PLANNING UNDER UNCERTAINTY IN SAFETY-CRITICAL SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Arec Jamgochian
January 2024

Arec Jamgochian

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mykel J. Kochenderfer) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mac Schwager)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Marco Pavone)

Approved for the Stanford University Committee on Graduate Studies

Abstract

From warehouses and manufacturing lines to homes and offices, from roads and seas, to skies and space, autonomous systems promise to improve efficiency, unlock human potential, and explore new frontiers. Many autonomous systems already make decisions that impact our everyday lives. As technology continues to develop and the cost of compute continues to decrease, autonomous systems will continue integrating into society. However, a unifying necessity for safety-critical systems to deploy autonomously in the real world is the need to be able to reason about their environments and make good decisions to satisfy their objectives.

For autonomous systems to be deployed successfully, it often does not suffice to plan deterministically, that is, assuming that everything will ‘go as planned‘ against a single string of outcomes. Rather, agents must reason about the uncertainty that can arise, either from inexact actuation or sensing, imperfect information, unclear objectives, unknown motives of other participants, or complex environments. These sources of uncertainty can significantly complicate autonomous decision-making and can ultimately lead to catastrophic errors. By explicitly reasoning about these sources of uncertainty, this thesis introduces new methods for planning safely against them.

First, this thesis investigates methods that use data to overcome uncertainty in action outcomes and agent objectives. Specifically, we consider using human driving demonstrations alongside simulators to overcome objective uncertainty for autonomous driving in complex urban environments. Previous approaches that used simulators to help imitate human driving were typically limited to relatively simple scenarios. We introduce Safety-Aware Hierarchical Adversarial Imitation

Learning (SHAIL), a method that scales safety-critical data-driven decision-making to complex problems through reliance on hierarchical decomposition and safety predictions. After building a simulator to test counterfactuals of real-world driving decisions, we demonstrate empirically that SHAIL can improve safety compared to other data-driven decision-making methods, especially in unseen driving scenarios.

Next, we turn to safe planning under outcome and state uncertainty when models for those uncertainties are known a priori. Here, we impose safety through constraints on agent plans, modeling problems as constrained partially observable Markov decision processes (CPOMDPs). Approximate CPOMDP solutions are typically limited to small, discrete actions and observation spaces. We introduce algorithms that extend online search-based planning in CPOMDPs to domains with large or continuous state, action, and observation spaces by using methods that artificially limit the width of a search tree in unpromising areas and satisfy constraints using dual ascent. We empirically compare the effectiveness of our proposed algorithms on continuous CPOMDPs that model both toy and real-world safety-critical problems. In doing so, we demonstrate that CPOMDP planning can be effective in continuous domains.

Unfortunately, the algorithms we introduce for safe online planning in continuous CPOMDPs are still restricted to relatively small problems. Fortunately, as noted for urban driving, many large planning problems can be decomposed hierarchically. In our final contribution, we introduce Constrained Options Belief Tree Search (COBeTS) to scale continuous CPOMDP planning to much larger problems with favorable hierarchical decompositions by planning over macro-actions (i.e. low-level controller options). We demonstrate COBeTS in several large, safety-critical, uncertain domains, showing that it can plan successfully while non-hierarchical baselines cannot. Importantly, we show that with constraint-satisfying macro-actions, COBeTS can guarantee safety regardless of planning time. In summary, our contributions improve planning safety in domains with quantifiable outcome, state, and/or objective uncertainty through novel applications of hierarchies and/or constraints.

Acknowledgments

This work would not have been possible without the support of many individuals and institutions. I would like to first and foremost thank the National Science Foundation Graduate Research Fellowship and the Virtual Vehicle GmbH K2 Digital Mobility program for funding my doctorate, as well as the Armenian General Benevolent Union for their scholarship support.

Next, I would like to thank my academic advisors. Thank you Mykel for taking a chance on me. Besides being incredibly knowledgeable and productive, you show time and time again how far thoughtfulness and unfettering kindness can go in bringing out the best in others. You are my biggest role model. Thanks as well to my reading committee Mac Schwager and Marco Pavone, not only for supporting this dissertation but also for supporting my growth as a roboticist. Taking your classes in my first year gave me some of my fondest memories, and I could not imagine being able to learn as much in as short a period of time without you.

Thank you Dr. Bernhard Bardstätter for all the academic guidance throughout the years. Thank you as well to Dr. Anton Fuchs and Dr. Daniel Watzenig, who alongside Bernhard helped facilitate multiple visits to Graz, Austria, including an incredible ten-week rotation at Virtual Vehicle GmbH. I'd also like to thank Dr. Christoph Stiller at KIT in Karlsruhe, Germany for facilitating my rotation there and treating me as a part of your lab. Your lab retreat taught me more about automated driving than I could have hoped to possibly learn elsewhere. This time in Karlsruhe and Graz was the most memorable of my PhD. I would not be the same person without these experiences and I am eternally grateful.

I would like to thank everyone who made my internships great learning experiences, namely, thank you to Omar Bentahar and Kyle Wray at Nissan-Renault-Mitsubishi for facilitating my research on autonomous vehicles, and Stephen Boyd and Tina Diao at BlackRock AI Labs for being incredible mentors in practical machine learning. Thank you Kyle for supporting my growth and research directions in planning, serving on my defense committee, and inspiring many themes of this thesis. Thank you Stephen for serving as my defense chair and for captivating me and every audience with masterful, humorous scientific communication.

This thesis would not have been possible without all of the many incredible mentors and collaborators along the way. Specifically, I'd like to thank Etienne Buehrle, Hugo Buurmeijer, Suhas Chundi, Johannes Fischer, Soyeon Jung, Bernard Lange, Sheng Li, Xiaobai Ma, John Mern, Robert Moss, Jinkyoo Park, Junyoung Park, Patrick Slade, Paula Stocco, Zachary Sunberg, Julie Walker, for all their wonderful collaborations, and Anthony Corso, Kunal Menda, Kyle Wray, and Di Wu for all their incredible mentorship. Thank you especially to Kunal for all of your one-on-one mentorship during COVID. You truly helped me become a significantly better programmer and researcher, and really gave me a home in SISL. I would like to acknowledge everyone in SISL for all the interesting conversations, alongside years of frisbee, lunches, and lab outings.

Thank you to the many friends I have met over the last few years. This thesis would not be possible without the rest of the Aero/Astro '20 cohort, whose companionship and joint commiseration were absolutely essential to getting through the MS program. Thank you to my lifelong friends from high school, undergrad, Valencia, Karlsruhe, Graz, and Yerevan who have made the last years the best of my life. Thank you to the wonderful roommates I've had the pleasure of living with over the years: Charly Zhang, Jesse Hoke, Ferdinand Legros, Jason Qin, Joey Curti, Bernard Lange, Andrei Graur, Alban Broze, Bhagya Bevangi, Adrien Specht, Cedric Katte, and Mobin YahyazadehJeloudar. Last but not least, thank you to the Rains graduate community. I have been incredibly fortunate to be able to live with an incredible community of smart, wacky, quirky individuals who love life, and this thesis would not be possible without you.

Thank you to my partner and adventure buddy Cheyenne for all her love and support this last year. This thesis would not have been possible without you. I truly admire your passion for science, I treasure all our goofy moments together, and I am looking forward to all the adventures the future has in store.

Finally, I'd like to thank my family for all of their support over the years. To my cousins in the United States thank you for being incredible role models to look up to. To my cousins in Armenia, thank you for treating me as siblings and always giving me a home. To my dad, thank you for giving me a love of culture. To my mom, thank you for your deep curiosity for the mysteries of the world, and for your lifelong struggle to make sure I have everything I need to succeed.

Contents

1 Introduction

Recent years have seen widespread deployment of systems that make decisions autonomously or semi-autonomously. These decision-making *agents* are designed for many purposes and come in many forms. They include physical systems designed to automate repetitive tasks (e.g. home robots, autonomous vehicles, delivery drones, industrial robots, agricultural robots), systems that work with humans to augment human decision-making (e.g. medical, aeronautical, and financial decision support systems), and systems that enable otherwise impossible tasks (e.g. underwater robots, space exploration vehicles). Autonomous agents promise to improve our lives, and their adoption in novel applications will no doubt continue.

A key unifying consideration for all these agents is safety–how can we ensure that agents designed to act autonomously in the real world cannot cause harm? In order to ensure safety, it is often necessary for agents to quantify and plan robustly against uncertainty in their environments. This thesis introduces methods that improve agents' abilities to plan safely under different modes of uncertainty, specifically through novel applications of constraints and hierarchies.

1.1 Planning under Uncertainty

The goal of planning is to use environment models to optimize sequences of decisions by predicting their effects. Planning stands as a cornerstone of autonomy, underpinning the ability of intelligent agents to operate independently. It is a crucial process that equips autonomous systems with the capability to set objectives, formulate strategies, and execute actions to achieve desired goals. Planning provides a roadmap for these agents in navigating complex and dynamic environments, allowing them to adapt to unforeseen circumstances and make decisions based on a structured assessment of available information.

Planning in the real world is complicated significantly because of the uncertainty. In real-world scenarios, uncertainty is pervasive and inherent, stemming from factors such as incomplete information, unpredictable events, and environmental variations. Ignoring uncertainty can lead to suboptimal decisions, inefficiencies, and unexpected outcomes. More importantly, accounting for uncertainty is essential for risk management and safety, particularly in fields like autonomous driving, finance, and healthcare, where the consequences of erroneous decisions can be severe. Planning under uncertainty allows for risk assessment and mitigation strategies to be incorporated into the decision-making process. By acknowledging and modeling uncertainty, planners can create more adaptable and robust solutions. This adaptability is especially crucial in dynamic environments where conditions may change rapidly, and rigid plans could prove ineffective. Finally, considering uncertainty promotes transparency and accountability in decision-making, as it forces planners to document assumptions, potential sources of error, and the limitations of their plans.

This thesis will consider planning under three modes of uncertainty:

1. **Outcome uncertainty**, often also called transition uncertainty, models how taking the same action from the same precondition (i.e. an environment *state*) can yield multiple different outcomes. The framework we use to model optimal sequential decision-making under transition uncertainty is the *Markov decision process* (*MDP*), which allows us to define sets of states (i.e. preconditions) that agents can be in, the actions they can take, the possible next-state transitions (i.e. postconditions) those actions induce, and the objectives (rewards) that those agents seek to optimize.

2. **State uncertainty** models the uncertainty that an agent might face about its decision-making preconditions that arise from imperfect information. State

and transition uncertainty can be modeled jointly using the *partially observable Markov decision process* (*POMDP*) framework, which augments an MDP with the set of observations that the agent may receive and how those observations are correlated with the true (hidden) state of the environment.

3. **Objective uncertainty**, alternatively called reward uncertainty, models the uncertainty that an agent may face in the objective it should optimize. It is often practical to consider overcoming objective uncertainty by relying on data about decisions made by experts. Outcome and objective uncertainty can be modeled using reward-free Markov decision processes (*MDP*$\backslash$*R*), and plans can be optimized while inferring objectives from data using *imitation learning*.

In summary, modeling uncertainty is essential for many decision-making systems but can add complexity to the planning problem. Next, we will discuss the importance of safety in planning and the approach we take for safe planning under uncertainty.

1.2 Safety in Planning

As mentioned, in many applications, considerations for safety are absolutely critical, as consequences for erroneous decisions can be catastrophic. There are many methods that try to achieve safety in planning.

Some methods try to evaluate whether decision-making systems or their components will become unsafe if deployed. Some examples of methods for evaluation under test include empirical methods for simulating possible failure modes, validation to search for sequences of decisions that could result in failures, and formal methods to verify that the systems are guaranteed to meet certain requirements. In contrast, the methods that we will employ for improving safety in this thesis focus on explicit *design* for runtime safety. The question we try to answer is — *in systems with high degrees of uncertainty, how can we design planning modules with confidence that plans are safe at runtime?* Many tools exist to design for runtime safety. For example, runtime monitoring with out-of-distribution detection or uncertainty quantification can enable safe decision-making through redundancy. Additionally, probabilistic safe sets can be integrated with control barrier functions to create control laws that ensure safety with a high probability. Instead, in this thesis, we will focus on the novel applications of *hierarchical decomposition* and *constraints* to solve safety-critical problems under uncertainty:

Hierarchical decomposition allows for extremely difficult problems to be broken down into hierarchies of subproblems with more tractable solutions. Hierarchies can induce safety by allowing safe solutions to scale to large decision-making problems, and by enabling runtime monitoring with safety redundancies.

Constraints specify safety through inviolable decision-making objectives. In contrast to *soft* constraints, which penalize unwanted behavior in the decision-making objective, *hard* constraints specify the safety conditions required for plans to satisfy in order to be feasible. Uncertainty while planning can make constraint satisfaction difficult.

1.3 *Approach*

In each chapter of this work, we describe a novel application of hierarchies and/or constraints to solve safety-critical planning problems under uncertainty. We use different representative safety-critical problem domains to empirically judge the performance of our contributions.

In Chapter 3, we investigate the problem domain of autonomous driving in urban environments. In many problems, it is not always straightforward to specify a planning objective, as optimal decision-making can be influenced by many factors. This problem is prevalent in autonomous driving, where a cluttered road environment with many interacting users can significantly complicate decision-making. However, for the specific domain of autonomous driving, large datasets of human (*expert*) driving can be used to plan without explicitly specifying objectives.

These human driving datasets consist typically of low-level trajectories, that is, observations about the environments alongside low-level actions that human drivers took (e.g. steering angles and accelerations). Typical imitation learning methods allow a controller to learn how to imitate these low-level decisions. However, in autonomous driving (and many other robotic applications), control methods can guarantee safety at the lowest level, and it is more practical instead to decide on appropriate controllers (e.g. high-level or macro actions) to select. As such, we investigate using a hierarchical decomposition to learn how to select sequences of controllers such that low-level trajectories are imitated under state and objective uncertainty.

We introduce Safety-Aware Hierarchical Adversarial Imitation Learning (SHAIL), which alternates between learning a high-level controller-selection policy, and a scorer that discriminates between real and simulated trajectories. By invoking a hierarchical decomposition, SHAIL is able to incorporate external predictions of controller safety to do runtime monitoring. These predictions can come from a number of methods, but prove essential as fail-safes for ensuring safety. Training SHAIL requires simulating counterfactual controller selection, which we do by building a simulator from the Interaction dataset of complex urban driving scenes. We evaluate SHAIL against a number of baseline imitation learning methods, finding it outperforms them in scenarios that are available in the training set, but more importantly, in scenarios that are held out. Evaluating and ensuring out-of-distribution safety is key for data-driven decision-making systems.

After investigating how hierarchies can be used to improve safety for planning under outcome and objective uncertainty, we turn to safe planning under outcome and state uncertainty. In many problems, decision-making agents lack the full information required to make optimal decisions and must make the trade-off between gathering necessary information and acting to optimize their objectives. Guaranteeing safety in such settings is extremely difficult, as the agents may not always have sufficient state information to satisfy safety specifications. In Chapters 4 and 5, we model safety specifications using constraints on accumulated costs, which may accrue gradually or suddenly. We model safe planning under uncertainty using *constrained partially-observable Markov decision processes* (*CPOMDPs*) and turn our attention to developing algorithms to solve CPOMDPs in large problem domains.

In Chapter 4, we investigate novel algorithms for solving CPOMDPs *online* using

Monte Carlo tree search (MCTS). Unlike offline planning, which optimizes a decision from every possible environment state, online planning is able to scale to much larger problems by only considering the possible outcomes immediately accessible at runtime. We introduce three MCTS methods that scale online CPOMDP planning to large and continuous action and successor spaces by leveraging *progressive widening*, a technique to artificially limit search tree branching until the search algorithm builds confidence in its plan. We demonstrate that our algorithms can successfully plan while satisfying safety constraints on three target continuous problem domains. These domains include a real-world-inspired carbon sequestration domain where an agent lacks information about subsurface geometries and properties and must plan to sequester carbon without it leaking to the surface.

