$$

where $\#\,\mathbf{1}(\tilde{v}, a^2)$ is the number of times that P2 selects action a^2 given the current state $\tilde{v}$, and $\#\,\mathbf{1}(\tilde{v})$ is the number of times the state $\tilde{v}$ is visited. For unseen state-action pairs, we do not estimate the policy given the pair as it won't be used in the test. Based on the maximum likelihood estimate of P2's strategy from the history, we have

$$\mathsf{L}_1 = \mathsf{L}(a_0^2 a_1^2 \cdots a_{n-1}^2 \mid \{\hat{\pi}_2^x \mid x \in X\}) = \prod_{i=0}^{n-1} \hat{\pi}_2^{x_i}(a_i^2 \mid s_i).$$

The likelihood ratio is computed as

$$\lambda = \frac{\mathsf{L}_0}{\mathsf{L}_1}.$$

We conduct a likelihood ratio test and calculate $\chi_n^2 = -2 \ln \lambda$, which is an approximate Chi-square distribution of n degree of freedom, and n is the number of parameters estimated with maximum likelihood estimation, *i.e.,* $|\{\hat{\pi}_2^x(a^2 \mid s) \mid \#\,\mathbf{1}(\tilde{v}, a^2) \neq 0\}|$. By selecting a confidence level α, we reject the null hypothesis H_0 if χ_n^2 is larger than the Chi-square percentile with n degrees of freedom given the level α.

6.5 Case Study

This section presents a robot motion planning example to illustrate the proposed deceptive planning method. This case study includes an inference function for P2 based on the sliding-window change detection, introduced next.

6.5.1 Inference with sliding-window change detection

We introduce a class of inference algorithms based on change detection in the Markov chain (MC) [99]. Given P2's finite hypothesis space X, P2 can construct a set of games $\{\mathcal{G}(x) \mid x \in X\}$. For each game $\mathcal{G}(x)$, it is assumed that there is a unique equilibrium $\langle \pi_1^{*,x}, \pi_2^{*,x} \rangle$, where $\pi_i^{*,x} \colon \mathsf{PrefPlays} \times A_i \to [0, 1]$ is a stochastic strategy for player i given the hypothesis x. This equilibrium induces a probability measure $\mathbf{Pr}^{\langle \pi_1^{*,x}, \pi_2^{*,x} \rangle}$ over histories in $\mathcal{G}(x)$. For simplicity in notation, we denote $\mathbf{Pr}^{\langle \pi_1^{*,x}, \pi_2^{*,x} \rangle}$ as $\mathbf{Pr}^x$.

When P2's current hypothesis is x, P2 can detect a change from x to some $x' \in X$ using a sliding-window change detection algorithm based on the Cumulative SUM (CUSUM) statistic [10]. First, we are given a data point in the forms of history $h = s_0 a_0 s_1 \cdots s_n$ and a nominal model x_0. We denote the interval of a time window of size $m+1$ as $[k, k+m]$, and the history within this time window is $s_k a_k \cdots s_{k+m} a_{k+m} s_{k+m+1}$. Second, we denote the i-th observation of the transitions *within the time window* as $y_i = (a_{k+i-1}, s_{k+i})$ for $1 \leq i \leq m+1$. When $i = 0$, the 0-th observation within the window is $y_0 = (s_k)$. Intuitively, given a data point and a nominal model x_0, the sliding-window change detection algorithm uses a subsequence of history over a time window and detects if a change has occurred in the model that generates the data during this time window. Specifically, for each hypothesis $x \in X$ and a nominal model x_0, the algorithm computes the log-likelihood ratio, for $1 \leq j \leq m+1$,

$$R_j^x = \sum_{i=0}^{j} r_i^x,$$

where $r_i^x = \ln \frac{\mathbf{Pr}^x(y_i)}{\mathbf{Pr}^{x_0}(y_i)}$, and $\mathbf{Pr}^x(y_i)$ (resp. $\mathbf{Pr}^{x_0}(y_i)$) is the probability of observing the transition given the probability measure $\mathbf{Pr}^x$ (resp. $\mathbf{Pr}^{x_0}$).

The change detection lies in the difference between the log-likelihood ratio and its current minimum value. The CUSUM score is given by,

$$Z_l^x = R_l^x - \min_{1 \leq j \leq l} R_j^x, \text{ for } 1 \leq l \leq m+1.$$

Recursively, the CUSUM score is updated for each hypothesis $x \in X$ as

$$Z_l^x = \max\{0, Z_{l-1}^x + \ln \frac{\mathbf{Pr}^x(y_l)}{\mathbf{Pr}^{x_0}(y_l)}\}, \tag{6.2}$$

where $Z_0^x = 0$.

A change is detected at time t when the score of at least one model, say Z_l^x, exceeds a user-defined constant threshold $c > 0$. Formally, the *time of change* is given by

$$t = \min\{l \mid \exists x \in X,\ Z_l^x \geq c\}.$$

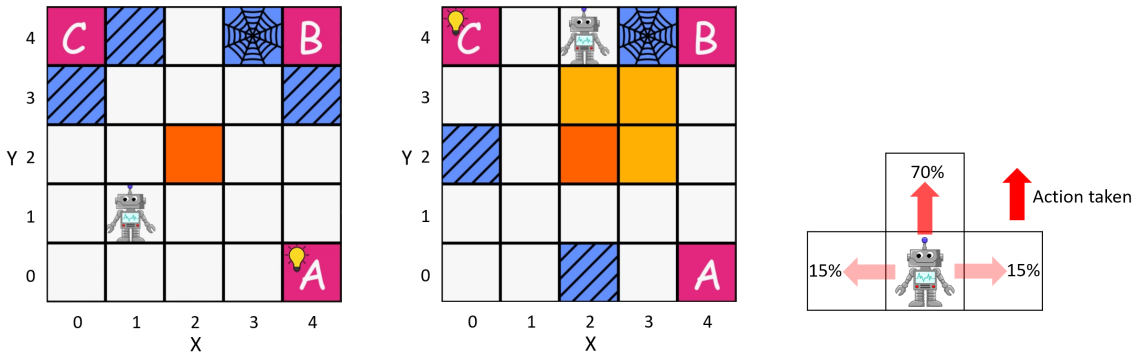
Once a change is detected, the algorithm sets the nominal model to be the current model, disregards the history until the change, and keeps running the online change detection given new observations from the change point onwards. When multiple models maintain similar CUSUM scores, we select one model based on some domain-specific heuristics or at uniformly random.

Lemma 11. Given a sliding-window change detection inference $\eta\colon X \times \mathsf{PrefPlays} \to X$ with window size $m+1$ and a finite hypothesis space X, two histories h_1, h_2 are (η, x)-*equivalent* if they share the same suffix [2] of length $m+1$.

Proof. The proof is based on the property of the change detection and thus omitted. □

6.5.2 Deceptive Planning with a Temporal Logic Objective

We consider two examples inspired by security games, referred to $world_1$ (Figure 6.11) and $world_2$ (Figure 6.12). In both worlds, a robot (P1) is to visit several regions of interest (labeled A, B, C and colored in red) according to a temporal ordering, and an observer (P2) can reallocate traps in cells colored in blue. Both games are concurrent: When P1 selects an action to move, P2 simultaneously chooses an action to reallocate the traps. When P1 enters the cell where P2 allocates the trap to that cell, we say that P1 is trapped. The game terminates in two ways: (a) P1 is trapped; (b) P1 completes its task.



(1) (2) (3)

Figure 6.1: (1) $world_1$'s initial configuration for P1 and P2. (2) $world_2$'s initial configuration for P1 and P2. Cells colored in yellow are walls. Bulbs indicate initial P2's predictions. (3) Robot's dynamics when the action "up" is taken.

We consider a set $\mathcal{AP}$ of atomic propositions:

- a: robot reaches regions A.
- b: robot reaches regions B.
- c: robot reaches regions C.

[2] For a word $w = \sigma_1\sigma_2\cdots\sigma_n$, a suffix of w is a word v of the form $\sigma_i\sigma_{i+1}\cdots\sigma_n$, where $1 \leq i \leq n$.

- o: robot reaches obstacles.

Formally, we describe P1's task by the formula as follows:

$$\varphi_1 = (\neg o \,\mathsf{U}\, a) \wedge (\neg(b \vee o) \,\mathsf{U}\, c).$$

That is, the robot needs to visit region A and region C without reaching obstacles. Before visiting region C, the robot cannot visit region B. The corresponding DFA is drawn in Figure 6.2.

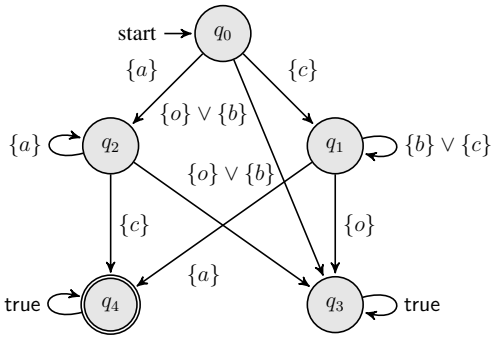


Figure 6.2: The task automaton with 5 states and 12 edges corresponds to φ_1, where $Q = \{q_i \mid i = 0, 1, 2, 3, 4\}$.

P1 can move in four compass directions, and P1's dynamics is plotted in Figure 6.13. A bouncing wall surrounds the grid world, *i.e.,* if P1 hits the wall, then P1 gets bounced back to P1's previous cell. The orange cell in the grid world is a static obstacle, labeled by o.

P2 can reallocate the traps (*i.e.,* dynamic obstacles) to a subset of cells colored in blue in $world_1$ and $world_2$. P2 can only use ℓ traps with n possible trap locations. Thus, the number of actions for P2 is $\binom{n}{\ell}$, *i.e.,* choose ℓ out of n. Every time P2 resets the location of any trap, it must wait at least k time steps to be able to reallocate any trap again. In the example of $world_1$, we select $n = 4$, $\ell = 1$, and let $k = 0$; In the example of $world_2$, we select $n = 3$, $\ell = 1$, and let k to be a variable.

In both examples: $world_1$ and $world_2$, the asymmetrical information is as follows:

- P1 knows the complete task φ_1.
- P2 does not know the complete task φ_1.

We refer to this situation as an *asymmetric information* case. On the other side, if P2 knows P1's complete task, then we refer to that as a *symmetric information* case. In the *asymmetric information* case, P2 has a hypothesis space $X = \{\neg o \,\mathsf{U}\, \phi \mid \phi \in \{a, b, c\}\}$.

Different behaviors under asymmetric and symmetric information cases in $world_1$

We compare P1's task completion rates between asymmetric and symmetric information cases.

In the asymmetric information case, for each $x \in X$, P2 solves a Stackelberg/leader-follower game and decides a trap configuration against the best response of P1 in game $\mathcal{G}(x)$. Let A_2 be the set of different configurations of traps. The strategy of P2 is obtained as follows: For any $s \in S$,

$$\pi_2^{*,x,2}(s) = \arg\min_{a^2 \in A_2} \max_{\pi_1} \mathbf{Pr}^{\pi_1}(hh' \models x \mid s, a^2),$$

where $\mathbf{Pr}^{\pi_1}(hh' \models x \mid s, a^2)$ is the probability of P1 satisfying the formula x given P2's action (trap configuration) a^2. For instance, if $x = \neg o \,\mathsf{U}\, a$ and robot is at the $(2, 4)$, then P2's optimal action is to allocate the trap to the blue cell right to the robot, that is $(3, 4)$. For each hypothesis $x \in X$ and state $s \in S$, P2 solves the optimal trap allocation action a^2 and also computes the best response of P1 that achieves the maximum probability of satisfying x from the state s. The joint strategy profiles for different hypotheses $x \in X$ also enable P2 to infer the subgoal of P1: P2 observes the behavior of P1 given the current trap location a^2 and then infers, for which x, P1's behavior matches with the best response given x and a^2 using the sliding-window change detection.

For the configuration of $world_1$, we evaluate different window sizes and find that sliding-window size $m + 1 = 2$ and user-defined threshold $c = 0.12$ achieve a good trade-off between space complexity and accuracy in prediction in this example. If we choose a window size and a user-defined threshold not accurately modeling P2, our method in subsection 6.4.4 enables P1 to detect the incorrect prediction of P2's behavior. Assuming P2 can reallocate traps anytime with $\pi_2^{B,2}$, we can construct an MDP $\tilde{\mathcal{H}}$ and solve for P1's optimal deceptive strategy denoted by $\tilde{\pi}_1^*$.