Though the methods introduced in Chapter 4 can plan over continuous spaces, they have significant limitations. First, they are limited to relatively small planning problems, as there is a tradeoff between the degree of branch sub-sampling and plan accuracy. Additionally, though they may satisfy constraints in the limit, there are no guarantees that they will satisfy constraints anytime, which is an important consideration for runtime safety.

Chapter 5 addresses these shortcomings by invoking a hierarchical decomposition and introducing Constrained Options Belief Tree Search (COBeTS) to plan over high-level *options* (e.g. macro-actions) instead of low-level actions. Similar to Chapter 3, COBeTS assumes a set of low-level control policies, but instead performs a hierarchical Monte Carlo tree search to optimize controller selection by imagining the outcomes of executing these low-level control policies. We show that COBeTS can be used to scale safe planning under outcome and state uncertainty to large robotic problems where non-hierarchical baselines fail. We also show that if the low-level controllers are able to satisfy assigned constraints, then COBeTSsearch will be safe regardless of how long it is allowed to run before returning a decision. Finally, we show empirically that hierarchical search with many (possibly bad) control options is able to outperform low-level search, motivating future work where safe search is used to compensate for automatically generated control policies.

In summary, this thesis provides different methods for improving safety in autonomous systems when planning under different modes of uncertainty. Specifically, this thesis considers applications of hierarchies and constraints in order to plan under outcome, state, and objective uncertainty. Table 1.1 summarizes how the different sections of this thesis address different modes of uncertainty by implementing hierarchies or constraints.

Table 1.1: Outline comparing the sources of uncertainty in each chapter alongside the methods used to achieve safety.

	Sources of Uncertainty			**Techniques for Safety**	
	Outcome	State	Objective	Hierarchies	Constraints
Chapter 3	✓		✓	✓	
Chapter 4	✓	✓			✓
Chapter 5	✓	✓		✓	✓

1.4 *Contributions*

Our main contributions as they pertain to hierarchical adversarial imitation learning, continuous constrained POMDP planning, and hierarchical constrained POMDP planning are summarized below.

Hierarchical Imitation Learning

We introduce Safety-aware Hierarchical Adversarial Imitation Learning (SHAIL) in order to learn how to choose high-level control policies (e.g. options, macro-actions) that ultimately imitate low-level trajectory data. We demonstrate SHAIL against other imitation learning methods on a simulator that we build from human driving demonstration data in urban environments.

Continuous Constrained POMDP Planning

We investigate the application of constraints to POMDP planning in continuous domains, formulating tree search algorithms to plan online by artificially limiting tree branching. We demonstrate that our algorithms are able to satisfy constraints in large or continuous planning problems, including on a real-world-inspired carbon sequestration domain.

Hierarchical Constrained POMDP Planning

We investigate inducing a hierarchy in order to extend continuous CPOMDP planning to even larger planning problems. We introduce a hierarchical belief-state tree search algorithm (COBeTS) and demonstrate that it is able to solve difficult planning algorithms that our continuous CPOMDP algorithms could not, including two new robotic localization CPOMDP problems. We show that when low-level controllers can satisfy constraints assigned to them by our search, that COBeTSis guaranteed to be safe anytime. Finally, we show that the reduction in search tree size induced by hierarchical planning allows for search over many more actions than in analogous non-hierarchical problems.

1.5 *Outline*

The remainder of this thesis is organized as follows.

Chapter 2 describes different formulations for decision-making under uncertainty. It first provides the necessary background on the mathematical frameworks for planning under outcome uncertainty (Markov decision processes), state uncertainty (partially observable Markov decision processes), and objective uncertainty (imitation learning). It then dives into some of the necessary background on the tools we apply for improving safety, namely hierarchies and constraints.

In Chapter 3, we introduce SHAIL, a method that applies hierarchies and safety predictors to improve imitation learning in the safety-critical application of autonomous driving in complex urban environments. We design a simulator derived from real-world driving data and evaluate SHAIL against other imitation learning methods in scenarios that are both seen and unseen during training.

In Chapter 4, we formulate methods that apply constraints to POMDPs with continuous domains and solve them online through tree search. We evaluate these methods in three different continuous domains.

Building on this, Chapter 5 introduces COBeTS to apply hierarchies to scale continuous CPOMDP planning to even larger problems. We evaluate our approach on large constrained robotic localization problems that non-hierarchical baselines cannot safely solve. We also examine the effects of planning with feasible low-level control options.

Finally, Chapter 6 summarizes this thesis and highlights our main contributions. We conclude by discussing areas for future work.

2 Background

This chapter overviews the necessary background and related work for this thesis. We begin with a brief outline of planning, then continue with overviews of optimal planning under outcome uncertainty, state uncertainty, and objective uncertainty. We then overview methods for achieving safety while planning under uncertainty by using hierarchical decompositions and constraints.

2.1 Planning

Autonomous agents use percepts from their environment to take actions [1]. Autonomy is captured in the study of artificial intelligence, or "[the automation of] activities that we associate with human thinking, activities such as decision-making, problem-solving, learning..." [2]. In planning, agents reason about the consequences of their actions in order to satisfy long-term objectives. Planning is a key pillar underpinning autonomy.

There are multiple distinctions that differentiate classes of modern planning algorithms. One such distinction is whether plans are open-loop or closed-loop. In *closed-loop planning*, agents optimize action plans with the assumption that they will be able to use future information in future decisions. As such, agents can optimize a *conditional plan* (i.e. a *contingency plan*, which is a decision tree of actions conditioned on future percepts. Computing optimal closed-loop plans is extremely difficult for many reasons. The *curse of dimensionality* describes the difficulty in scaling optimal planning as a result of large action and perception spaces. The *curse of history* describes the combinatorial explosion in planning complexity with time

```
procedure Execute($s_0$)
    $s \leftarrow s_0$
    while $\neg$Terminal($s$)
        $a \leftarrow$ Plan($s$)
        $s \leftarrow$ ActAndPerceive($s, a$)
```

Algorithm 2.1: Fully observable receding horizon control

horizon. In contrast, *open-loop plans* assume no future perception, instead optimizing a single sequence of actions against all possible outcomes. Though this may pacify the curse of history, it can often lead to suboptimal plans.

Another methodological distinction is whether plans are computed offline or online. In *offline planning*, a decision-making policy π is pre-computed for fast runtime decision-making given any history of environment percepts. In contrast, *online planning* describes the process of planning in real-time while interacting with the environment. Online planning can often scale to larger problems than offline planning by considering only the possible decision-making outcomes reachable from the real-time environment state.

Online planning is typically coupled with actuation in a *receding horizon* fashion. That is, autonomous systems will plan a sequence of actions, execute the first action in their plan, perceive new environment information, and replan. In control theory, this is called *closed-loop control* or feedback control. Open-loop planning coupled with receding-horizon (closed-loop) control is often referred to as *model predictive control (MPC)*.

Algorithm 2.1 depicts receding horizon control in a fully observable environment, that is, one where all information required to make an informed decision is observed at each time step. Throughout this thesis, decision-making preconditions are referred to as *states* $s \in \mathcal{S}$, where decisions are referred to as *actions* $a \in \mathcal{A}$. Starting from an initial state s_0 and continuing until a termination condition is reached, in each step of a feedback controller, a planning routine selects an action for the agent to take and the agent perceives the outcome.

In contrast to feedback control, *sensorless planning* (also called conformant planning or *open-loop control*) describes actuation when sensor feedback is not available from the environment. Sensor feedback is the default assumption in the rest of this thesis, that is, that planning agents are able to observe the impact of their actions and use that information to replan.

Note that open-loop planning differs from open-loop control, just as closed-loop planning differs from closed-loop control. Figure 2.1 (todo) differentiates some of these nuances. We depict planning and actuation in four different settings, with planning depicted vertically and actuation depicted horizontally. In sensorless, open-loop planning (i.e. open-loop control with open-loop planning, Figure 2.1a), a single sequence of decisions is planned and executed independent of feedback. In closed-loop control from offline planning (Figure 2.1b), a policy optimized offline is queried for actions at every step. In model predictive control (Figure 2.1c), a new open loop plan is generated at each time step after receiving environment percepts, and the first step is executed. In online, receding-horizon, closed-loop planning (Figure 2.1d), a new closed-loop plan is regenerated at each step, and the first action is executed.

Classical planning is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment [1]. Classical planning was developed to plan high-level robot actions by using first-order logic to reason symbolically about the environment. STRIPS-style planning uses an axiomatic model to encode logical statements about a static environment alongside preconditions and effects induced by actions. It then searches for sequences of actions that can transition the environment to a goal state[3].

Over the years, classical planning has evolved and been standardized using the Planning Domain Definition Language [4]. It has shown success in critical planning applications in controlled environments, including autonomous space mission planning with Remote Agent [5] and the Europa toolkit [6], warehouse operations planning [7], and large-scale military logistics [8], which more than paid back DARPA's 30-year investment in AI [1].

Though STRIPS-style classical planning can scale to extremely large problems, it has a number of limitations. One shortcoming is a loose definition of optimality. Classical planning investigates whether or not a sequence of actions can transition the environment to a goal state. It does not, for example, compare different action sequences to minimize costs accrued in their execution.

Additionally, many real-world decisions face significant uncertainty, which can arise for many reasons. Such reasons might include imperfect actuation, imperfect sensing, imperfect information, unclear objectives, latent motives of other participants, and unpredictability in complex environments. Safety-critical applications must be robust to uncertainty. Explicitly reasoning over that uncertainty can significantly improve the quality of safe decision-making. Though explicitly modeling uncertainty can seem daunting, access to data and faster computing resources have facilitated a shift in modeling paradigms.

Due to the prevalence of uncertainty in safety-critical applications, the rest of this thesis relies on methods for *planning under uncertainty*. As such, the remaining background chapters will overview common frameworks for planning under uncertainty. Section 2.2 will outline rational decision-making under uncertainty about the outcomes that agent actions can have. Section 2.3 will augment this outcome-uncertain decision-making framework with state uncertainty, while Section 2.4 will augment this outcome-uncertain decision-making framework with objective uncertainty. Meanwhile, Section 2.5 will overview how to improve safety by augmenting these formulations with hierarchical decomposition, while Section 2.6 will overview how to achieve safety through the application of constraints.

2.2 *Markov decision processes (MDPs)*

As discussed in Section 2.1, significant shortcomings of classical planning include a narrow definition of optimality and a lack of consideration for uncertainty. Unfortunately, both of these shortcomings are often important considerations for real-world safety-critical decision-making. For example, it is often the case that decisions made given the same decision-making preconditions can result in different outcomes, for example, as a result of imperfect actuation or external environmental factors.

The Markov decision process (MDP) is a powerful mathematical framework that defines decision-making under outcome uncertainty [9], [10]. In an MDP, decisions are made in discrete intervals and can cause a multiplicity of stochastic successor states. These successor states follow a *Markovian* probability distribution, that is, one that remains stationary with time. Crucially, the agent state is fully observable, meaning that after actuating, the agent observes the true outcome of its decision and regains all of the necessary information to make future decisions.

The MDP will lay the groundwork for rational decision-making under uncertainty in this thesis. In this chapter, we provide a formal definition of the MDP and a notion of optimality. We then provide different solution methods for different classes of MDPs. Finally, we conclude by discussing alternative frameworks for fully observable planning under outcome uncertainty.

2.2.1 *Formal definition*

The key components of the MDP are as follows:

- **States**, $s \in \mathcal{S}$, describe the instantaneous state of the environment and include all information for optimal decision-making.
- **Actions**, $a \in \mathcal{A}$, describe the space of allowed decisions.
- **Reward models**, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, describe the immediate rewards for taking actions, encoding objectives for agents to optimize.
- **Transition models**, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ encode the Markov likelihoods with which actions a transition states s to successor states s'. That is, $T(s, a, s') = P(s' \mid s, a)$.

The MDP encodes a fully observable stochastic process where at discrete time steps, an agent takes an action, receives a reward, and perceives the new state of the environment. The transition model encodes the uncertainty in the agent dynamics conditioned on the agent action and the previous environment state. In discrete state spaces, the transition model encodes the mapping from a state and action to a probability mass function over the successor state (i.e. $\sum_{s'} T(s, a, s') = 1$ for all s, a). In continuous state spaces, the transition model encodes a probability density function (i.e. $\int T(s, a, s')ds' = 1$ for all s, a). We note that for the remainder of this section, expectations over transitions are computed assuming discrete state spaces, but can be easily extended to continuous state spaces by replacing sums with integrals where appropriate.

In fully observable MDPs, agent *policies* π map states to actions. Optimal policies optimize expected cumulative reward. That is, a rational agent should form a policy that optimizes all the rewards it will see over its lifetime. Though MDPs can be defined over finite or indefinite horizons, in this thesis, we will study the infinite-horizon MDP. In infinite-horizon MDPs, the expected cumulative reward can be unbounded. As such, one notion of optimality is defined using the expected average reward, that is, the reward received per time step. However, a more commonly used approach is to introduce a discount factor $\gamma \in (0, 1)$ and optimize the expected discounted cumulative reward of policy π:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))\right], \tag{2.1}$$

where the expectation is computed over stochastic state transitions. Sometimes, MDP models are accompanied by initialization states s_0 or state distributions $s_0 \sim b_0(s) = p(s_0 = s)$. Realizations of a Markov decision process up to time T are referred to as trajectories $\tau_T = s_0 a_0 s_1 a_1 s_2 a_2 ... s_{T-1} a_{T-1} s_T a_T$.

Common approaches to optimizing policies in MDPs rely on *dynamic programming* to compute and store optimal solutions to planning subproblems. As such, it is convenient to use the *Bellman equations* to define *value* (or *utility*) as the expected discounted cumulative reward of starting in state s and following policy π:

$$V_R^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V_R^\pi(s'), \tag{2.2}$$

and to define the *action-value* of starting in state s, taking action a and then following π:

$$Q_R^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_R^\pi(s'), \tag{2.3}$$

An optimal policy π^* is one that obtains maximal value at all states, and therefore must satisfy the *Bellman optimality equations*:

$$V_R^*(s) = \max_{a\in\mathcal{A}} Q_R^*(s,a), \tag{2.4}$$

$$Q_R^*(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} T(s,a,s')V_R^*(s'), \tag{2.5}$$

$$\pi^*(s) = \arg\max_{a\in\mathcal{A}} Q_R^*(s,a). \tag{2.6}$$

In infinite horizon MDPs, optimal policies will be stationary and deterministic. In contrast, in finite-horizon MDPs, optimal policies may be time-dependent. In constrained MDPs, which we discuss in Section 2.6, optimal policies may be stochastic.

2.2.2 *Solution methods*

With relatively few discrete states and actions, two algorithms are useful for solving infinite-horizon, discrete-time, discounted MDPs optimally. The first, *policy iteration*, alternates between evaluating and improving policies. In policy evaluation, the value of a fixed policy is found by solving Equation (2.2) jointly across in every state. In policy improvement, one-step lookaheads are used alongside evaluated values in order to generate a new policy. That is, at iteration k:

$$\pi^{k+1}(s) = \arg\max_{a\in\mathcal{A}} R(s,a) + \gamma \sum_{s'\in\mathcal{S}} T(s,a,s')V_R^k(s'). \tag{2.7}$$

The second algorithm, *value iteration*, iterates *Bellman backups*, or applications of the Bellman optimality equations in Equation (2.4), across the state space. Iteration can either be done synchronously [10] or asynchronously [11]. Both algorithms are guaranteed to converge towards an optimal policy. As the *Bellman residual* between successive iterations, $\|V_R^{k+1} - V_R^k\|_\infty$, drops below a threshold δ, the *policy loss* to the optimal policy $\|V_R^\pi - V_R^*\|_\infty$ is bounded by $2\delta\gamma^2/(1-\gamma)^2$ [12].