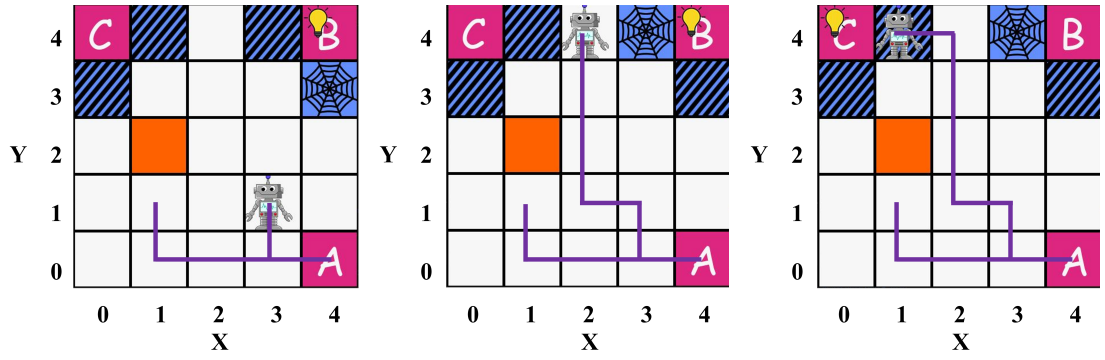
In the symmetric information case, P1 and P2 both have exact knowledge of task specification φ_1, and P1 wants to maximize the P1's probability of finishing the task; P2 wants to minimize the P1's probability of finishing the task. We denote the Nash Equilibrium strategy profile by $\langle \pi_1^*, \pi_2^* \rangle$, where the Nash Equilibrium strategy profile is obtained as follows:

$$\langle \pi_1^*, \pi_2^* \rangle = \arg\min_{\pi_2 \in \Pi_2} \max_{\pi_1 \in \Pi_1} \mathbf{Pr}^{\langle \pi_1, \pi_2 \rangle}(hh' \models \varphi_1).$$

Table 6.1: The completion rates for P1 in asymmetric information case and symmetric information case in $world_1$.

Info	P1 Policy	P2 Policy	Completion rate (P1)
Asymmetric	$\tilde{\pi}_1^*$	$\pi_2^{B,2}$	**66.96%**
Symmetric	π_1^*	π_2^*	29.69%

In Table 6.1, we list P1's completion rates for its task specification: one for asymmetric information case and one for symmetric information case. Table 6.1 indicates that under asymmetrical information, by following the deceptive strategy given P2 plays BSR strategy, P1 has a higher probability of satisfying the specification than the case by following the Nash Equilibrium strategy profile. The *value of deceit* in $world_1$ is $\mathsf{VoD} = \frac{66.96\%}{29.69\%} = 2.26$.



(1) (2) (3)

Figure 6.3: Three key steps of deception in the simulation. (1) P2 predicts P1 will reach B. (2) P2 reallocates the trap given P1's position. (3) P2 predicts that P1 will reach C, but it is too late for P2 to respond.

In this case, P2 can only place traps near region B and region C but not region A. We plot three key steps during the simulation in Figure 6.3. The solid lines denote the robot's trajectories. In Figure 6.3 (1), P2 predicts that P1 will reach region B after observing that the robot goes up. The prediction does not change until the robot reaches $(1, 4)$ in Figure 6.3 (3). When the robot reaches $(2, 4)$, P2 still predicts b (see Figure 6.3 (2)) and places the trap at $(3, 4)$ (see Figure 6.3 (3)). When the robot reaches $(1, 4)$, P2 correctly predicts c. But it is too late, and P2 cannot prevent the robot from reaching region C. The deceptive strategy leverages this information asymmetry to lead P1 to achieve a higher probability of finishing its task. We provide a short video [3] demonstrating the difference between P2's behaviors in the cases with asymmetric and symmetric information, respectively.

[3] https://www.dropbox.com/s/i98ka56gdhdvxgq/video_10_09_2021.mp4?dl=0

Next, we investigate how delays in reallocating traps for P2 would affect the completion rate of P1. However, in the $world_1$ example, we observed in experiments that any delay in reallocation could easily lead P1 to complete its task. Based on this observation, we construct another example $world_2$, and evaluate the completion rates for every k steps of delay and effectiveness of model mismatch in this example $world_2$.

Reallocation every k steps of delay in $world_2$

In this example, we assume that P2 is restricted to only reallocate the trap after k steps since the last reallocation, where k is an integer. P1 is aware of P2's delay k and synthesizes the deceptive strategy. Figure 6.4 shows the completion rate of the task (values of P1 at initial state $(2, 4)$ in Figure 6.12) under different steps of delay up to $k = 3$. The results indicate that with the increase of steps of delay, the probability of completing the task increases, and P1 exploits P2's delay and lack of information.

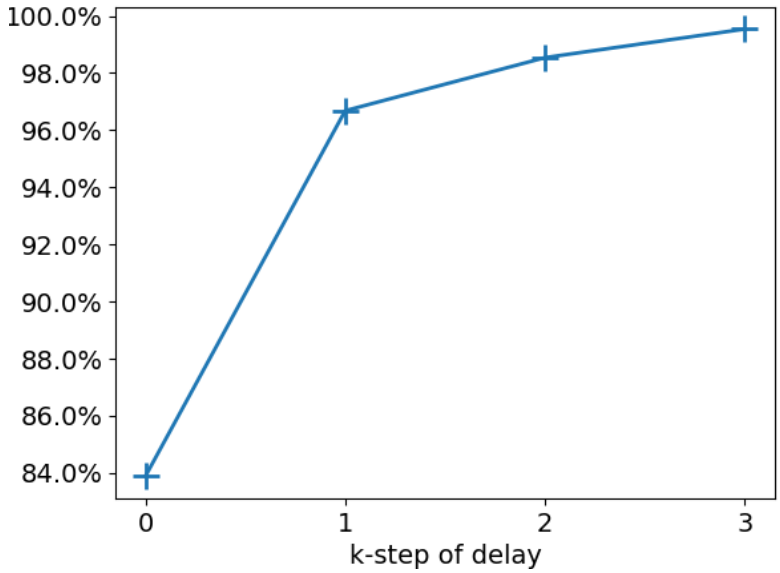


Figure 6.4: The task completion rates of P1 given P2 with k-step delay in reallocating traps, for $k = 0, 1, 2, 3$.

Detection of model mismatch in $world_2$

We use experiments in the configuration $world_2$ to demonstrate the effectiveness of the detection mechanism, that is, to identify whether there is a deviation from the predicted

opponent model of P2. We set the significance level $\alpha = 0.05$. If the likelihood of observed action sequences is smaller than or equal to 0.05, we reject the null hypothesis: the data is generated by our predicted model of P2.

We consider a case that P1 follows policy $\tilde{\pi}_1^*$, and P2 plays the policy predicted by P1 for the first four steps. After the first four steps, we let P2 play a random policy π_2^R, *i.e.*, $\pi_2^R(a \mid s) = \frac{1}{|A(s)|}$, for all $a \in A(s)$. The mismatch is detected at the 7-th step of the online interaction, and P1 is alerted that P2 deviates from the predicted policy. We compute λ after each step and plot it in Figure 6.5, where we also plot the χ^2. (The reason predicted $\lambda = 0$ is that the predicted policy $\pi_2^{B,2}$ is deterministic.) From Figure 6.5, we see that at the 7-th step of online interaction, we have $\lambda > \chi^2$, so we reject the null hypothesis H_0. The degree of freedom in the Chi-square detector is the number of the state-action pairs.

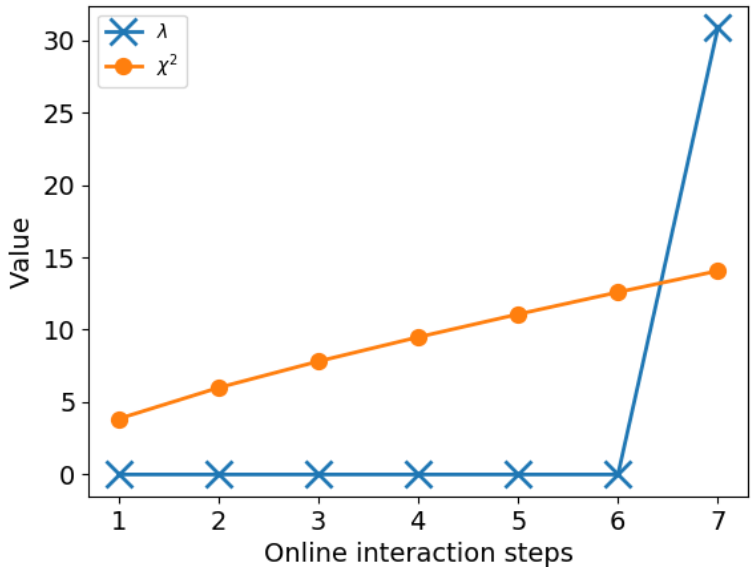


Figure 6.5: The likelihood ratio λ for online interaction between P1 and P2.

Complexity Our realization of the proposed framework in examples includes three major components: (a) Inference with sliding-window change detection, (b) Equilibrium solving of Stackelberg games, (c) MDP planning for deceptive planning. The inference with sliding-window change detection has an $O(m)$ time complexity, where $m + 1$ is the window size. It is noted that P2's BSR strategies are computed using a set of leader policies computed offline based on solving a set of Stackelberg games, one for each hypothesis. Given P2's BSR policy, we can reduce solving P1's optimal deceptive strategy problem

into an MDP planning problem, which can be solved in polynomial time in the size of the states and actions [81], where the state space is the product of the states in the game, the set of inference-equivalent histories, the DFA states, and a set of hypotheses. We solve the equilibrium of Stackelberg games and solve the MDP with the value iteration algorithm. We run algorithms on a Windows 10 machine with AMD Ryzen 9 5900X CPU and 16 GB RAM. The computational time of equilibrium solving of Stackelberg games is about 5 seconds, and the computational time of MDP planning is 140 seconds.

Finally, it is remarked that the deceptive planner can use different components given different inference algorithms and solutions of P2's BSR strategies. This analysis of complexity may not generalize to other classes of hypergames.

6.6 Conclusion

In this section, we propose a solution concept for a class of hypergames to solve deceptive strategies with temporal logic objectives. Our hypergame framework identifies two key components for deceptive planning: Opponent modeling and deceptive planning. The general framework can be extended to other classes of games with incomplete information. It is also important to note that the proposed approach does not generalize easily to partially observable games with two-sided partial observations. This is because two players may have different observations over the same history and incomplete information about what observation the other player has. The partial observation may potentially make player 1's subjective model of player 2's perceptual game diverge from the actual perceptual game of player 2.

Chapter 7

Conclusions

To express temporal objectives, this thesis considers a class of temporal logic formulas, PCTL and LTL formulas, for optimal planning of stochastic systems. PCTL formulas can reason about probabilistic system properties; LTL formulas can reason about liveness, fairness, and safety. This work starts with PCTL and formulates a constrained optimization problem for probabilistic planning with MDP incorporating the softmax Bellman operator. We enforce chance constraints translated from a class of PCTL formulas in this optimization. We demonstrate the correctness of the proposed translations. This thesis successfully develops a scalable ADP method by adopting stochastic programming techniques and on-policy sampling. Under several assumptions, we provide the almost sure convergence of the proposed ADP algorithm. The case study illustrates the efficacy of the ADP algorithm and the satisfaction of temporal objectives in PCTL formulas.

Leveraging existing LTL to DFA conversion, we can compactly represent the LTL formulas expressing temporal objectives. We describe an algorithm to discover structural information, termed topological order, in the translated DFA. Topological order helps generalize the optimal backup order from MDPs to product transition systems and divides the product state space into levels. Based on the generalized optimal backup order, updating the values in a level-by-level manner guarantees the optimality of value functions. The case study indicates the improved convergence of the classic algorithm. Further, we show that this generalized optimal backup order applies to our proposed ADP algorithm.

We use neural networks to approximate value and policy functions. When the stochastic system has continuous-state space, we devise an actor-critic, model-free reinforcement learning algorithm to avoid intractable system models. This actor-critic algorithm integrates topological order and modular learning to boost empirical performance further. The case

study shows that our proposed actor-critic algorithm matches or beats the baselines. Furthermore, we demonstrate the successful synthesis of a policy applicable to a real RC platform.

Hypergames allow capturing subjective views of both players. Given the inference function, we model the opponent's subjective view evolution to reduce the game to an optimal planning problem. We establish a solution concept of hypergames. This solution concept leads to a deceptive planning algorithm that learns a subjectively rationalizable strategy. Further, we design an online detection mechanism to alert the modeling error of the opponent's subjective view. The case study confirms the effectiveness of the synthesized policy in deception.

Future work can focus on extending our translation of PCTL formulas to a large class of temporal logic formulas. To further deal with large-scale MDPs, developing a distributed ADP based on decomposition-based planning is beneficial.