For MDPs with continuous or a large number of discrete states and actions, different algorithms can be used to approximate optimal plans. For continuous MDPs with linear Gaussian transitions and concave quadratic rewards, a linear quadratic regulator (LQR) can be used to determine an exactly optimal closed-loop policy, expressing actions as a linear transformation of state [13]. With nonlinear but approximately Gaussian dynamics, iterative LQR (iLQR) [14] and differential dynamic programming (DDP) [15] can determine approximately optimal closed-loop plans by iterating dynamic programming with linearized dynamics and convexified rewards.

Model predictive control (MPC) methods generate open-loop plans in a receding horizon controller. In MDPs with unimodal transition uncertainty, it is common to ignore uncertainty and optimize most-likely trajectories, often referred to as *trajectory optimization*. Methods to do so include convex programming, mixed integer programming, and sequential convex programming [16], [17]. Direct collocation methods optimize actions and future states jointly while including dynamics as hard constraints, while shooting methods optimize actions and roll out dynamics in the objective function.

Under significant outcome uncertainty, open-loop planning may optimize actions against a set of sampled action-conditioned scenarios [18]. One way of doing this is by using common random numbers to generate action-dependent trajectory scenarios and optimizing the average reward of those trajectories [19]. To generate robust open-loop plans under uncertainty, robust MPC methods optimize plans against worst-case scenarios [20]. Multi-forecast MPC methods jointly optimize multiple plans against multiple scenarios but constrain the first actions to be identical across plans [21], [22].

One can generate closed-loop plans online through expectimax search, alternating maximization over actions with expectation over outcomes. However, exact forward search can be extremely difficult as the size of the search tree (and therefore search complexity) scales exponentially with the planning horizon. Instead, alternative search methods rely on heuristics to efficiently explore the search tree [23] or samples to approximate plan quality [24].

Monte Carlo tree search methods combine reward sampling and exploration heuristics to perform online closed-loop planning efficiently [25]. Common heuristics for action selection during search trade-off exploitation with optimistic exploration, for example by using the estimated upper confidence bound of action-values [26]. However, large numbers of actions or state transitions hinder the search, as samples will tend to follow independent trajectories. One solution is to discretize actions and successor states. Another solution is to *progressively widens* the search tree, that is, to artificially limit the time spent in unpromising search areas by limiting the number of children of a tree node to a polynomial in its visit count [27]. Yet another solution is to guide search using heuristics learned from past searches. Heuristic learning in MCTS has been able to learn, from self-play, how to beat world experts in chess and Go [28].

So far, we have discussed methods for planning in MDPs with known transition and reward models. When transitions and rewards are unknown a priori, it may be possible to interface with a simulator that gives agents feedback about the results of their actions (*online reinforcement learning* [29]. Using online reinforcement learning to learn an offline policy may be more practical than online planning in large MDPs that are easy to simulate. When interfacing with a simulator is impractical, it may be possible to optimize plans using large amounts of collected data (*offline reinforcement learning*) [30].

2.2.3 *Alternative frameworks and solutions methods*

Markov decision processes provide one framework for planning under uncertainty. *Stochastic Shortest Path* (SSP) problems provide a more expressive framework that can be useful for modeling discrete-time, indefinite, fully observable decision-making problems with outcome uncertainty, and crucially, without discounting. An SSP augments an MDP with a definitive initial state s_0 and a goal state s_g that can eternally transition only to itself while accruing zero reward. Conveniently, all MDPs can be expressed as equivalent SSPs, though the reverse is untrue [31]. Optimal planning algorithms in SSPs yield the same solutions as value or policy iteration and policy iteration in the equivalent MDPs, but can sometimes converge much more quickly [32], [33].

Harmonic functions enable solutions to SSPs that are very useful for robotic motion planning, where a robot must quickly and smoothly navigate from one location to another while avoiding collisions [34]. Alternative techniques for fast and safe robotic path planning include probabilistic roadmaps (PRMs) and rapidly exploring random trees (RRT), which both sample the configuration space of a robot to find safe paths while avoiding collisions [35], and the dynamic window approach to restrict and score admissible actions [36].

2.3 *Partially observable Markov decision processes (POMDPs)*

Section 2.2 introduced the Markov decision process as a framework for modeling decision-making under outcome uncertainty. Though actions could induce a multiplicity of successor states, those states would be fully observed, giving the decision-making agent all of the necessary information to plan its subsequent actions.

In this chapter, we relax the assumption of full observability. Now, we assume that the agent only observes partial, imperfect information correlated to its state. As such, at any decision-making step, the agent's true state remains uncertain. Partial observability and the associated *state uncertainty* model many real-world systems. For instance, most autonomous systems have imperfect sensors that perceive noisy or incomplete state measurements. Additionally, when interacting with humans, autonomous systems may benefit by reasoning over those human's latent intentions, which can be modeled as partially observable states.

To frame planning under both outcome and state uncertainty, this thesis will use the partially observable Markov decision process (POMDP), which extends the MDP to partially observed states. This chapter will give a formal definition of the POMDP, outline notions of optimality, and overview different solution approaches.

2.3.1 *Formal definition*

The partially observable Markov decision process (POMDP) is defined as an extension of the MDP. Specifically, a POMDP model augments an MDP model with two additional components:

- **Observations**, $o \in \mathcal{O}$, are emitted by successor states after actuation.
- **Observation models**, $Z : \mathcal{A} \times \mathcal{S} \times \mathcal{O} \rightarrow \mathbb{R}_+$, encode correlations between states and observations using a Markov likelihood with which observations o are emitted from successor states s' after taking actions a. That is, $Z(a, s', o) = p(o \mid a, s')$.

In POMDPs, rather than perceive s', agents perceive observations o correlated with s', where those correlations are encoded with observation models. Similar to states and actions, observations can be either discrete or continuous. Similar to transition models, observation models over discrete observations must encode valid probability mass functions (i.e. $\sum_o Z(a, s', o) = 1$ for all a, s'), and observation models over continuous observations must encode valid probability density functions (i.e. $\int Z(a, s', o) do = 1$ for all a, s'). Again, we note that for the remainder of this section, expectations over transitions and observations are computed assuming discrete state and observation spaces, but can be easily extended to continuous state or observation spaces by replacing sums with integrals where appropriate.

Because an agent planning in a POMDP does not have access to the true state, it must instead plan based on the *history* of actions and emitted observations. Given an initial *belief*, or distribution over initial state $b_0(s) = p(s_0 = s)$, the history after having taken t actions and receiving t observations is written as $h_t = b_0 a_0 o_1 a_1 o_2 ... a_{t-1} o_t$. Generating and storing policies for any arbitrarily long history can be non-trivial. Fortunately, instantaneous belief over state is a sufficient statistic for history and can be calculated iteratively using belief updating, or *filtering*:

$$\begin{aligned} b_t(s) = p(s_t \mid h_t) &\propto p(o_t \mid a_{t-1}, s_t) p(s_t \mid a_{t-1}, h_{t-1}) && (2.8) \\ &\propto Z(a_{t-1}, s, o_t) \sum_{s' \in \mathcal{S}} T(s', a, s) b_{t-1}(s'). && (2.9) \end{aligned}$$

With discrete states, beliefs can be encoded as vectors on an $|S|$-dimensional probability simplex $\mathbf{b}_t \in \Delta^{|S|}$. These belief vectors can be updated exactly. In continuous state and observation spaces, linear Gaussian transitions and observations enable Kalman filtering [37] while approximations extend to nonlinear transitions and observations with unimodal uncertainty [38]. In a more general setting, particle filters use samples to represent and update beliefs [39], [40].

Because beliefs are sufficient statistics, it is common for POMDP policies π to map beliefs to actions. Again, rational policies in infinite-horizon discounted POMDPs optimize expected cumulative discount return $\mathbb{E}[\sum_{t=0}^{T} \gamma^t R(b_t, a_t)]$, where expected immediate rewards are computed over beliefs, that is, $R(b, a) = \sum_s b(s) R(s, a)$. POMDPs can be thought of as *belief-state MDPs*, a class of continuous MDP over belief-states with stochastic transitions that follow:

$$T(b, a, b') = \begin{cases} P(o \mid b, a) & \text{if } b' = b'_{a,o} \text{ for any } o \in \mathcal{O} \\ 0 & \text{otherwise,} \end{cases} \tag{2.10}$$

where $b'_{a,o}$ is the updated belief resulting from taking action a and perceiving o, and $P(o \mid b, a) = \sum_{s,s'} T(s, a, s') Z(a, s', o) b(s)$.

In a POMDP, the value of following a policy π from belief b is

$$V_R^{\pi}(b) = R(b, \pi(b)) + \gamma \sum_{o \in \mathcal{O}} P(o \mid b, \pi(b)) V_R^{\pi}(b'_{\pi(b),o}). \tag{2.11}$$

We can also define the action-value of starting in belief b, taking action a and then following π as

$$Q_R^{\pi}(b, a) = R(b, a) + \gamma \sum_{o \in \mathcal{O}} P(o \mid b, a) V_R^{\pi}(b'_{a,o}). \tag{2.12}$$

As with MDPs, an optimal policy π^* satisfies the Bellman optimality equations

$$V_R^*(b) = \max_{a \in \mathcal{A}} Q_R^*(b, a), \tag{2.13}$$

$$Q_R^*(b, a) = R(b, a) + \gamma \sum_{o \in \mathcal{O}} P(o \mid b, a) V_R^*(b'_{a,o}), \tag{2.14}$$

$$\pi^*(b) = \arg\max_{a \in \mathcal{A}} Q_R^*(b, a). \tag{2.15}$$

2.3.2 *Solution methods and extensions*

POMDPs are very challenging to solve. Exact solutions to finite-horizon POMDPs are PSPACE-complete in the problem size [41] while infinite-horizon POMDPs are undecidable [42]. Thankfully, POMDP algorithms use clever techniques to form tractable approximate solutions. This subsection will discuss methods for offline closed-loop planning, online open-loop planning, and online closed-loop planning, and will conclude by briefly discussing extension classes of POMDPs.

Offline approximate solutions rely on different techniques for representing the policy concisely. Point-based representations learn policies using values associated with a series of belief points that cover the reachable belief space. It is common to use *alpha vectors* to encode values and their gradients at belief points. The value function encoded by a set of alpha vectors Γ is piecewise linear and convex (PWLC) over beliefs,

$$V_R^\pi(b) = \max_{\boldsymbol{\alpha} \in \Gamma} \boldsymbol{\alpha}^\top \mathbf{b}, \tag{2.16}$$

while policies extract actions from the maximizing alpha vector at a belief.

Perseus builds a point-based policy through value iteration by performing randomized Bellman backups at beliefs sampled smartly from the reachable belief space [43]. HSVI maintains upper and lower value bounds and uses heuristic search to guide point-based backups [44]. SARSOP focuses backups near the optimally reachable belief set, sampled using predictions from related belief values [45]. These methods are limited to discrete state, action, and observation POMDPs of relatively small size.

Instead of representing values over points in the belief space, finite-state controllers (FSCs) represent offline policies as a directed graph, where nodes encode actions to take when control is relegated to a node, and edges encode how control nodes transition conditioned on observations. In discrete state, action, and observation POMDPs, deterministic finite-state controllers can be learned through policy iteration [46], [47], while stochastic, fixed-size finite-state controllers can be learned using nonlinear programming [48] or gradient ascent [49]. In large or continuous state spaces, FSCs can be learned using sampling-based techniques [50], [51].

Alternative approaches to offline POMDP planning represent and learn policies in a compressed subspace of beliefs. For example, value-directed compression (VDC) performs linear compression [52] while exponential family PCA finds a non-linear compression in the exponential family [53].

In online planning, under linear Gaussian dynamic and measurement models with quadratic rewards, the linear quadratic Gaussian controller (LQG) provides optimal closed-loop control by combining Kalman filtering with LQR. Under nonlinear Gaussian models, approximate Kalman filters can naturally be combined with iLQR for approximately optimal planning [54]. In many robotic applications, it often suffices to plan open-loop trajectories from the expected state using model predictive control. Additional work extends the linear quadratic regulator to belief-state planning, using iLQR jointly over the mean and covariance of a Gaussian filter while assuming maximum likelihood observations[55]. Just as in MDPs, in large problem domains, common random numbers can be used to optimize open-loop actions against determinized, action-dependent scenarios [19].

Finally, online closed-loop planning algorithms rely again on building an expectimax search tree, with maximization over actions and expectation over belief transitions, which are induced jointly by stochastic state transitions and observations. AEMS uses heuristics relying on upper and lower bounds to guide forward search [56]. POMCP extends MCTS to POMDPs with large state spaces by sampling states and assigning trajectories based on shared histories [57]. POMCPOW and PFT-DPW extend POMCP to continuous action and observation spaces by artificially limiting branching factors with node visit count [58]. VOMCPOW improves action sampling [59] using distances in action space. DESPOT simplifies searching in large state spaces by combining heuristic search against determinized scenarios that rely on common random numbers [60], a process which is extremely amenable to parallelization [61]. Additionally, recent approaches use information from previous searches to learn heuristics that can guide future ones, using neural networks [62], [63] or Gaussian Processes [64].

The POMDP model provides a basic framework for optimal decision-making under outcome and state uncertainty. It has many subclasses and extensions that are practical in different planning applications. For example, mixed-observability POMDP planning provides efficient solutions in problems where only a part of the state space is partially observable [65]. Rho-POMDPs define a belief-dependent reward function, which can be extremely favorable for adaptive sensing [66]–[68]. POMDP-lite provides an efficient solution when hidden states are constant or evolve deterministically [69]. In multi-agent planning, DecPOMDPs can model agents that must cooperate and plan in a decentralized fashion in order to optimize a shared reward [70]).

An important class of POMDP models planning under state and outcome uncertainty when an agent has to optimize competing objectives. We will discuss these multi-objective POMDPs (MO-POMDPs) in greater detail in Section 2.6. In particular, we will describe work around constrained POMDPs (CPOMDPs), as they provide a natural framework for expressing safety through constraints, pertain to our contributions in Chapters 4 and 5.

2.4 Imitation learning

Previous sections discussed optimal planning under outcome and state uncertainty. For plans to be considered optimal, they need to optimize well-defined objectives. However, planning objectives are not always well-defined. In extremely complex environments, it might instead be more practical to observe demonstrations of how other agents plan in order to mimic their decision-making. In this section, we will discuss *imitation learning*, the process of using data to overcome objective uncertainty by learning to imitate decisions made by experts.

2.4.1 Formal definitions

Planning objectives can be hard to specify, especially in real-world scenarios. Consider the case of autonomous driving. While we generally want vehicles to be effective, efficient, and safe, it is not necessarily clear how to encode rules for all complex interactions with other road participants into a single objective function. Reward misspecification can lead to undesired behavior [71]. Instead, in imitation learning, we wish to use trajectories of demonstrations to learn decision-making policies directly.

Recall that a trajectory aggregates an episode of T observations and actions, $\tau_T = s_0 a_0 s_1 a_1 ... s_T$. In certain applications, like autonomous driving, it may be much easier to collect trajectories than to encode decision-making objectives. Imitation learning methods extract policies from trajectories, and can largely be bucketed into one of three regimes:

1. In the **data-only** regime, the policy-maker only has access to demonstration trajectories.

2. In the **reward-free MDP (MDP\R)** regime, the policy-maker has access to demonstration trajectories alongside the true environment or a simulator of the environment (which is a reward-free MDP). The environment can be useful for testing a policy or simulating counterfactuals.

3. In the **queryable expert** regime, the policy-maker has access to demonstration trajectories, the environment, and an expert that they can query to generate new, expert demonstrations.

Swamy, Choudhury, Bagnell, *et al.* denote these as off-policy Q-value, on-policy reward, and on-policy Q-value *moment-matching* regimes respectively[72]. This arises from the view of imitation learning as a process of learning policies whose resulting trajectory distributions match moments against expert trajectory distributions [73], [74].

Each regime has its own advantages and pitfalls. The data-only regime is the simplest and most scalable, as it has no reliance on an environment simulator. However, policies trained in the data-only regime can generate cascading errors while driving systems towards never-before-seen states. Cascading errors are alleviated in the MDP\R regime, but training a policy in the true environment can be unsafe while creating a simulator of the environment can be costly, and worse, not capture all the nuances of reality (the simulation-to-reality, or *Sim2Real* gap). The queryable expert regime provides a path for safely training an imitative policy in the real world, but querying experts can be extremely inefficient and expensive.

In the data-only regime, *behavior cloning* describes the common approach of using demonstration trajectories to build policies that directly regress states to actions. For example, given a dataset of observed states and associated actions $\mathcal{D} = \{(s_i, a_i)_{i=1,\ldots,N}\}$, behavior cloning may optimize a stochastic policy conditioning actions on states by maximizing their likelihood under a parameterized policy π_θ, where π^* is parameterized by

$$\theta^* = \arg\max_\theta \prod_{(s,a)\in\mathcal{D}} \pi_\theta(a \mid s). \tag{2.17}$$

Additional work in the data-only regime uses information about state transitions within trajectories to match distributions over trajectories [74].

In Chapter 3, we will introduce a method that leverages hierarchies in the MDP\R regime, where a simulator can simulate counterfactuals in order to predict and avoid cascading errors. In the MDP\R regime, imitation learning convolves objective learning with policy optimization. *Inverse reinforcement learning* (IRL) describes the process of recovering a reward function from data, while *reinforcement learning* (RL) optimizes a policy given an objective function and an environment. In the MDP\R setting, imitation learning can be described as the convolution of these two steps: RL $\circ$ IRL.

Traditional methods alternate between inverse and forward reinforcement directly by inferring an MDP and solving that MDP. Abbeel and Ng infer a reward function as a linear set of features [75]. Later methods improve upon this by regularizing for policies that maximize trajectory entropy in unseen states [76], [77]. Framing this convolution problem as a two-player adversarial game, more recent methods learn policies directly by leveraging generative adversarial networks to train policies alongside discriminators that differentiate between expert and novice trajectories [78], [79]. Inverse Q-learning (IQ-Learn) performs this convolution non-adversarially by mapping policy optimization and reward inference together to a single action-value function to optimize [80].

2.4.2 *Applications and extensions*

Imitation learning has been applied for data-driven decision-making in many applications ranging from rotorcraft, fixed-wing aircraft, robotic manipulation, warehouse robotics, and home robotics [81], [82]. Facilitated by large datasets of human driving demonstrations [83]–[86], much research focuses on automated driving. The National Motor Vehicle Crash Causation Survey conducted from 2005 to 2007 concluded that 94% of serious accidents in the United States were caused by driver error [87]. Autonomous driving promises to alleviate these catastrophic errors. One significant challenge for decision-making in an autonomous car is scaling decision-making to handle the large number of rare but catastrophic scenarios seen on the road. As it is hard to predict and capture all of these events using rules, it can instead be practical and more easily scalable to learn how to imitate human driving.

The first demonstrations of autonomous road driving without special guides emerged in the late 1980s [88], [89], with early applications of behavior cloning attributed to Pomerleau[90]. More recent research has addressed imitation learning while querying human drivers to take over driving in dangerous situations [91]. Research in MDP\R imitation has imitated highway driving using GAIL[92].

A large body of research extends imitation learning to multi-agent settings, where the goal is to learn a set of policies that allow a team of agents to reproduce collaborative decision-making [93]. These works have also been applied to the joint control of multiple cars [94], [95]. Other extensions have explored inverse reinforcement learning as a method for autonomous agents to infer the objectives of collaborating humans, enabling shared autonomy [96].

Finally, additional extensions explore hierarchies, in order to decompose both decision-making and imitation. Many different approaches have investigated hierarchical autonomy systems where objectives for at least one layer are inferred using imitation learning. That imitation is typically done assuming labels of actions at the appropriate level of hierarchy [97]. Options-aware imitation learning jointly infers high-level and low-level policies by combining hierarchical reinforcement learning methods with expectation maximization to infer trajectory hierarchies [98].

We will overview hierarchical decomposition in the general context of decision-making under uncertainty in Section 2.5. In Chapter 3, we will introduce a hierarchical imitation learning method that relies on an environment simulator and safety prediction mechanisms to safely imitate human drivers in complex urban environments. Contemporaneously to our work, other works address safe hierarchical imitation learning for autonomous driving in urban environments by learning goal-conditioned low-level policies. Bronstein, Palatucci, Notz, *et al.* build a hierarchical policy in the reward-free MDP regime by conditioning low-level policies on high-level route roadgraphs that are known at train time [99]. Xu, Chen, Ivanovic, *et al.* build a hierarchical policy in the data-only regime by combining high-level goal prediction with low-level goal-conditioned trajectory generation, where goal-conditioned trajectories are optimized to be safe against behavior predictions of other road participants [100].

2.5 *Hierarchies*

Previous chapters have discussed optimal planning under uncertainty when successive actions are of the same fidelity. However, this is impractical for complicated tasks. Instead, it is more practical to consider planning with hierarchies of fidelity. For instance, rather than plan all muscle movements to *bake a cake*, it is more practical to first plan high-level actions (e.g. *collect all the ingredients*, *print a recipe*, *preheat the oven...*), and then plan low-level muscle movements for each action.

The first real-world applications of planning in robotics were organized in such a hierarchy. On the Shakey robot, PDP-10 programs were organized in a three-layer hierarchy. Programs in the lowest level drove all of the motors and captured sensory information, programs in the intermediate level supervised primitive actions, such as moving to a designated position and processing visual images, while programs in the highest level of the hierarchy would plan more complex actions [101]. This same kind of hierarchy underpins modern-day autonomous driving. At the highest level, search-based route planners plan routes. At middle levels, decision-making may choose appropriate waypoints and actions while considering other road participants. At the lowest levels, controllers plan sequences of actions, namely steering angles and accelerations, to smoothly actuate the autonomous car [102].

Hierarchical decomposition comes with a number of desirable benefits. First, it enables planning in large problems where planning at the lowest levels would be impossible. Additionally, it allows for interpretability in plans, helping users diagnose autonomous systems. Crucially, this enables runtime monitoring of plan execution, which enables redundancy in safety by allowing safety mechanisms to be included at multiple levels. Runtime supervision has been a critical enabler of autonomy [103].

The rest of this chapter overviews different methods for modeling hierarchical decomposition in MDPs. We begin by outlining the *options* framework, briefly discuss semi-Markov decision processes, and conclude by overviewing alternative frameworks for hierarchical decision-making.

2.5.1 *The options framework and Semi-Markov decision processes*

One popular framework for modeling hierarchies within a Markov decision process is through *options*. Options (alternatively called macro-actions or low-level controllers) encode a series of high-level actions that an agent can take that trigger lower-level policies. The options framework extends a Markov decision process with a set of options $\hat{\mathcal{A}}$. Each unique option $\hat{a} \in \hat{\mathcal{A}}$ is defined with the following:

- **Initiation states**, $\mathcal{I}_{\hat{a}} \subseteq \mathcal{S}$, encode all the valid states from which each option $\hat{a}$

can initiate.

- **Option policies**, $\pi^L_{\hat{a}} : \mathcal{S} \to \mathcal{A}$, describes the policy with which each option $\hat{a}$ selects actions.
- **Termination distributions**, $\beta_{\hat{a}} : \mathcal{S} \to [0, 1]$, describes the likelihood with which each option $\hat{a}$ terminates at each state.

In the options framework, a high-level policy π or π^H selects from different control options, so long as they can be initialized from the current state. That is, $\pi^H(s_t) = \hat{a} \implies s_t \in \mathcal{I}_{\hat{a}}$. Once an option is selected, it runs its policy until it reaches a state where it samples termination. Upon termination, control is returned to the high-level controller, which selects a new valid option. Underlying action primitives can also be included in the option set by encoding a policy that is guaranteed to run that action for only one step and return control. Additionally, the options framework can be used to model additional levels of hierarchy, for example, by calling mid-level options that can call low-level options.

In the options framework, we can say that options calls, or decisions, come in fixed epochs. The *decision epoch* associated with the e-th option call starts at time t_e and lasts for a *process time* of τ_e steps. The next epoch begins when the next decision takes place, at $t_{e+1} = t_e + \tau_e$.

The options framework induces two decision processes. While the low-level process of underlying actions follows the underlying MDP, the high-level process between successive options calls follows a discrete-time semi-Markov decision process (SMDP). An SMDP modifies an MDP with transitions that jointly model the successor state alongside the number of steps to get there. Additionally, it modifies the reward structure to account for path length. One way of representing this modification is as follows:

- **Transition models**, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{N} \to \mathbb{R}_+$ encode the joint probability of transition to successor states at different future points in discrete time. That is $T(s, a, s', \tau) = P(s', \tau \mid s, a)$.

- **Reward models**, $R : \mathcal{S} \times \mathcal{A} \times \mathbb{N} \rightarrow \mathbb{R}$, encode the expected discounted reward from a τ-step transition that starts in s after taking action a.

We can recover the high-level SMDP process from the options framework by mapping options to SMDP actions, low-level transitions and termination distributions to SMDP transitions, and expected discounted reward according to τ-step option policies to SMDP rewards.

2.5.2 *Alternative hierarchical frameworks*

While options will be the primary framework for modeling hierarchical decision-making in Chapters 3 and 5, many other frameworks can also model hierarchies in MDPs. For example, *hierarchical abstract machines* (*HAMs*) represent a hierarchy of policies as a set of machines, each with a set of machine states that follow a stochastic machine state transition distribution. Different types of machine states have different functions: *action* states execute a machine state-dependent action, *choice* states choose successor machine states, *call* states call a different machine, stepping one level down in the hierarchy, while *stop* states terminate the machine and step up one level of the hierarchy [104].

Another approach, *MAXQ*, represents a hierarchical policy as a directed acyclic graph of admissible subroutines, with leaves representing primitive actions. Crucially, MAXQ does not assume knowledge about the best choice of subroutine, only that they are available to an agent in a routine. By encoding that knowledge, planning with MAXQ can efficiently attribute values to multiple routines in order to scale both learning and planning [105].

Many of the ideas behind options, HAMs, and MAXQ were adopted from earlier research on *hierarchical task networks* (HTN) in classical planning. HTNs encode hierarchies by first describing high-level, broad actions, and then expanding those actions into lower-level refinements. That way, a STRIPS-style planning algorithm can first do an easier search over broad actions, and then gradually undertake each action by planning over its refinements in higher levels of detail. These ideas became prevalent through planners such as ABSTRIPS [106], NOAH [107], INTERPLAN [108], NONLIN [109], and O-PLAN [110].

2.6 Constrained POMDPs (CPOMDPs)

In previous sections, we outlined principled frameworks for rational decision-making under outcome, state, and objective uncertainty. We also saw how hierarchies can simultaneously scale decision-making while improving safety by ensuring redundancy against failure modes. In this section, we will overview methods for planning under uncertainty when balancing multiple objectives, and discuss how to achieve safety in the plans themselves by imposing constraints.

2.6.1 Multi-objective planning

In previous sections, we have discussed methods to optimize plans against a single objective function. However, many real-world problems are multi-objective. This is particularly true in safety-critical systems, where we want to optimize planning performance alongside safety. Multi-objective, (partially observable) Markov decision processes (MO-(PO)MDPs) are an important class of (PO)MDP that formalize planning with multiple objectives. These objectives are typically encoded by augmenting (PO)MDP models with additional reward functions $\mathbf{R}$ and associated value functions $\mathbf{V}_R$ [111].

Policies can trade off the importance of different objectives in different ways. If a policy optimizes at least one objective, it is said to be *Pareto optimal*. The set of Pareto optimal policies forms a *Pareto frontier* of optimal solutions [112]. Policies not on the Pareto frontier are suboptimal, as they are dominated by policies that have better performance on at least one objective.

Different techniques target different policies on the Pareto frontier. One straightforward technique to balance the trade-offs between competing objectives is to scalarize them into a single reward function to optimize using single-objective (PO)MDP planners. For example, linear scalarization may form $R(s, a) = \boldsymbol{\lambda}^\top \mathbf{R}(s, a)$ for some $\boldsymbol{\lambda}$. Scalarization may require expert knowledge about tradeoffs between objectives, and, depending on the scalarization routine, may not retrieve every Pareto-optimal policy.

Another approach uses lexicographies (e.g. L(PO)MDPs) to define an ordering of objectives to optimize [113]. Given this ordering, an L(PO)MDP solver will optimize successive objectives only as long as the former objectives stay close to their optimal value. On the other hand, constrained (PO)MDPs will optimize a single objective subject to constraints on other objectives [114]. Topological (PO)MDPs provide a general framework for representing relationships between objectives but will remain out of scope for this thesis [115].

In the next sections, we will formalize planning in constrained MDPs and POMDPs, as constraints allow for a natural framework to express safety and pertain to our contributions in Chapters 4 and 5.

2.6.2 CPOMDP formal definition

Constrained MDPs and constrained POMDPs augment MDP and POMDP models with a set of constraints that a feasible policy must satisfy. The most common expression of these constraints is through a set of cost functions that must satisfy a cost budget in expectation when accumulated. That is, C(PO)MDP models augment (PO)MDP models with the following:

- **Cost models**, $\mathbf{C} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k_+$, describe the k instantaneous costs accrued for taking actions, encoding alternative objectives for agent planning.

- **Constraint budgets**, $\hat{\mathbf{c}} \in \mathbb{R}^k_+$, describe the k constraint budgets that accumulated costs must satisfy for a solution to be feasible.