In the class of hypergames considered, we have made some assumptions about the adversary's inference mechanism and strategies. Future work can generalize the deterministic inference to probabilistic inference. One direction is considering a set of probabilistic distributions forming continuous hypothesis space X. In addition, robust Markov decision processes can be incorporated to deal with mismatches in the opponent model, provided the range of mismatches is known prior. If P2 has multiple BSR strategies, then P1 needs to consider more. One direction for P1 is to learn which strategy P2 uses and adapt its deceptive strategy accordingly. Another is for P1 to be robust for a range of P2's possible equilibrium strategies.

Ultimately, we present a comprehensive, efficient, model-free framework for formal policy synthesis for stochastic dynamic systems with temporal objectives. This thesis broadens the horizon of the probabilistic optimal planning for stochastic systems given high-level specifications in single- and double-player views, which potentially has extensive applications in defense operations, robotics, economics, networking, and other cyber-physical systems.

Bibliography

[1] ACHOUR, N., AND BRAIKIA, K. An mdp-based approach oriented optimal policy for path planning. In *2010 International Conference on Machine and Web Intelligence* (2010), IEEE, pp. 205–210.

[2] AHMADI, M., CUBUKTEPE, M., JANSEN, N., JUNGES, S., KATOEN, J.-P., AND TOPCU, U. The partially observable games we play for cyber deception. *ArXiv abs/1810.00092* (2018).

[3] AHO ALFRED, V., HOPCROFT JOHN, E., ULLMAN JEFFREY, D., AHO ALFRED, V., BRACHT GLENN, H., HOPKIN KENNETH, D., STANLEY JULIAN, C., JEAN-PIERRE, B., SAMLER, B. A., PETER, B. A., ET AL. *Data structures and algorithms*. USA: Addison-Wesley, 1983.

[4] ALI, H., GONG, D., WANG, M., AND DAI, X. Path planning of mobile robot with improved ant colony algorithm and mdp to produce smooth trajectory in grid-based environment. *Frontiers in neurorobotics 14* (2020), 44.

[5] ARMSTRONG, D., FRAZIER, G., CARTER, S., AND FRAZIER, T. A controller-based autonomic defense system. In *Proceedings DARPA Information Survivability Conference and Exposition* (2003), vol. 2, IEEE, pp. 21–23.

[6] ASADI, K., AND LITTMAN, M. L. An alternative softmax operator for reinforcement learning. In *International Conference on Machine Learning* (2017), PMLR, pp. 243–252.

[7] ASHTON, J., SPENCER, M. E., AND HUNTER, S. *Autonomous RC Car Platform*. PhD thesis, Worcester Polytechnic Institute, 2019.

[8] BAIER, C., AND KATOEN, J.-P. *Principles of model checking*. MIT press, 2008.

[9] BARTO, A. G., SUTTON, R. S., AND ANDERSON, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, 5 (1983), 834–846.

[10] BASSEVILLE, M., NIKIFOROV, I. V., ET AL. *Detection of abrupt changes: theory and application*, vol. 104. Prentice Hall Englewood Cliffs, 1993.

[11] BELLMAN, R. E. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[12] BELTA, C., YORDANOV, B., AND AYDIN GOL, E. *Temporal Logics and Automata*. Springer International Publishing, Cham, 2017, pp. 27–38.

[13] BELTA, C. A., MAJUMDAR, R., ZAMANI, M., AND RUNGGER, M. Formal Synthesis of Cyber-Physical Systems (Dagstuhl Seminar 17201). *Dagstuhl Reports 7*, 5 (2017), 84–96.

[14] BENNETT, P. G. Hypergames: developing a model of conflict. *Futures 12*, 6 (1980), 489–507.

[15] BERNINI, N., BESSA, M., DELMAS, R., GOLD, A., GOUBAULT, E., PENNEC, R., PUTOT, S., AND SILLION, F. Reinforcement learning with formal performance metrics for quadcopter attitude control under non-nominal contexts. *arXiv preprint arXiv:2107.12942* (2021).

[16] BERTSEKAS, D. P. *Nonlinear programming*. Athena scientific Belmont, 1999.

[17] BERTSEKAS, D. P. Neuro-dynamic programming. In *Encyclopedia of optimization*. Springer, 2008, pp. 2555–2560.

[18] BERTSEKAS, D. P., ET AL. Dynamic programming and optimal control 3rd edition, volume ii. *Belmont, MA: Athena Scientific* (2011).

[19] BLOEM, R., CHATTERJEE, K., AND JOBSTMANN, B. Graph Games and Reactive Synthesis. In *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Springer International Publishing, Cham, 2018, pp. 921–962.

[20] BOUYER, P. On the Computation of Nash Equilibria in Games on Graphs (Invited Talk). In *26th International Symposium on Temporal Representation and Reasoning (TIME 2019)* (Dagstuhl, Germany, 2019), J. Gamper, S. Pinchinat, and G. Sciavicco, Eds., vol. 147 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 3:1–3:3.

[21] BOZKURT, A. K., WANG, Y., AND PAJIC, M. Secure planning against stealthy attacks via model-free reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), IEEE, pp. 10656–10662.

[22] BROCKMAN, G., CHEUNG, V., PETTERSSON, L., SCHNEIDER, J., SCHULMAN, J., TANG, J., AND ZAREMBA, W. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[23] BUONGIORNO, J. Generalization of faustmann's formula for stochastic forest growth and prices with markov decision process models. *Forest science 47*, 4 (2001), 466–474.

[24] CARR, S., JANSEN, N., AND TOPCU, U. Verifiable rnn-based policies for pomdps under temporal logic constraints. *arXiv preprint arXiv:2002.05615* (2020).

[25] CHATTERJEE, K., AND HENZINGER, T. A. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences 78*, 2 (2012), 394–413.

[26] CHEN, T., FOREJT, V., KWIATKOWSKA, M., PARKER, D., AND SIMAITIS, A. Prism-games: A model checker for stochastic multi-player games. In *International Conference on TOOLS and Algorithms for the Construction and Analysis of Systems* (2013), Springer, pp. 185–191.

[27] CHOW, Y., GHAVAMZADEH, M., JANSON, L., AND PAVONE, M. Risk-constrained reinforcement learning with percentile risk criteria. *Journal of Machine Learning Research 18*, 167 (2018), 1–51.

[28] CIESINSKI, F., AND GRÖSSER, M. On probabilistic computation tree logic. In *Validation of Stochastic Systems*. Springer, 2004, pp. 147–188.

[29] CUBUKTEPE, M., XU, Z., AND TOPCU, U. Distributed policy synthesis of multiagent systems with graph temporal logic specifications. *IEEE Transactions on Control of Network Systems 8*, 4 (2021), 1799–1810.