In infinite-horizon discounted CMDPs or CPOMDPs, it is useful to define the *cost value* of starting in a state s or belief b and following policy π using the Bellman equations presented in Equations (2.2) and (2.13), as well as the cost action-values when first taking action a:

$$\mathbf{V}_C^\pi(s) = \mathbf{C}(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s')\mathbf{V}_C^\pi(s'), \tag{2.18}$$

$$\mathbf{V}_C^\pi(b) = \mathbf{C}(b, \pi(b)) + \gamma \sum_{o \in \mathcal{O}} P(o \mid b, \pi(b))\mathbf{V}_C^\pi(b'_{\pi(b),o}), \tag{2.19}$$

$$\mathbf{Q}_C^\pi(s, a) = \mathbf{C}(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s')\mathbf{V}_C^\pi(s'), \tag{2.20}$$

$$\mathbf{Q}_C^\pi(b, a) = \mathbf{C}(b, a) + \gamma \sum_{o \in \mathcal{O}} P(o \mid b, a)\mathbf{V}_C^\pi(b'_{a,o}). \tag{2.21}$$

Given these definitions, we can define a C(PO)MDP policy as feasible if it satisfies the cost constraints from an initial state or belief, and as optimal if remains feasible while maximizing expected discounted cumulative rewards, that is,

$$\pi^* = \underset{\pi \in \Pi}{\arg\max} \quad V_R^\pi(x) \tag{2.22}$$

$$\text{such that} \quad V_{C_k}^\pi(x) \leq \hat{c}_k \ \forall\, k, \tag{2.23}$$

where x is the initial state for CMDPs and the initial belief for CPOMDPs. This constrained optimization problem can be represented equivalently through its Lagrangian:

$$\pi^* = \underset{\pi \in \Pi}{\arg\max} \min_{\boldsymbol{\lambda} \geq 0} V_R^\pi(x) - \boldsymbol{\lambda}^\top (\mathbf{V}_C^\pi(x) - \hat{\mathbf{c}}). \tag{2.24}$$

Unlike in (PO)MDPs, the optimal policy in C(PO)MDPs can be stochastic. With k constraints, an optimal policy may optimize reward while satisfying constraints in expectation by mixing policies with up to $k+1$ actions at each state. The intuition is that riskier actions that violate constraints to yield larger rewards can be balanced with safer actions to satisfy constraints. With a stochastic policy, we compute costs, rewards, and probabilities in expectation, for example,

$$\mathbf{C}(b, \pi(b)) = \sum_s b(s)\mathbf{C}(s, \pi(s)) = \sum_{s,a} b(s)\pi(a \mid s)\mathbf{C}(s, a). \tag{2.25}$$

Finally, we note that in online planning for receding horizon control, we update the constraint budget at each time step, with the motivation that future plans should depend on the remaining budget rather than the initial budget. To show this, consider the constraints at any point in time in the partially observable setting:

$$\mathbb{E}\left[\sum_{\tau=t}^{\infty}\gamma^{\tau-t}\mathbf{C}(b_\tau,a_\tau)\right]\leq\hat{\mathbf{c}}_t \tag{2.26}$$

$$\mathbb{E}\left[\mathbf{C}(b_t,a_t)\right]+\gamma\mathbb{E}\left[\sum_{\tau=t+1}^{\infty}\gamma^{\tau-t-1}\mathbf{C}(b_\tau,a_\tau)\right]\leq\hat{\mathbf{c}}_t \tag{2.27}$$

$$\mathbb{E}\left[\sum_{\tau=t+1}^{\infty}\gamma^{\tau-t-1}\mathbf{C}(b_\tau,a_\tau)\right]\leq\frac{\hat{\mathbf{c}}_t-\mathbb{E}\left[\mathbf{C}(b_t,a_t)\right]}{\gamma}, \tag{2.28}$$

meaning that planning with $\hat{\mathbf{c}}_{t+1}=[\hat{\mathbf{c}}_t-\mathbb{E}[\mathbf{C}(b_t,a_t)]]^+/\gamma$ preserves constraint satisfaction while ensuring a positive constraint budget at every step.

Next, we will overview different solution methods for CPOMDPs, as they pertain to our contributions in Chapters 4 and 5.

2.6.3 *Solution methods*

In CMDPs with small numbers of discrete states and actions, one can optimize Equations (2.22) and (2.23) as a linear program. Additional methods for optimizing CMDPs are discussed by Altman [116]. The field of stochastic model predictive control investigates receding-horizon open-loop planning in continuous CMDPs, often with unimodal transition uncertainty [18].

Methods for offline planning in constrained POMDPs have been restricted to discrete state, action, and observation spaces. Solution methods to generate point-based policy representations include dynamic programming [114], [117] and approximate linear programming [118]. Solution methods that generate policy graphs or finite state controller policies include column generation [119] and projected gradient ascent [120].

Online planning algorithms can generate better solutions by searching across immediately reachable beliefs. Undurti and How combine forward search with a heuristic generated from offline planning to prune unsafe subtrees [121]. CC-POMCP [122] plans online in extremely large state spaces by performing Lagrangian-guided partially observable Monte Carlo tree search [57], with dual ascent to optimize the Lagrange multipliers. As MCTS can have high variance, recent work uses information from past searches to learn a safety heuristic to help guide safe search [123].

Alternative works model safety using chance constraints rather than expected cost constraints. Chance constraints model distributional robustness by imposing constraints on the distribution of cost values, for example, that $\mathbb{P}(V_{C_k}^{\pi} > \hat{c}_k) < \alpha$. RAO* models chance-constrained in offline closed-loop POMDP planning, where failures are defined as the inclusion of unsafe states in a trajectory [124]. Chen, Shimizu, Sun, *et al.* optimizes open-loop plans in continuous POMDPs with per-time-step collision-avoidance chance constraints [125]. Alternative methods for safe planning under uncertainty optimize robust metrics of return distributions, for example, value-at-risk – a particular quantile of returns – or conditional value-at-risk – the expectation of returns inside a quantile [126], [127].

3 Hierarchical Imitation Learning

We begin by considering safe decision-making under outcome and objective uncertainty with the case study of autonomous driving. Designing a safe and human-like decision-making system for an autonomous vehicle is a challenging task for reasons described in Section 2.4. Generative imitation learning is one possible approach for automating policy-building by leveraging both real-world and simulated decisions. Previous work that applies generative imitation learning to autonomous driving policies focused on learning a low-level controller for simple settings. However, to scale to complex settings, many autonomous driving systems combine fixed, safe, optimization-based low-level controllers with high-level decision-making logic that selects the appropriate task and associated controller.

In this chapter, we attempt to scale imitation learning to complex, safety-critical environments by introducing Safety-Aware Hierarchical Adversarial Imitation Learning (SHAIL). Leveraging frameworks for hierarchical decomposition described in Section 2.5, SHAIL learns a high-level policy that selects from a set of low-level controller instances in a way that imitates low-level driving data on-policy. We introduce an urban roundabout simulator that controls non-ego vehicles using real data from the Interaction dataset. We then demonstrate empirically that even with simple controller options, our approach can produce better behavior than previous approaches in driver imitation that have difficulty scaling to complex environments.

The work presented in this chapter is based on a research collaboration with Etienne Buehrle and Johannes Fischer [128]. Our implementations are available at `https://github.com/sisl/InteractionImitation`.

3.1 Introduction

The development of autonomous vehicles will greatly impact urban traffic. Of particular importance is the safety and predictability of autonomous vehicles when interacting with complex environments. Achieving safe and human-like behavior will require a) multiple levels of safety redundancy, b) large amounts of real, "expert" driving data, and c) advanced simulators to test behavior before deploying.

Recent reinforcement learning approaches add levels of redundancy to policies learned in simulation by allowing for a hierarchy of control that passes between high-level action selectors and safe low-level optimization-based driving controllers [129], [130]. Though the addition of hierarchical safety layers is intuitive and adds levels of redundancy, the success of any reinforcement learning-based approach hinges on the design of the reward function. A misspecified reward function can be catastrophic.

To resolve the issue of reward misspecification, imitation learning approaches instead rely on demonstrations from an expert in the environment. Data availability invites the use of imitation learning methods that do not interact with the environment (i.e. off-policy methods, such as behavior cloning) [90]. However, these methods suffer from cascading errors when vehicles encounter out-of-distribution states [131]. Some on-policy approaches will query an expert to help guide the learning process safely [131], but querying an expert can be costly or impractical. Adversarial imitation learning approaches have been applied with simulators on-policy to circumvent the need for a queryable driving expert [92], [95], but these approaches have mostly been tested in simple driving settings and are still not collision-free.

We approach the safety and environment simplicity limitations of these prior applications of adversarial imitation learning to autonomous driving by taking a hierarchical approach. We note that many autonomous driving systems combine fast, safe, optimization-based controllers for low-level control with high-level logic to select appropriate tasks, controllers, and controller parameters. High-level logic might choose between different options (e.g. `LaneChangeLeft`, `Accelerate`, `TurnRight`, `EasyBrake`, `HardBrake`), then pass control to an instance of a low-level controller

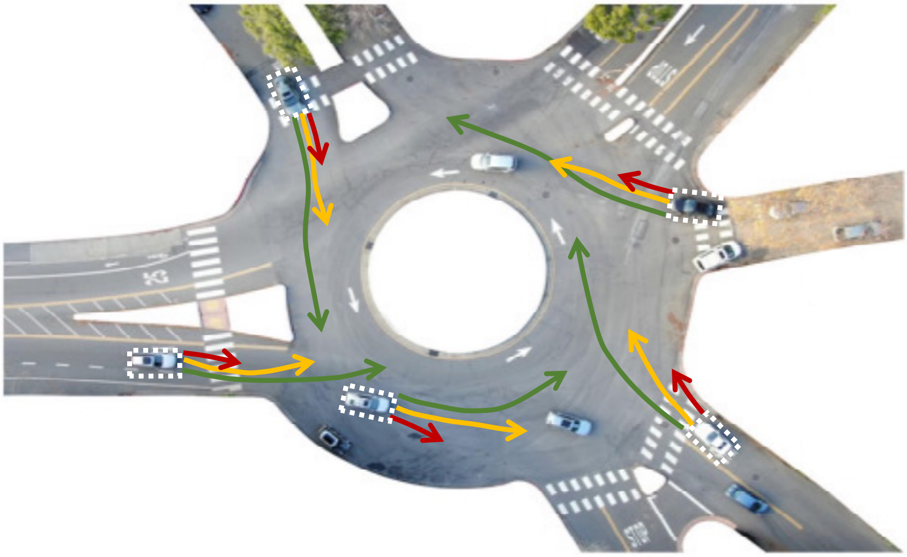


Figure 3.1: With SHAIL, the ego vehicle learns to choose from a set of safe high-level options to navigate a complex driving environment derived from the Interaction dataset [84]. The learner requires only low-level expert states and actions as opposed to high-level actions or a reward function.

with the appropriate task and parameters for the chosen option. However, labels for these high-level choices are typically inaccessible in expert trajectories, making direct learning difficult.

A large body of work exists around hierarchical imitation learning formulations for different robotics problems [97], [98], [132]–[135]. In this paper, we employ a method for learning a high-level controller-selection policy that imitates low-level driving data on-policy given a set of *known* low-level controller instances, as is appropriate for autonomous driving. A depiction of our problem setting is shown in Figure 3.1. We introduce Safety-Aware Hierarchical Adversarial Imitation Learning (SHAIL), which maintains the same low-level occupancy measure-matching objective of previous adversarial imitation learning approaches applied to driving [73], [92], but assumes that low-level data is generated within the options framework [136] and reformulates the objective accordingly. Additionally, SHAIL implements a safety awareness layer to adjust the high-level controller selection policy based on active reasoning about the safety or feasibility of different options.

To demonstrate the effectiveness of SHAIL, we develop a simulator based on real driving data from complex urban driving scenarios in the Interaction dataset [84]. Our simulator allows us to adjust the acceleration of an ego vehicle along its real path while other agents in a scene behave according to data. We test a basic implementation of SHAIL in roundabouts, a dynamic driving scenario that is typically difficult for an autonomous vehicle to safely navigate. We compare SHAIL against an IDM adaptation, behavior cloning (BC), non-hierarchical generative adversarial imitation learning (GAIL), and an ablation of SHAIL without the safety layer (HAIL), observing that SHAIL indeed yields safer and more realistic driving behavior.

In summary, the main contributions of this paper are to:

- Introduce SHAIL, a methodology for learning a safe high-level action-selection policy that imitates low-level observations and actions,
- Introduce a simulator for complex driving scenarios based on real data, and,
- Empirically demonstrate the efficacy of SHAIL compared to IDM, behavior cloning, and non-hierarchical imitation learning.

The remainder of this chapter will outline background and related work, describe the methodology for hierarchical adversarial imitation learning with safety constraints, present our experiments and results, and conclude with a brief discussion.

3.2 *Background*

This section provides necessary background on reinforcement learning, imitation learning, and hierarchical planning.

3.2.1 *Reinforcement and Imitation Learning*

As discussed in Section 2.2, optimal decision-making under uncertainty can be framed in the context of Markov Decision Processes (MDPs). With discrete states, an MDP can be defined as a tuple $\langle \mathcal{S}, \mathcal{A}, T, R, b_0, \gamma \rangle$ and includes a finite state space $\mathcal{S}$, the action space $\mathcal{A}$, a stochastic transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, an initial state distribution $b_0 : \mathcal{S} \to [0, 1]$, and a positive discount factor $\gamma < 1$. An MDP policy maps states to a distribution over actions to take $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. An optimal policy maximizes expected cumulative discounted reward, $\pi^* \in \arg\max_{\pi \in \Pi} \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 \sim b_0(\cdot)]$.

In the reinforcement learning setting, the exact transition and reward functions T and R are unknown, but we can interact with an environment to receive generated samples of next state and reward $s', r \sim G(s, a)$. There is a body of work in which reinforcement learning is used to generate policies for different autonomous driving scenarios [137]–[139]. These works requires the use of a driving simulator to produce realistic transitions, as well as the manual specification of a reward function. Designing a reward function to capture all desired behavior is extremely difficult, and it is common for learning agents to exploit misspecified reward functions [71].

In the imitation learning setting, instead of receiving a reward signal, we rely on data in the form of trajectory rollouts from an expert who interacts with the environment. The imitation learning problem can be viewed as a problem of moment-matching between the expert and learner distributions, and methods can broadly be characterized as seeking to match Q-value moments off-policy, Q-value moments on-policy, or reward moments on-policy [72]. In off-policy Q-value moment matching, the learned imitating policy cannot interact with the environment until execution time [74]. The most straightforward approach to learn a policy in this setting is through behavior cloning (BC), in which a supervised learner regresses states to actions. This approach has a long history in autonomous driving systems [90], [140].

Behavior cloning suffers from an accumulation of errors during testing as an agent ends up in states it has not seen during training, a phenomenon often referred to as covariate shift [131]. In the on-policy Q-value-matching setting, this accumulation of errors is reduced by introducing a query-able expert who can correct deviating trajectories during training [131]. However, training with a query-able expert can be costly, timely, and impractical. In contrast, on-policy reward moment-matching algorithms assume no access to a query-able expert, but still assume interactions with the environment during training (e.g. using a simulator).

The state-action occupancy measure under a policy π is the (unnormalized) γ-discounted stationary distribution of states and actions visited under that policy, $\rho^{\pi}(s,a) = \pi(a \mid s)\rho^{\pi}(s)$, where $\rho^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid \pi)$. We can similarly define the state-action occupancy measure of the expert policy, $\rho^{\text{exp}}(s,a)$. One perspective formulates imitation learning as moment-matching between expert and learned occupancy measures, done by minimizing some f-divergence between the associated distributions $\min_{\pi} D_f((1-\gamma)\rho^{\text{exp}}(\cdot,\cdot) \| (1-\gamma)\rho^{\pi}(\cdot,\cdot))$ [73]. In the on-policy reward matching setting, this objective can be written as a two-player game between a policy generator π_θ and an observation-action discriminator D_ϕ:

$$\min_{\pi_\theta} \max_{D_\phi} \mathbb{E}_{(s,a)\sim\rho^{\text{exp}}(\cdot,\cdot)}[D_\phi(s,a)] - \mathbb{E}_{(s,a)\sim\rho^{\pi_\theta}(\cdot,\cdot)}[f^*(D_\phi(s,a))], \tag{3.1}$$

where f^* denotes the convex conjugate of f. This objective can be optimized by alternating between discriminator gradient ascent steps to optimize the discriminator parameters ϕ and policy gradient ascent steps to optimize the parameters θ of a stochastic policy. This latter step can be viewed as reinforcement learning with an 'imagined' reward signal of $r(s,a) = f^*(D_\phi(s,a))$. These steps use Monte Carlo methods (and a replay buffer) to estimate the expectations [78], [79].

These generative methods have been used to imitate highway driving behavior [92]. Later work improves upon this by augmenting the learned reward model with soft constraints to avoid bad states and actions [95]. Two shortcomings of these works are that they a) mostly consider highway driving, a relatively simple driving scenario, and b) only learn low-level controllers, for which safety is more difficult to guarantee in comparison to traditional optimization-based controllers.