[30] DAI, P., WELD, D. S., GOLDSMITH, J., ET AL. Topological value iteration algorithms. *Journal of Artificial Intelligence Research 42* (2011), 181–209.

[31] DE ALFARO, L., AND HENZINGER, T. Concurrent omega-regular games. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)* (2000), pp. 141–154.

[32] DE ALFARO, L., HENZINGER, T., AND KUPFERMAN, O. Concurrent reachability games. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)* (1998), pp. 564–575.

[33] DE ALFARO, L., HENZINGER, T. A., AND KUPFERMAN, O. Concurrent reachability games. *Theoretical Computer Science 386*, 3 (Nov. 2007), 188–217.

[34] DE FARIAS, D. P., AND VAN ROY, B. The linear programming approach to approximate dynamic programming. *Operations research 51*, 6 (2003), 850–865.

[35] DE FARIAS, D. P., AND VAN ROY, B. On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of operations research 29*, 3 (2004), 462–478.

[36] DEGRIS, T., WHITE, M., AND SUTTON, R. S. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839* (2012).

[37] DIMITROVA, R., FU, J., AND TOPCU, U. Robust optimal policies for markov decision processes with safety-threshold constraints. In *Decision and Control (CDC), 2016 IEEE 55th Conference on* (2016), IEEE, pp. 7081–7086.

[38] DING, X., SMITH, S. L., BELTA, C., AND RUS, D. Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control 59*, 5 (2014), 1244–1257.

[39] DING, X. C. D., SMITH, S. L., BELTA, C., AND RUS, D. Ltl control in uncertain environments with probabilistic satisfaction guarantees. *IFAC Proceedings Volumes 44*, 1 (2011), 3515–3520.

[40] DURET-LUTZ, A., LEWKOWICZ, A., FAUCHILLE, A., MICHAUD, T., RENAULT, E., AND XU, L. Spot 2.0—a framework for ltl and ω-automata manipulation. In *International Symposium on Automated Technology for Verification and Analysis* (2016), Springer, pp. 122–129.

[41] EGOROV, M., KOCHENDERFER, M. J., AND UUDMAE, J. J. Target surveillance in adversarial environments using pomdps. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (2016), AAAI Press, pp. 2473–2479.

[42] EL CHAMIE, M., YU, Y., AÇIKMEŞE, B., AND ONO, M. Controlled markov processes with safety state constraints. *IEEE Transactions on Automatic Control 64*, 3 (2018), 1003–1018.

[43] FAGIN, R., HALPERN, J. Y., AND MEGIDDO, N. A logic for reasoning about probabilities. *Information and computation 87*, 1-2 (1990), 78–128.

[44] FERGUSON-WALTER, K., FUGATE, S., MAUGER, J., AND MAJOR, M. Game theory for adaptive defensive cyber deception. In *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security* (New York, NY, USA, 2019), HotSoS '19, ACM, pp. 4:1–4:8.

[45] FROYLAND, G. Statistically optimal almost-invariant sets. *Physica D: Nonlinear Phenomena 200*, 3-4 (2005), 205–219.

[46] FU, J., AND TOPCU, U. Probably approximately correct mdp learning and control with temporal logic constraints. *arXiv preprint arXiv:1404.7073* (2014).

[47] FUDENBERG, D., AND TIROLE, J. *Game theory*. 1991.

[48] GAO, Q., PAJIC, M., AND ZAVLANOS, M. M. Deep imitative reinforcement learning for temporal logic robot motion planning with noisy semantic observations. In *2020 IEEE International Conference on Robotics and Automation (ICRA)* (2020), IEEE, pp. 8490–8496.

[49] GHARESIFARD, B., AND CORTES, J. Evolution of players' misperceptions in hypergames under perfect observations. *IEEE Transactions on Automatic Control 7*, 57 (2012), 1627–1640.

[50] GHARESIFARD, B., AND CORTÉS, J. Stealthy deception in hypergames under informational asymmetry. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 44*, 6 (2014), 785–795.

[51] GNEEZY, U. Deception: The role of consequences. *The American Economic Review 95*, 1 (2005), 384.

[52] GOEREE, J. K., HOLT, C. A., AND PALFREY, T. R. *Stochastic Game Theory for Social Science: A Primer on Quantal Response Equilibrium*. Edward Elgar Publishing, 2020, pp. 8–47.

[53] HAARNOJA, T., TANG, H., ABBEEL, P., AND LEVINE, S. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning* (2017), PMLR, pp. 1352–1361.

[54] HAARNOJA, T., ZHOU, A., ABBEEL, P., AND LEVINE, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning* (2018), PMLR, pp. 1861–1870.

[55] HANDEL, M. I. *Masters of war: classical strategic thought*. Routledge, 2005.

[56] HASANBEIG, M., ABATE, A., AND KROENING, D. Certified reinforcement learning with logic guidance. *arXiv preprint arXiv:1902.00778* (2019).

[57] HASANBEIG, M., KANTAROS, Y., ABATE, A., KROENING, D., PAPPAS, G. J., AND LEE, I. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *2019 IEEE 58th Conference on Decision and Control (CDC)* (2019), IEEE, pp. 5338–5343.

[58] HORÁK, K., ZHU, Q., AND BOŠANSKỲ, B. Manipulating adversary's belief: A dynamic game approach to deception by design for proactive network security. In *International Conference on Decision and Game Theory for Security* (2017), Springer, pp. 273–294.

[59] HOUSE, J. T., AND CYBENKO, G. Hypergame theory applied to cyber attack and defense. In *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense IX* (2010), E. M. Carapezza, Ed., vol. 7666, International Society for Optics and Photonics, SPIE, pp. 39 – 49.

[60] HUANG, L., AND ZHU, Q. Dynamic bayesian games for adversarial and defensive cyber deception. In *Autonomous cyber deception*. Springer, 2019, pp. 75–97.

[61] ICARTE, R. T., KLASSEN, T. Q., VALENZANO, R., AND MCILRAITH, S. A. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research 73* (2022), 173–208.

[62] IOFFE, S., AND SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (2015), PMLR, pp. 448–456.

[63] JIANG, Y., BHARADWAJ, S., WU, B., SHAH, R., TOPCU, U., AND STONE, P. Temporal-logic-based reward shaping for continuing reinforcement learning tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2021), vol. 35, pp. 7995–8003.

[64] JOTHIMURUGAN, K., BANSAL, S., BASTANI, O., AND ALUR, R. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems 34* (2021).