3.2.2 Hierarchical Planning

Human driving in complex environments is naturally hierarchical. As discussed in Section 2.5, one can model hierarchical planning with options [136], where a low-level controller $\hat{a}$ is chosen from a finite set of options $\hat{\mathcal{A}}$ and executed until termination, upon which a new valid option is chosen. In addition to the components of an MDP, an options model over a discrete state space defines K options (indexed by $\hat{a}$) which each define a set of states from which they can be initialized $\mathcal{I} \subseteq \mathcal{S}$, a low-level control policy $\pi^L : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, and the probability that the option will be terminated from any given next state $\beta : \mathcal{S} \rightarrow [0, 1]$. A high-level policy over options denotes the probability of choosing from the valid set of options in any given state $\pi^H(\hat{a} \mid s)$, where $s \in \mathcal{S}$ and $\hat{a} \in \hat{\mathcal{A}}$ such that $s \in \mathcal{I}_{\hat{a}}$.

Recent work considers hierarchical reinforcement learning for planning in driving scenarios [129], [130], [141]. While still suffering from the pitfalls of a manually specified reward function, these approaches have the benefit that a high-level action-selector can hand over control to safe, low-level, optimization-based planners. Mirchevska, Hügle, Kalweit, *et al.* use this approach to learn a high-level controller that can choose safe gaps in highway traffic for an optimization-based low-level controller to navigate to [129]. In their approach, the high-level controller only targets reachable gaps, while if a targeted gap no longer is reachable during low-level execution, control is passed back to the high-level controller.

Additional work considers hierarchical approaches to imitation learning. For example, Henderson, Chang, Bacon, *et al.* perform imitation learning hierarchically (as opposed to hierarchical imitation learning) by learning multiple generators and discriminators to match low-level data, and learning a mixture-of-experts policy over those generators to follow [142]. Le, Jiang, Agarwal, *et al.* describes a formulation to perform hierarchically-guided behavior cloning and dataset aggregation, however, this assumes labeling of the high-level option, which we do not [97].

Jing, Huang, Sun, *et al.* perform hierarchical on-policy reward moment-matching by framing an objective to match the moments over states, actions, and options and alternating between expert option label inference and joint policy training. In this work, we keep low-level options fixed, as is more appropriate for driving. As a result, our work is in line with moment-matching objectives over state and action [73] as opposed to those additionally over options [98]. This can be viewed as a subclass of Jing, Huang, Sun, *et al.* [98] where there is no need to infer latent options in the data, or as an extension of Ghasemipour, Zemel, and Gu [73] in which the state-action pairs are drawn hierarchically.

Much of recent work that performs vehicle behavior prediction hierarchically (e.g. [143]) can be easily extended to off-policy hierarchical imitation learning, as many policies learned to predict vehicle behavior when conditioned on a goal could be extended to control an ego vehicle. A flavor of on-policy hierarchical imitation learning has been applied to driving policies, in which long-horizon planning learned to mimic a query-able expert is interleaved with fast, short-horizon, low-level optimal control [144]. In contrast, we propose learning a high-level controller on-policy that imitates low-level data without a query-able expert, and characterize a broader class of high-level control options.

3.3 Methodologies

This section formulates SHAIL, first by reformulating the occupancy measure-matching objective for a policy that generates low-level data hierarchically, and then by designing a safety-aware high-level controller.

3.3.1 Hierarchical Adversarial Imitation Learning

We first formulate the occupancy measure-matching objective in Equation (3.1) for a policy that is generating states and actions hierarchically. We do this by expanding the occupancy measure over options that would lead to state s and action a during their execution, and states in which the options begin executing. We expand over the initiation states $s_\tau = h$ that begin executing the options $\hat{a}$ at time τ under which low-level states and actions s and a can be observed at time t:

$$\begin{aligned}
\rho^{\pi}(s,a) &= \sum_{t=0}^{\infty} \gamma^t P(s_t = s, a_t = a) && (3.2)\\
&= \sum_{h,\hat{a}} \sum_{t=0}^{\infty} \sum_{\tau=0}^{t} \gamma^{\tau} P(s_\tau = h, \hat{a}_\tau = \hat{a}) \gamma^{t-\tau} P(s_t = s, a_t = a \mid s_\tau = h, \hat{a}_\tau = \hat{a}) && (3.3)\\
&= \sum_{h,\hat{a}} \sum_{\tau=0}^{\infty} \gamma^{\tau} P(s_\tau = h, \hat{a}_\tau = \hat{a}) \sum_{t=\tau}^{\infty} \gamma^{t-\tau} P(s_t = s, a_t = a \mid s_\tau = h, \hat{a}_\tau = \hat{a}) && (3.4)\\
&= \sum_{h,\hat{a}} \sum_{\tau=0}^{\infty} \gamma^{\tau} P(s_\tau = h, \hat{a}_\tau = \hat{a}) \sum_{t=0}^{\infty} \gamma^{t} P(s_t = s, a_t = a \mid s_0 = h, \hat{a}_0 = \hat{a}) && (3.5)\\
&= \sum_{h,\hat{a}} \rho^{\pi^H}(h,\hat{a}) \rho^{\pi^L}(s,a \mid h,\hat{a}), && (3.6)
\end{aligned}$$

where $\rho^{\pi^H}(h,\hat{a}) = \pi^H(\hat{a} \mid h) \sum_{\tau=0}^{\infty} \gamma^{\tau} P(s_\tau = h)$ is the discounted occupancy measure for ending up in a high-level state h and initiating option $\hat{a}$, and $\rho^{\pi^L}(s,a \mid h,\hat{a}) = \pi^L_{\hat{a}}(a \mid s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid s_0 = h, \hat{a}_0 = \hat{a})$ is the discounted occupancy measure of states and actions under an option $\hat{a}$ which was initialized in state h at time 0.

We apply this hierarchical representation of occupancy measure $\rho^{\pi}(s,a)$ to reformulate the measure-matching objective in Equation (3.1) for policy data that is generated hierarchically:

$$\begin{aligned}
&\min_{\pi} \max_{D_\phi} \mathbb{E}_{(s,a)\sim\rho^{\exp}(\cdot,\cdot)}[D_\phi(s,a)] - \sum_{s,a} \rho^{\pi}(s,a) r(s,a) && (3.7)\\
=&\min_{\pi} \max_{D_\phi} \mathbb{E}_{(s,a)\sim\rho^{\exp}(\cdot,\cdot)}[D_\phi(s,a)] - \sum_{s,a,h,\hat{a}} \rho^{\pi}(s,a \mid h,\hat{a}) \rho^{\pi}(h,\hat{a}) r(s,a) && (3.8)\\
=&\min_{\pi^H_\theta} \max_{D_\phi} \mathbb{E}_{(s,a)\sim\rho^{\exp}(\cdot,\cdot)}[D_\phi(s,a)] - \sum_{h,\hat{a}} \rho^{\pi^H_\theta}(h,\hat{a}) \sum_{s,a} \rho^{\pi^L}(s,a \mid h,\hat{a}) r(s,a) && (3.9)\\
=&\min_{\pi^H_\theta} \max_{D_\phi} \mathbb{E}_{(s,a)\sim\rho^{\exp}(\cdot,\cdot)}[D_\phi(s,a)] - \mathbb{E}_{(h,\hat{a})\sim\rho^{\pi^H_\theta}(\cdot,\cdot)}[\tilde{r}(h,\hat{a})], && (3.10)
\end{aligned}$$

where $\tilde{r}(h,\hat{a}) = \mathbb{E}_{(s,a)\sim\rho^{\pi^L}(\cdot,\cdot\mid h,\hat{a})}[f^*(D_\phi(s,a))]$.

Optimizing this objective can be done in a fashion similar to optimizing the objective in Equation (3.1). Discriminator updates remain identical, while generator updates require performing policy gradients on $\pi_\theta^H(\hat{a} \mid h)$ where the new 'imagined' high-level reward $\tilde{r}(h, \hat{a})$ accumulates the discounted low-level 'imagined' discrimination rewards from the execution of the chosen option. That is, $\tilde{r}(h, \hat{a})$ can be estimated as $\sum_{t=0}^{T_{\hat{a}}} \gamma^t r(s_t, a_t)$, where $s_0 = h, a_t \sim \pi_{\hat{a}}^L(\cdot|s_t)$, and $T_{\hat{a}}$ is the duration of the option.

3.3.2 *Safety-Aware Hierarchical Adversarial Imitation Learning*

Many practical implementations of policy gradients rely on a fixed-size action space [145], [146]. Because of this restriction, we are limited to an option set where any option can be initialized from every state, i.e. $\mathcal{I}_{\hat{a}} = \mathcal{S}$ for all $\hat{a} \in \hat{\mathcal{A}}$. This assumption can be very limiting in terms of safety. Oftentimes, we have information about restricted options from different states (e.g. an `Accelerate` option should not be taken from a red light). Additionally, we might be able to make predictions about the safety of different controllers. For example, this can be done strictly with formulations of reachability of a controller, or more loosely through notions of scene understanding (e.g. 'it is probably unsafe to make a turn since there are vehicles crossing the intersection'). SHAIL improves upon the formulation of hierarchical adversarial imitation learning presented in the previous section by designing a high-level option selection policy that incorporates sensitivity to option safety.

Safety awareness is incorporated by assuming that the agent can reason about the safety or availability of different options from different states. We introduce a binary random variable z which predicts the safety or availability of a low-level controller, denoting the probability that an option $\hat{a}$ is safe when executed from a high-level state s as $p_{\text{safe}}(z^1 \mid s, \hat{a})$. This allows us to design the options such that control is passed back to the high-level option selector according to this safety prediction, i.e. $\beta_{\hat{a}}(s) = 1 - p_{\text{safe}}(z^1 \mid s, \hat{a})$. This option termination formulation expands on the one used by Mirchevska, Hügle, Kalweit, *et al.* in the hierarchical reinforcement learning setting [129] to admit probabilistic controller safety predictions rather than binary controller availability.

With this formulation, we additionally design a high-level controller that conditions on controller safety:

$$\pi_\theta^H(\hat{a} \mid s, z^1) \propto p(\hat{a}, z^1 \mid s) = p_{\text{safe}}(z^1 \mid s, \hat{a}) \psi_\theta^H(\hat{a} \mid s), \tag{3.11}$$

where ψ_θ^H is a learnable controller selector. This high-level controller reweights options based on predictions of their safety or availability. It can be easily shown that learning with this substituted policy is equivalent to minimizing the divergence to an agent occupancy measure conditioned on safety, $\rho^\pi(s, a \mid z^1)$. This scheme requires at least one option with nonzero safety probability (e.g. a permanent 'safe' controller), otherwise, the high-level policy will not represent a valid distribution over controllers. Additionally, to learn a useful option selector, the options should have some semantic meaning that holds across different initialization states.

Learning ψ_θ^H with policy gradients on this policy formulation requires storing the safety probabilities seen during option initiation in the replay buffer. That is, for each option $\hat{a}$ initiated from a state h, the replay buffer consists of samples of the form $(h, \hat{a}, p_{\text{safe}}(z^1 \mid h, \cdot), \tilde{r}(h, \hat{a}))$.

3.4 Experiments

Our experiments demonstrate the effective use of SHAIL in a driving simulator. We introduce our own simulator based on real data in urban driving environments, and demonstrate the improvements regarding safety that can be achieved in comparison with baseline models, even by a simple SHAIL implementation.

3.4.1 Simulator

To test this approach in a more complicated driving environment, we introduce the Interaction simulator.[1] The Interaction simulator is an OpenAI Gym [147] simulator that uses underlying data from the Interaction dataset of complex urban driving

[1] `https://github.com/sisl/InteractionSimulator`

scenes [84]. The dataset consists of recorded track files from driving scenarios in different urban driving situations like roundabouts or intersections.

The simulator itself fixes vehicle paths and spawn times based on the recorded data in the Interaction dataset, and admits control of vehicle accelerations along the path. This is a reasonable navigation strategy, as it may be common for a separate module to determine the path of a vehicle navigating a complex scene.

In our experiments[2], we focus on controlling a single vehicle that is modeled with double integrator dynamics and moves along its recorded path while non-ego vehicles follow their recorded trajectories. Simulations are terminated when the vehicle leaves the scene.

We assume that the ego vehicle encodes its absolute velocity, yaw rate, and lidar-like measurements of the relative position and velocity of the closest vehicle in each 72° sector of its surroundings. We use this very simple subset of autonomous vehicle features to avoid overfitting to our small dataset.

3.4.2 *Models*

We evaluate the following baseline models in our experiments:

Expert: The expert model uses the default accelerations from the Interaction dataset.

IDM: The Intelligent Driver Model (IDM) models vehicle accelerations in fixed-lane driving when following a vehicle [148]. This model is problematic for uncontrolled driving, where it is not clear which vehicle the ego should 'follow'. We choose the follow vehicle as the closest vehicle that lies within two meters of the ego's planned path and has less than a 30° difference in heading. We target a desired speed of 8.94 m/s (20 mph), a minimum spacing of 3 m, a desired time headway of 0.5 s, a nominal acceleration of 3 m/s^2, and a comfortable braking deceleration of 2.5 m/s^2.

BC: We implement a behavior cloning agent that directly regresses features to a mean and standard deviation parameterizing a normal distribution for ego vehicle

[2] `https://github.com/sisl/InteractionImitation`

acceleration. Our model is a feedforward neural network with layers of fixed hidden size. We train our model by minimizing the negative log-likelihood of expert actions under the action distribution predicted from their preceding states.

GAIL: We compare against a model learned through Generative Adversarial Imitation Learning (GAIL) [78], [92]. Both discriminator and policy models are feedforward neural networks, the latter again outputting parameters for a normal distribution over next action. We use the same optimization objective as Ho and Ermon, as well as proximal policy optimization (PPO) [145] to learn a policy. We do not compare recurrent policies, as our episodes are too short to compare against recurrent hierarchical policies.

SHAIL: To demonstrate the effectiveness of SHAIL, we formulate a *very simple* hierarchical policy. Our high-level controller chooses from a set of options that target a particular velocity at a particular future time, $\hat{\mathcal{A}} = \{(v, t) \mid v \in \mathcal{V},\ t \in \mathcal{T}\}$, where $\mathcal{V}$ and $\mathcal{T}$ are discrete velocity and time sets. The low-level controller for each option commands a fixed acceleration to bring the vehicle to the desired velocity at the desired time. The safety predictor returns a binary indicator for whether the option is scheduled to collide with other vehicles if they maintain their velocity. Additionally, we overwrite the largest deceleration option to always be valid, thereby rendering it a default 'safe' option `HardBrake`. Again, we use the objective from Ho and Ermon, and PPO for policy gradients. We also learn a version of SHAIL without the safety layer or early option termination for ablation (**HAIL**).

3.4.3 *Training and Metrics*

Our experiments focus on model performance in roundabouts, a customarily tricky scenario for an autonomous system to navigate. Specifically, we look at the `DR_USA_Roundabout_FT` scene, which includes five recordings consisting of over 750 ego vehicle tracks. We believe it to be the most difficult roundabout scene in the dataset. Upon collision with a controlled vehicle, the environment is reset with a new random vehicle.

We perform two experiments. In our first experiment (*in-distribution*), we train and test models in the same environment, which selects vehicles only from the first track file. The purpose of this experiment is to compare absolute potential model performance. This in-distribution testing corresponds with what was done by Kuefler, Morton, Wheeler, *et al.* [92].

In our second experiment (*out-of-distribution*), we train and validate in an environment that randomly selects vehicles from scene recordings 1–4, and we report metrics on scene 5. This out-of-distribution testing evaluates how the models perform on unseen vehicle data, though we acknowledge that we are still operating in the same driving setting. In both experiments, hyperparameters (e.g. model architecture, options sets, etc.) are optimized by choosing the ones which yield the highest success rate in the training environment. Please refer to our code for all parameters.