[65] JUNGES, S., JANSEN, N., DEHNERT, C., TOPCU, U., AND KATOEN, J.-P. Safety-constrained reinforcement learning for mdps. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2016), Springer, pp. 130–146.

[66] KARAMAN, S., AND FRAZZOLI, E. Linear temporal logic vehicle routing with applications to multi-uav mission planning. *International Journal of Robust and Nonlinear Control 21*, 12 (2011), 1372–1395.

[67] KUPFERMAN, O., AND VARDI, M. Y. Model checking of safety properties. *Formal Methods in System Design 19*, 3 (2001), 291–314.

[68] KWIATKOWSKA, M., NORMAN, G., AND PARKER, D. Stochastic model checking. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems* (2007), Springer, pp. 220–270.

[69] KWIATKOWSKA, M., NORMAN, G., AND PARKER, D. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)* (2011), G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806 of *LNCS*, Springer, pp. 585–591.

[70] KWON, Y., AND AGHA, G. Ltlc: Linear temporal logic for control. In *International Workshop on Hybrid Systems: Computation and Control* (2008), Springer, pp. 316–329.

[71] LAHIJANIAN, M., ANDERSSON, S., AND BELTA, C. Control of markov decision processes from pctl specifications. In *Proceedings of the 2011 American Control Conference* (2011), IEEE, pp. 311–316.

[72] LAHIJANIAN, M., WASNIEWSKI, J., ANDERSSON, S. B., AND BELTA, C. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *2010 IEEE International Conference on Robotics and Automation* (2010), IEEE, pp. 3227–3232.

[73] LAVAEI, A., SOMENZI, F., SOUDJANI, S., TRIVEDI, A., AND ZAMANI, M. Formal controller synthesis for continuous-space mdps via model-free reinforcement learning. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)* (2020), IEEE, pp. 98–107.

[74] LEFEBVRE, D., AND DAOUI, C. Control design for bounded partially controlled tpns using timed extended reachability graphs and mdp. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 50*, 6 (2018), 2273–2283.

[75] LENNARTSON, B., AND JIA, Q.-S. Reinforcement learning with temporal logic constraints. *IFAC-PapersOnLine 53*, 4 (2020), 485–492.

[76] LI, L., AND FU, J. Approximate dynamic programming with probabilistic temporal logic constraints. *American Control Conference* (2019).

[77] LI, L., AND FU, J. Topological approximate dynamic programming under temporal logic constraints. In *2019 IEEE 58th Conference on Decision and Control (CDC)* (2019), IEEE, pp. 5330–5337.

[78] LI, X., VASILE, C.-I., AND BELTA, C. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), IEEE, pp. 3834–3839.

[79] LI, X., XING, K., ZHOU, M., WANG, X., AND WU, Y. Modified dynamic programming algorithm for optimization of total energy consumption in flexible manufacturing systems. *IEEE Transactions on Automation Science and Engineering 16*, 2 (2018), 691–705.

[80] LILLICRAP, T. P., HUNT, J. J., PRITZEL, A., HEESS, N., EREZ, T., TASSA, Y., SILVER, D., AND WIERSTRA, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[81] LITTMAN, M. L., DEAN, T. L., AND KAELBLING, L. P. On the complexity of solving markov decision problems. *arXiv preprint arXiv:1302.4971* (2013).

[82] LUNA, R., LAHIJANIAN, M., MOLL, M., AND KAVRAKI, L. E. Asymptotically optimal stochastic motion planning with temporal goals. In *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 335–352.

[83] MANNA, Z., AND PNUELI, A. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, Berlin, Heidelberg, 1992.

[84] MAO, H., CHEN, Y., JAEGER, M., NIELSEN, T. D., LARSEN, K. G., AND NIELSEN, B. Learning markov decision processes for model checking. *arXiv preprint arXiv:1212.3873* (2012).

[85] MASTERS, P., AND SARDINA, S. Deceptive path-planning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (2017), AAAI Press, pp. 4368–4375.

[86] MASTERS, P., AND SARDINA, S. Deceptive path-planning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (2017), AAAI Press, pp. 4368–4375.

[87] MISRA, S., BERA, S., AND OBAIDAT, M. S. Economics of customer's decisions in smart grid. *IET Networks 4*, 1 (2015), 37–43.

[88] MNIH, V., BADIA, A. P., MIRZA, M., GRAVES, A., LILLICRAP, T., HARLEY, T., SILVER, D., AND KAVUKCUOGLU, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (2016), PMLR, pp. 1928–1937.

[89] MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., AND RIEDMILLER, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[90] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., ET AL. Human-level control through deep reinforcement learning. *nature 518*, 7540 (2015), 529–533.

[91] MUNIR, A., AND GORDON-ROSS, A. An mdp-based dynamic optimization methodology for wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems 23*, 4 (2011), 616–625.

[92] MYAGMAR, S., LEE, A. J., AND YURCIK, W. Threat modeling as a basis for security requirements. In *Symposium on requirements engineering for information security (SREIS)* (2005), vol. 2005, Citeseer, pp. 1–8.

[93] NACHUM, O., NOROUZI, M., XU, K., AND SCHUURMANS, D. Bridging the gap between value and policy based reinforcement learning. *arXiv preprint arXiv:1702.08892* (2017).

[94] NIU, L., AND CLARK, A. Optimal secure control with linear temporal logic constraints. *IEEE Transactions on Automatic Control 65*, 6 (2019), 2434–2449.

[95] OLSON, E. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation* (2011), IEEE, pp. 3400–3407.

[96] ORNIK, M., AND TOPCU, U. Deception in optimal control. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (2018), IEEE, pp. 821–828.

[97] OSBORNE, M. J., AND RUBINSTEIN, A. *A course in game theory*. 1994.

[98] PITERMAN, N., PNUELI, A., AND SA'AR, Y. Synthesis of Reactive(1) Designs. In *Verification, Model Checking, and Abstract Interpretation* (Berlin, Heidelberg, 2006), E. A. Emerson and K. S. Namjoshi, Eds., Lecture Notes in Computer Science, Springer, pp. 364–380.

[99] POLANSKY, A. M. Detecting change-points in markov chains. *Computational statistics & data analysis 51*, 12 (2007), 6013–6026.

[100] POLYDOROS, A. S., AND NALPANTIDIS, L. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems 86*, 2 (2017), 153–173.

[101] PUTERMAN, M. L. Markov decision processes. *Handbooks in operations research and management science 2* (1990), 331–434.