To avoid collisions caused by non-ego vehicles following their recorded trajectories, our test environment overrides non-ego accelerations with the IDM policy in case of an impending not-at-fault collision. For each model, we report the success rate (the rate at which an episode does not terminate in a collision), the average traveled distance (m), the root mean square error in position to the expert measured at 10 seconds into the trajectory (m), the average absolute difference between the average speed of each vehicle under expert and modeled control (m/s), and the Jensen-Shannon divergence between the distributions over all accelerations. The divergence is estimated by fitting a histogram distribution to the two sets of samples.

3.4.4 *Results*

Our simulation environment is visualized in Figure 3.2, while results from multiple runs of both in- and out-of-distribution roundabout experiments are shown in Table 3.1. From the success rate and average distance traveled metrics, we can see that the simple IDM rule-following policy, behavior cloning, and GAIL all have trouble successfully navigating through the roundabout. From the in-distribution experiment, we see that incorporating safe options, even those as simple as the ones we suggest, can yield better performance when driving in complex environments.

Table 3.1: A comparison of performance between expert, IDM, BC, GAIL, HAIL (ablation), and SHAIL policies in both in-distribution (above) and out-of-distribution (below) roundabout experiments. We report metric means and two standard deviations after training each model five times.

Model	**Success %**	**Travel Dist.**	**RMSE**$_{10s}$	$\lvert\Delta V_{\textbf{avg}}\rvert$	**Accel. JSD**
Expert	100	82.1	—	—	—
IDM	66.2	67.5	19.2	1.52	0.050
BC	45.3±3.0	49.8±1.4	22.0±1.9	1.88±0.11	0.275±0.017
GAIL	68.3±4.8	65.3±3.9	14.9±1.1	**1.08**±0.14	**0.016**±0.021
HAIL (ab)	53.0±24.1	54.5±15.3	18.3±4.8	1.87±0.86	0.333±0.008
SHAIL	**77.7**±3.1	**70.6**±2.3	**14.7**±2.1	1.27±0.23	0.312±0.012
Expert	100	81.9	—	—	—
IDM	56.3	59.9	21.0	1.45	0.061
BC	40.4±3.1	48.8±1.3	22.5±0.5	1.92±0.05	0.290±0.018
GAIL	51.6±6.8	54.5±4.7	**14.7**±1.6	1.41±0.21	**0.031**±0.018
HAIL (ab)	39.9±8.6	46.3±6.5	21.1±5.0	2.10±0.55	0.328±0.016
SHAIL	**64.4**±6.8	**61.6**±3.6	18.1±1.8	**1.37**±0.34	0.300±0.027

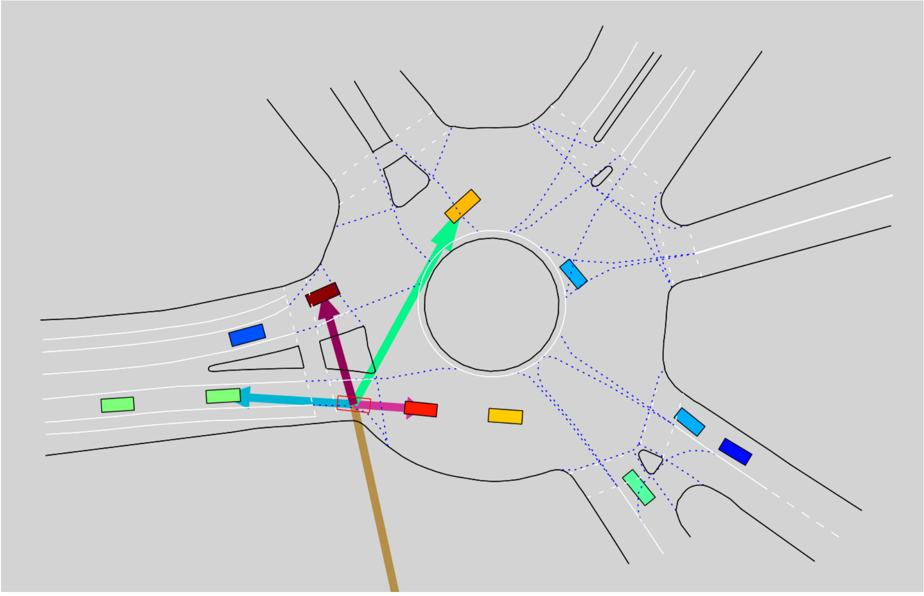


Figure 3.2: A single time-step of a policy learned by SHAIL interacting with the environment. The ego vehicle observes its own motion state and lidar-like measurements capturing relative state information about up to five surrounding vehicles.

We see improvements in the success rate and travel distance metrics. We see that metrics that judge similarity to expert position and speed perform comparably well in both GAIL and SHAIL, indicating some human-like behavior in both.

We note that the distribution over SHAIL accelerations is quite far from the expert distribution, especially when compared to GAIL. This disparity can be attributed to our overly simple option design. Our controllers attempt to target different velocities at different points by holding fixed accelerations, ultimately resulting in jerky behavior. We could bridge this gap by implementing more comfortable, human-like controller options.

In our ablation study, implementing the same options without any safety layering (HAIL) results in a severe drop in performance. Intuitively, when implementing options without safety or termination criteria, we are reducing the space of immediate actions available to the agent. Even if we could learn a good imitating controller that predicts which options might be safe from a particular state, we have no method for terminating if the options become unsafe. In our experiments, this is made even worse by our simple controllers, which stick to the action plan that was initiated during option selection and do not adjust their plan based on environment feedback.

We see similar results in our out-of-distribution experiment, noting that the performance gap between SHAIL and other models is even greater. Though none of the learning methods perform as well as they would in-distribution, this performance gap suggests that SHAIL could be a good approach for navigating new situations. We note though that our out-of-distribution experiment tested in the same roundabout setting as training, just with new vehicle data. Though we believe this to be the hardest setting, it would be interesting to train our approach over different settings and test on a fully unseen one to see how well the learned safe option selector could generalize.

3.5 *Discussion*

Previous work applying adversarial imitation learning to autonomous driving focuses on learning low-level control policies. However, since many autonomous driving systems rely on optimization-based control to provide safe low-level policies, it may be more prudent to rely on data-driven tools to learn high-level control policies. In this work, we introduced Safety-Aware Hierarchical Adversarial Imitation Learning (SHAIL), a methodology for learning high-level control policies in a simulator such that low-level expert trajectories are imitated. SHAIL incorporates an additional layer to reason over option safety and availability, allowing to guarantee that only safe and feasible actions are executed. To demonstrate our approach, we developed a simulator based on the Interaction dataset of real complex urban driving scenarios. Finally, we compared SHAIL to previous approaches to empirically demonstrate the safety improvements that it affords when navigating a roundabout.

SHAIL inherits all of the limitations from generative adversarial networks, policy optimization, and simulation-based approaches to policy learning. Additionally, SHAIL limits high-level options to a fixed set of predetermined low-level control policy instances. These options must hold semantic meaning in different initialization states for meaningful learning to occur. One limitation of our experiments is the simplicity of the low-level controllers used, which are meant only to demonstrate the potential of our proposed method. More advanced low-level controllers, safety predictors, and termination criteria can easily be substituted into our method. SHAIL can be extended to perform reward augmentation to design policies that avoid known unfavorable behavior. It can also potentially be applied across settings to learn more generalizable option-selection.

This discussion concludes our discussion of safe planning under outcome and objective uncertainty. The next chapters will shift to discussing our contributed methods for online planning while satisfying constraints under outcome and state uncertainty.

4 Continuous Constrained POMDP Planning

As overviewed in Section 2.6, Constrained Partially Observable Markov Decision Processes (CPOMDPs) provide a framework for planning safely under outcome and state uncertainty by imposing inviolable cost-value budgets. Previous work performing online planning for CPOMDPs had only been applied to discrete action and observation spaces. In this chapter, we propose algorithms for online CPOMDP planning for continuous state, action, and observation spaces by combining dual ascent with progressive widening. We empirically compare the effectiveness of our proposed algorithms on continuous CPOMDPs that model both toy and real-world safety-critical problems. Additionally, we compare against unconstrained solutions that scalarize cost constraints into rewards and highlight the myopic limitations of the default exploration scheme.

The work presented in this chapter is based on a research collaboration with Anthony Corso [149]. Our implementations are available at `https://github.com/sisl/CPOMDPExperiments`.

4.1 Introduction

Partially observable Markov decision processes (POMDPs) provide a mathematical framework for planning under uncertainty [42], [150]. An optimal policy for a POMDP maximizes the long-term expected reward that an agent accumulates while considering uncertainty from the agent's state and dynamics. Planning, however, is often multi-objective, as agents will typically trade-off between maximizing multiple rewards and minimizing multiple costs. Though this multi-objectivity can be handled explicitly [111], often times, multiple objectives are scalarized into a single reward function and penalties are captured through soft constraints. The drawback to this approach, however, is the need to define the parameters that weight the rewards and costs.

Constrained POMDPs (CPOMDPs) model penalties through hard constraints on expected cost-value, but are often harder to solve than POMDPs with scalarized reward functions. Policies for CPOMDPs with discrete state, action, and observation spaces can be generated offline through point-based value iteration [117], with locally-approximate linear programming [118], or projected gradient ascent on finite-state controllers [120]. Additional work develops an online receding horizon controller for CPOMDPs with large state spaces by combining Monte Carlo Tree Search (MCTS) [57] with dual ascent to guarantee constraint satisfaction [122]. However, this method is limited to discrete state and action spaces.

In this work, we extend MCTS with dual ascent to algorithms that leverage double progressive widening [27], [58] in order to develop online solvers for CPOMDPs with large or continuous state, action, and observation spaces. Specifically we extend three continuous POMDP solvers (POMCP-DPW, POMCPOW and PFT-DPW) [58] to create the constrained versions (CPOMCP-DPW, CPOMCPOW, and CPFT-DPW). In our experiments, we a) demonstrate our three solvers on toy and real-world constrained problems, b) compare against an unconstrained solver using reward scalarization, and c) demonstrate shortcomings of our solvers that are associated with pessimistic backpropagation.

4.2 Background

POMDPs

Recall from Section 2.3 that a POMDP augments an MDP with observations $o \in \mathcal{O}$ and an observation model Z that maps a state transition to a distribution over emitted observations. An agent policy maps an initial state distribution and a history of actions and observations to an instantaneous action. An optimal policy acts to maximize expected discounted reward [42], [150].

Offline POMDP planning algorithms [43], [45] yield compact policies that act from any history but are typically limited to relatively small state, action, and observation spaces. Online algorithms yield good actions from immediate histories during execution [151]. Silver and Veness [57] apply Monte-Carlo Tree Search (MCTS) over histories to plan online in POMDPs with large state spaces (POMCP). Progressive widening is a technique for slowly expanding the number of children in a search tree when the multiplicity of possible child nodes is high or infinite (i.e. in continuous spaces) [27], [152]. Sunberg and Kochenderfer [58] apply double progressive widening (DPW) to extend POMCP planning to large and continuous action and observation spaces. Additional work considers methods for selecting new actions when progressively widening using a space-filling metric [59] or Expected Improvement exploration [64]. Wu, Yang, Zhang, *et al.* [153] improve performance by merging similar observation branches.

Constrained planning

Constrained POMDPs augment the POMDP tuple with a cost function $\mathcal{C}$ that maps each state transition to a vector of instantaneous costs, and a cost budget vector $\hat{\mathbf{c}}$ that the expected discounted cost returns must satisfy. An optimal CPOMDP policy π maximizes expected discounted reward subject to hard cost constraints. This objective can be expanded from Equations (2.22) and (2.23) as:

$$\max_{\pi} V_R^{\pi}(b_0) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(b_t, a_t) \mid b_0\right] \tag{4.1}$$

$$\text{s.t. } V_{C_k}^{\pi}(b_0) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t C_k(b_t, a_t) \mid b_0\right] \leq \hat{c}_k \;\forall\, k, \tag{4.2}$$

where b_0 is the initial state distribution, and belief-based reward and cost functions return the expected reward and costs from transitions from states in those beliefs.

Altman [116] overviews fully-observable Constrained Markov Decision Processes (CMDPs), while Piunovskiy and Mao [154] solve them offline using dynamic programming on a state space augmented with a constraint admissibility heuristic. Early offline CPOMDP solvers use an α-vector formulation for value and perform cost and reward backups [114] or augment the state space with cost-to-go and perform point-based value iteration [117]. CALP [118] iterates solving a linear program with a fixed set of reachable beliefs to perform locally-approximate value iteration and expanding the belief set to represent a good policy. Walraven and Spaan [119] improve upon this by leveraging column generation. Wray and Czuprynski [120] represent a CPOMDP policy with a finite state controller learned offline through projected gradient ascent.

To generate good actions online, Undurti and How [121] perform lookahead search up to a fixed depth while using a conservative constraint-minimizing policy learned offline to estimate the cost at leaf nodes and prune unsafe branches. More recently, CC-POMCP performs CPOMDP planning by combining POMCP with dual ascent [122]. Besides tracking cost values, CC-POMCP maintains estimates for Lagrange multipliers that are used to guide search. Between search queries, CC-POMCP updates Lagrange multipliers using constraint violations. CC-POMCP outperforms CALP in large state spaces.

4.3 Approach

Recall from Section 2.6 that the Lagrangian of the constrained POMDP planning problem can be formulated as

$$\max_{\pi} \min_{\boldsymbol{\lambda} \geq 0} [V_R^{\pi}(b_0) - \boldsymbol{\lambda}^{\top}(V_{\mathbf{C}}^{\pi}(b_0) - \hat{\mathbf{c}})]. \tag{4.3}$$

POMCP [122] optimizes this objective directly by interleaving optimization for π using POMCP and optimization of $\boldsymbol{\lambda}$ using dual ascent. The planning procedure for CC-POMCP is summarized in the first procedure in Listing 1, with lines 6 and 7 depicting the dual ascent phase with an update schedule α_i. During policy

```
1:  procedure Plan($b$)
2:      $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}_0$
3:      for $i \in 1 : n$
4:          $s \leftarrow$ sample from $b$
5:          Simulate($s, b, d_{\max}$)
6:          $a \sim$ GreedyPolicy($b, 0, 0$)
7:          $\boldsymbol{\lambda} \leftarrow [\boldsymbol{\lambda} + \alpha_i(\mathbf{Q}_C(ba) - \hat{\mathbf{c}})]^+$
8:      return GreedyPolicy($b, 0, \nu$)
9:  procedure GreedyPolicy($h, \kappa, \nu$)
10:     $Q_\lambda(ba) := Q(ba) - \boldsymbol{\lambda}^\top \mathbf{Q}_C(ba) + \kappa\sqrt{\frac{\log N(h)}{N(ha)}}$
11:     return StochasticPolicy($Q_\lambda(ba), \nu$)
12: procedure ActionProgWiden($h$)
13:     if $|C(h)| \leq k_a N(h)^{\alpha_a}$
14:         $a \leftarrow$ NextAction($h$)
15:         $C(h) \leftarrow C(h) \cup \{a\}$
16:     $\pi \leftarrow$ GreedyPolicy($h, c, \nu$)
17:     return sample from $\pi$
```

Listing 4.1: Common procedures for online continuous CPOMDP planning with dual ascent.

optimization, actions are always chosen with respect to the value of the Lagrangian with exploration parameter κ (lines 10 and 11). The `StochasticPolicy` procedure (not shown) builds a stochastic policy of actions that are ν-close to the maximal Lagrangian action-value estimate.

One significant shortcoming of using POMCP for policy optimization is that it only admits discrete action and observation spaces. In this work, we address this shortcoming using progressive widening, in which a new child node is only added to parent node p when the number of children nodes $|C(p)| \leq kN(p)^\alpha$, where $N(p)$ is the number of total visits to p, and k and α are tree-shaping hyperparameters that can differ for action and observation branching.