[102] RUTTEN, J. J., KWIATKOWSKA, M., NORMAN, G., AND PARKER, D. *Mathematical techniques for analyzing concurrent and probabilistic systems*. No. 23. American Mathematical Soc., 2004.

[103] SASAKI, Y. Subjective Rationalizability in Hypergames. *Advances in Decision Sciences 2014* (2014), 263615.

[104] SASAKI, Y., AND KIJIMA, K. Hierarchical hypergames and bayesian games: A generalization of the theoretical comparison of hypergames and bayesian games considering hierarchy of perceptions. *Journal of Systems Science and Complexity 29*, 1 (2016), 187–201.

[105] SCHILLINGER, P., BÜRGER, M., AND DIMAROGONAS, D. V. Hierarchical ltl-task mdps for multi-agent coordination through auctioning and learning. *The international journal of robotics research* (2019).

[106] SCHILLINGER, P., BÜRGER, M., AND DIMAROGONAS, D. V. Auctioning over probabilistic options for temporal logic-based multi-robot cooperation under uncertainty. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (May 2018), pp. 7330–7337.

[107] SCHLENKER, A., THAKOOR, O., XU, H., TAMBE, M., VAYANOS, P., FANG, F., TRAN-THANH, L., AND VOROBEYCHIK, E. Deceiving Cyber Adversaries: A Game Theoretic Approach. 11.

[108] SCHULMAN, J., WOLSKI, F., DHARIWAL, P., RADFORD, A., AND KLIMOV, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[109] SEVERINI, T. A. *Likelihood methods in statistics*. Oxford University Press, 2000.

[110] SHOUKRY, Y., NUZZO, P., BALKAN, A., SAHA, I., SANGIOVANNI-VINCENTELLI, A. L., SESHIA, S. A., PAPPAS, G. J., AND TABUADA, P. Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming. In *2017 IEEE 56th annual conference on decision and control (CDC)* (2017), IEEE, pp. 1132–1137.

[111] SINCLAIR, A. *Algorithms for random generation and counting: a Markov chain approach*. Springer Science & Business Media, 2012.

[112] SINHA, A., FANG, F., AN, B., KIEKINTVELD, C., AND TAMBE, M. Stackelberg Security Games: Looking Beyond a Decade of Success. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (Stockholm, Sweden, July 2018), International Joint Conferences on Artificial Intelligence Organization, pp. 5494–5501.

[113] SONG, Z., PARR, R., AND CARIN, L. Revisiting the softmax bellman operator: New benefits and new perspective. In *International conference on machine learning* (2019), PMLR, pp. 5916–5925.

[114] STECH, F. J., HECKMAN, K. E., AND STROM, B. E. Integrating cyber-d&d into adversary modeling for active cyber defense. In *Cyber Deception* (2016).

[115] SUGIYAMA, M. *Statistical reinforcement learning: modern machine learning approaches*. Chapman and Hall/CRC, 2015.

[116] SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[117] TADIĆ, V. B., MEYN, S. P., AND TEMPO, R. Randomized algorithms for semi-infinite programming problems. In *Probabilistic and Randomized Methods for Design under Uncertainty*. Springer, 2006, pp. 243–261.

[118] TEMPO, R., CALAFIORE, G., AND DABBENE, F. *Randomized algorithms for analysis and control of uncertain systems: with applications*. Springer Science & Business Media, 2012.

[119] TESAURO, G., ET AL. Temporal difference learning and td-gammon. *Communications of the ACM 38*, 3 (1995), 58–68.

[120] THAKOOR, O., TAMBE, M., VAYANOS, P., XU, H., KIEKINTVELD, C., AND FANG, F. Cyber Camouflage Games for Strategic Deception. In *Decision and Game Theory for Security* (Cham, 2019), T. Alpcan, Y. Vorobeychik, J. S. Baras, and G. Dán, Eds., Lecture Notes in Computer Science, Springer International Publishing, pp. 525–541.

[121] TOMPKINS, P., STENTZ, A., AND WETTERGREEN, D. Mission-level path planning and re-planning for rover exploration. *Robotics and Autonomous Systems 54*, 2 (2006), 174–183.

[122] VAN HASSELT, H., GUEZ, A., AND SILVER, D. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence* (2016), vol. 30.

[123] VANE, R. R. I. *Using Hypergames to Select Plans in Competitive Environments*. PhD thesis, George Mason University, 2000.

[124] VARDI, M. Y. An automata-theoretic approach to linear temporal logic. In *Logics for concurrency*. Springer, 1996, pp. 238–266.

[125] WAGNER, A. R., AND ARKIN, R. C. Acting deceptively: Providing robots with the capacity for deception. *International Journal of Social Robotics 3*, 1 (2011), 5–26.

[126] WANG, J., DING, X., LAHIJANIAN, M., PASCHALIDIS, I. C., AND BELTA, C. A. Temporal logic motion control using actor–critic methods. *The International Journal of Robotics Research 34*, 10 (2015), 1329–1344.

[127] WANG, M., HIPEL, K. W., AND FRASER, N. M. Solution concepts in hypergames. *Applied Mathematics and Computation 34*, 3 (1989), 147 – 171.

[128] WATKINS, C. J., AND DAYAN, P. Q-learning. *Machine learning 8*, 3 (1992), 279–292.

[129] WOLFF, E. M., TOPCU, U., AND MURRAY, R. M. Optimization-based trajectory generation with linear temporal logic specifications. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014), IEEE, pp. 5319–5325.

[130] WONGPIROMSARN, T., ULUSOY, A., BELTA, C., FRAZZOLI, E., AND RUS, D. Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In *2013 IEEE International Conference on Robotics and Automation* (2013), IEEE, pp. 5011–5018.

[131] YAN, Z., KREIDIEH, A. R., VINITSKY, E., BAYEN, A. M., AND WU, C. Unified automatic control of vehicular systems with reinforcement learning. *IEEE Transactions on Automation Science and Engineering* (2022).

[132] YAO, B., IMANI, F., AND YANG, H. Markov decision process for image-guided additive manufacturing. *IEEE Robotics and Automation Letters 3*, 4 (2018), 2792–2798.

[133] YUAN, L. Z., HASANBEIG, M., ABATE, A., AND KROENING, D. Modular deep reinforcement learning with temporal logic specifications. *arXiv preprint arXiv:1909.11591* (2019).

[134] ZHANG, T., HUANG, L., PAWLICK, J., AND ZHU, Q. Game-theoretic analysis of cyber deception: Evidence-based strategies and dynamic risk mitigation. *ArXiv abs/1902.03925* (2019).