Progressively widening can be straightforwardly applied to CC-POMCP. The extended version of this chapter[1] includes an outline of the resulting algorithm,

[1] Available at `arxiv.org/abs/2212.12154`

CPOMCP-DPW. However, as noted by Sunberg and Kochenderfer [58], progressively widening in large observation spaces leads to particle collapse as each observation node after the first step only holds a single state. To alleviate this, they propose POMCPOW to iteratively build up particle beliefs at each observation node and PFT-DPW to perform belief tree search using particle filter beliefs. They discuss when these algorithms may outperform vanilla progressive widening and note the absence of optimality guarantees.

In Algorithms 4.2 and 4.3, these approaches for POMDP planning in continuous spaces are combined with dual ascent in order to perform CPOMDP planning in continuous spaces. We note that these algorithms are amenable to methods that combine similar observation nodes [153] or implement better choices for actions [64] to provide optimality guarantees [59].

4.4 *Experiments*

We consider three constrained variants of continuous POMDP planning problems in order to empirically show the efficacy of our methods. We demonstrate the different continuous CPOMDP solvers, compare the use of CPOMCPOW against POMCPOW with scalarized costs, and investigate cost backpropagation to highlight limitations. We use Julia 1.6 and the POMDPs.jl framework in our experiments [155]. Our full experimentation details, including hyperparameter choices, are available at `github.com/sisl/CPOMDPExperiments`.

CPOMDP Problems

We enumerate the CPOMDP problems below, along with whether their state, action, and observation spaces are (D)iscrete or (C)ontinuous.

1. **Constrained LightDark** (C, D, C): In this adaptation of LightDark [58], the agent can choose from moving in discrete steps of $\mathcal{A} = \{0, \pm 1, \pm 5, \pm 10\}$ in order to navigate to $s \in [-1, 1]$, take action 0, and receive $+100$ reward. An action of 0 elsewhere accrues a -100 reward and the agent yields a per-step

```
**procedure** SIMULATE($s, h, d$)
    **if** $d = 0$
        **return** $0, \mathbf{0}$
    $a \leftarrow$ ACTIONPROGWIDEN($h$)
    $s', o, r, \mathbf{c} \leftarrow G(s, a)$
    **if** $|C(ha)| \leq k_o N(ha)^{\alpha_o}$
        $M(hao) \leftarrow M(hao) + 1$
    **else**
        $o \leftarrow$ select $o \in C(ha)$ w.p. $\frac{M(hao)}{\sum_o M(hao)}$
    append $s'$ to $B(hao)$
    append $\mathcal{Z}(o \mid s, a, s')$ to $W(hao)$
    **if** $o \notin C(ha)$
        $C(ha) \leftarrow C(ha) \cup \{o\}$
        $V', \mathbf{C}' \leftarrow$ ROLLOUT($s', hao, d - 1$)
    **else**
        $s' \leftarrow$ select $B(hao)[i]$ w.p. $\frac{W(hao)[i]}{\sum_{j=1}^{m} W(hao)[j]}$
        $r \leftarrow R(s, a, s')$
        $\mathbf{c} \leftarrow C(s, a, s')$
        $V', \mathbf{C}' \leftarrow$ SIMULATE($s', hao, d - 1$)
    $V \leftarrow r + \gamma V'$
    $\mathbf{C} \leftarrow \mathbf{c} + \gamma \mathbf{C}'$
    $N(h) \leftarrow N(h) + 1$
    $N(ha) \leftarrow N(ha) + 1$
    $Q(ha) \leftarrow Q(ha) + \frac{V - Q(ha)}{N(ha)}$
    $\mathbf{Q}_C(ha) \leftarrow \mathbf{Q}_C(ha) + \frac{\mathbf{C} - \mathbf{Q}_C(ha)}{N(ha)}$
    $\bar{\mathbf{c}}(ha) \leftarrow \bar{\mathbf{c}}(ha) + \frac{\mathbf{c} - \bar{\mathbf{c}}(ha)}{N(ha)}$
    **return** $V, \mathbf{C}$
```

Algorithm 4.2: CPOMCPOW simulate procedure

```
**procedure** SIMULATE$(\cdot, b, d)$
    **if** $d = 0$
        **return** $0$
    $a \leftarrow$ ACTIONPROGWIDEN$(b)$
    **if** $|C(ba)| \leq k_o N(ba)^{\alpha_o}$
        $b', r, \mathbf{c} \leftarrow G_{\text{PF}(m)}(b, a)$
        $C(ba) \leftarrow C(ba) \cup \{(b', r, \mathbf{c})\}$
        $V', \mathbf{C}' \leftarrow$ ROLLOUT$(b', d-1)$
    **else**
        $b', r, \mathbf{c} \leftarrow$ sample uniformly from $C(ba)$
        $V', \mathbf{C}' \leftarrow$ SIMULATE$(\cdot, b', d-1)$
    $V \leftarrow r + \gamma V'$
    $\mathbf{C} \leftarrow \mathbf{c} + \gamma \mathbf{C}'$
    $N(b) \leftarrow N(b) + 1$
    $N(ba) \leftarrow N(ba) + 1$
    $Q(ba) \leftarrow Q(ba) + \frac{V - Q(ba)}{N(ba)}$
    $\mathbf{Q}_C(ba) \leftarrow \mathbf{Q}_C(ba) + \frac{\mathbf{C} - \mathbf{Q}_C(ba)}{N(ba)}$
    $\bar{\mathbf{c}}(ba) \leftarrow \bar{\mathbf{c}}(ba) + \frac{\mathbf{c} - \bar{\mathbf{c}}(ba)}{N(ba)}$
    **return** $V, \mathbf{C}$
```

Algorithm 4.3: CPFT-DPW simulate procedure

reward of -1. The agent starts in the dark region, $b_0 = \mathcal{N}(2, 2)$, and can navigate towards the light region at $s = 10$ to help localize itself. However, there is a cliff at $s = 12$, above which the agent will receive a per-step cost of 1. The agent must maintain a cost budget of $\hat{c} = 0.1$, and so taking the $+10$ action immediately would violate the constraint.

2. **Constrained Van Der Pol Tag** (C, C, C): In this problem, a constant velocity agent must choose its orientation in order to intercept a partially observable target whose dynamics follow the Van der Pol oscillator [58]. In our adaptation, rather than penalizing taking good observations in the reward function, we formulate a cost constraint that dictates that the discounted number of good observations taken must be less than 2.5.

3. **Constrained Spillpoint** (C, C, C) This CPOMDP models safe geological carbon capture and sequestration around uncertain subsurface geometries and properties [156]. In the original POMDP, instances of CO_2 leaking through faults in the geometry are heavily penalized, both for the presence of a leak and for the total amount leaked. In our adaptation, we instead impose a hard constraint of no leaking.

	LightDark		**Van Der Pol Tag**		**Spillpoint**	
Model	V_R	V_C [≤ 0.1]	V_R	V_C [≤ 2.5]	V_R	V_C [≤ 0]
`CPOMCPOW`	17.1±0.7	0.090±0.002	**24.5±0.4**	1.57±0.001	3.93±0.17	0.001±0.000
`CPFT-DPW`	**51.9±0.4**	0.044±0.002	−0.6±0.3	1.05±0.01	**4.19±0.15**	0.001±0.000
`CPOMCP-DPW`	−6.5±0.4	0.000±0.000	12.5±0.5	1.71±0.01	3.17±0.18	0.001±0.000

Table 4.1: Continuous CPOMDP online algorithm demonstrations comparing mean discounted cumulative rewards and costs across 100 LightDark simulations, 50 Van Der Pol Tag simulations, and 10 Spillpoint simulations.

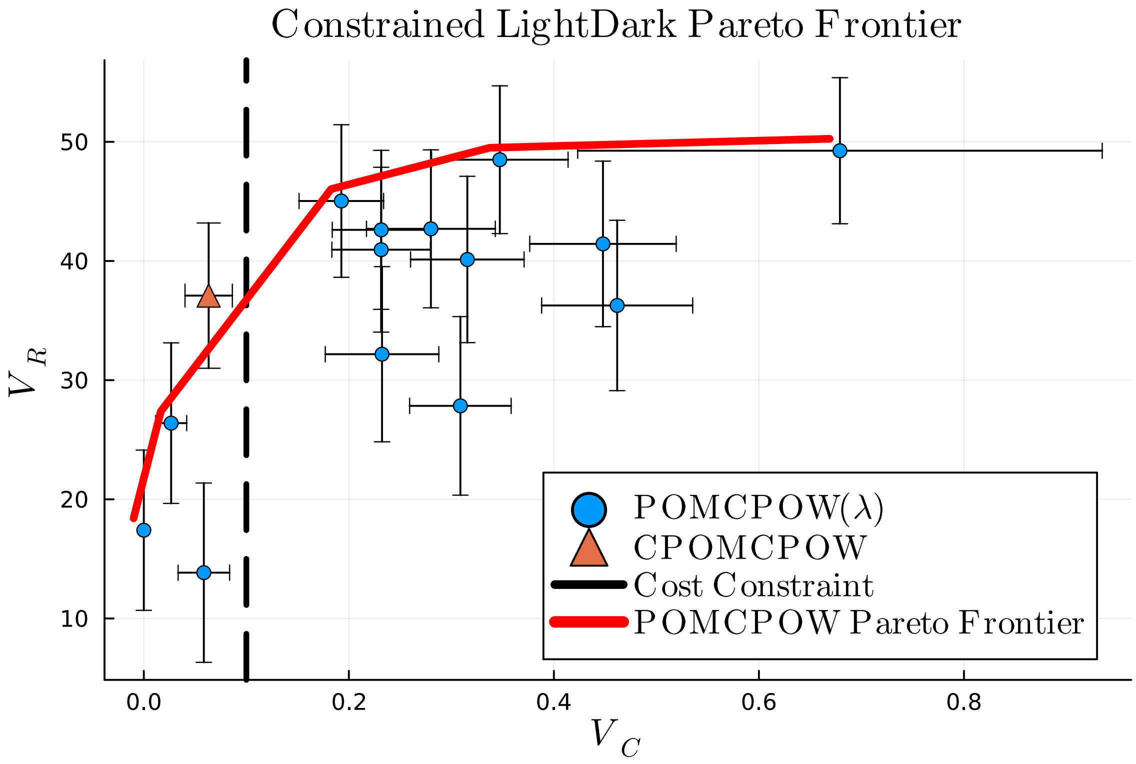


Figure 4.1: The V_R vs. V_C Pareto frontier of solutions to the scalarized LightDark POMDP solved with POMCPOW and the Constrained LightDark CPOMCPOW solution. Error bars depict standard error after 100 simulations.

CPOMDP algorithm comparison

Table 4.1 demonstrates the use of the different algorithms, comparing mean rewards and costs on the three target problems. We note that performance is highly dependent on the rollout policies, which are different for each solver and problem. Crucially, all of our methods can generate desirable behavior while satisfying hard constraints without scalarization. *This is especially evident in the Spillpoint problem, where setting hard constraints minimizes CO_2 leakage while improving the reward generated by unconstrained POMCPOW reported by Corso, Wang, Zechner,* et al. [156].

Hard constraints vs. reward scalarization

Next, we demonstrate the benefit of imposing hard constraints. To do so, we create an unconstrained LightDark problem by scalarizing the reward function linearly and having an unconstrained solver optimize $\bar{R}(s,a) = R(s,a) - \lambda C(s,a)$. We then vary choices of λ and compare the reward and cost outcomes when using POMCPOW to solve the scalarized problem against using CPOMCPOW with the underlying CPOMDP.

In Figure 4.1, we depict the Pareto frontier when simulating 100 episodes at each design point. We see that the solution to the constrained problem using CPOMCPOW lies on the approximate frontier while consistently satisfying the cost constraint. We can therefore see that constrained solvers can yield high reward values at a fixed cost while eliminating the need to search over scalarization parameters.

Cost backpropagation

Model	$N(b_0 a)/N(b_0)$	$Q_c(b_0 a)$	$\Delta Q_\lambda(b_0 a)$
Default	$[0.10, 0.19, 0.08]$	$[0.18, 0.32, 1.15]$	$[5.3, 5.5, 18.9]$
Min	$[0.14, 0.33, 0.18]$	$[0.00, 0.00, 0.29]$	$[4.9, \mathbf{2.1}, 4.3]$
Uncstr.	$[0.09, 0.30, 0.35]$	—	$[8.9, 6.6, \mathbf{4.5}]$

Table 4.2: Statistics corresponding with actions 1, 5, and 10 when running CPOMCPOW with the default cost propagation, minimal cost propagation, and on the unconstrained problem. ΔQ_λ denotes the gap to the best Lagrangian action-value, and the action taken most often is bolded.

Finally, we notice that using a single dual parameter to guide the search globally can result in overly conservative policies, as a globally constraining dual parameter would still guide the search in safe subtrees. To examine this, we simulate CPOMCPOW searches from the initial LightDark belief and compare backpropagating costs normally against backpropagating the minimal cost-value across sibling branches, i.e. the best-case cost assuming a closed loop. In Table 4.2, we compare statistics averaged across 50 searches for taking actions 1, 5, and 10. While in the unconstrained problem, the agent chooses the 10 action to localize itself quickly, we note that the constrained agent should choose the 5 action to carefully move towards the light region without overshooting and violating the cost constraint. We see that with the default search mechanism, the costs at the top level of the search tree are overly pessimistic, noting that actions of 1 or 5 should have zero cost-value as they are recoverable. For this search, propagating minimal costs achieves the desired result.

4.5 Discussion

Planning under uncertainty is often multi-objective. Though multiple objectives can be scalarized into a single reward function with soft constraints, CPOMDPs provide a mathematical framework for POMDP planning with hard constraints. Previous work performs online CPOMDP planning for large state spaces, but small, discrete action and observation spaces by combining MCTS with dual ascent [122]. We proposed algorithms that extend this to large or continuous action and observation spaces using progressive widening, demonstrating our solvers empirically on toy and real CPOMDP problems.

Limitations: A significant drawback of CC-POMCP is that constraint violations are only satisfied in the limit, limiting its ability to be used as an anytime planner. This is worsened when actions and observations are continuous, as the progressive widening can miss subtrees of high cost. Finally, we note the limitation of using a global dual parameter λ to guide the whole search as different belief nodes may necessitate different safety considerations.

In this chapter, though we demonstrated online planning in continuous CPOMDPs, the problems we investigated were relatively limited in their scope. In the next chapter, we will introduce a method that can scale CPFT-DPW to much larger problems by leveraging hierarchical decompositions. This method will have the added benefit that under the right assumptions, it can guarantee constraint satisfaction anytime.

5 Hierarchical Constrained POMDP Planning

As noted in Chapter 4, though Constrained Partially Observable Markov Decision Processes (CPOMDPs) can generalize safe planning under state and transition uncertainty, online CPOMDP planning is extremely difficult in large or continuous problem domains. Fortunately, in many large domains, hierarchical decomposition (Section 2.5) can simplify planning by using tools for low-level control given high-level action primitives (options).

In this chapter, we introduce Constrained Options Belief Tree Search (COBeTS), which combines ideas from Chapters 3 and 4, leveraging hierarchical decomposition to scale online closed-loop CPOMDP planning to larger problems. We show that if primitive option controllers are defined to satisfy assigned constraint budgets, then COBeTS will satisfy constraints anytime. Otherwise, COBeTS will guide the search towards a safe sequence of option primitives, and hierarchical monitoring can be used to achieve runtime safety. We demonstrate COBeTS in several safety-critical, constrained partially observable robotic domains, showing that it can plan successfully in continuous CPOMDPs while non-hierarchical baselines cannot.

The work presented in this chapter is based on a research collaboration with Hugo Buurmeijer, Kyle Wray, and Anthony Corso [157]. Our implementations are available at `https://github.com/sisl/COBTSExperiments`.

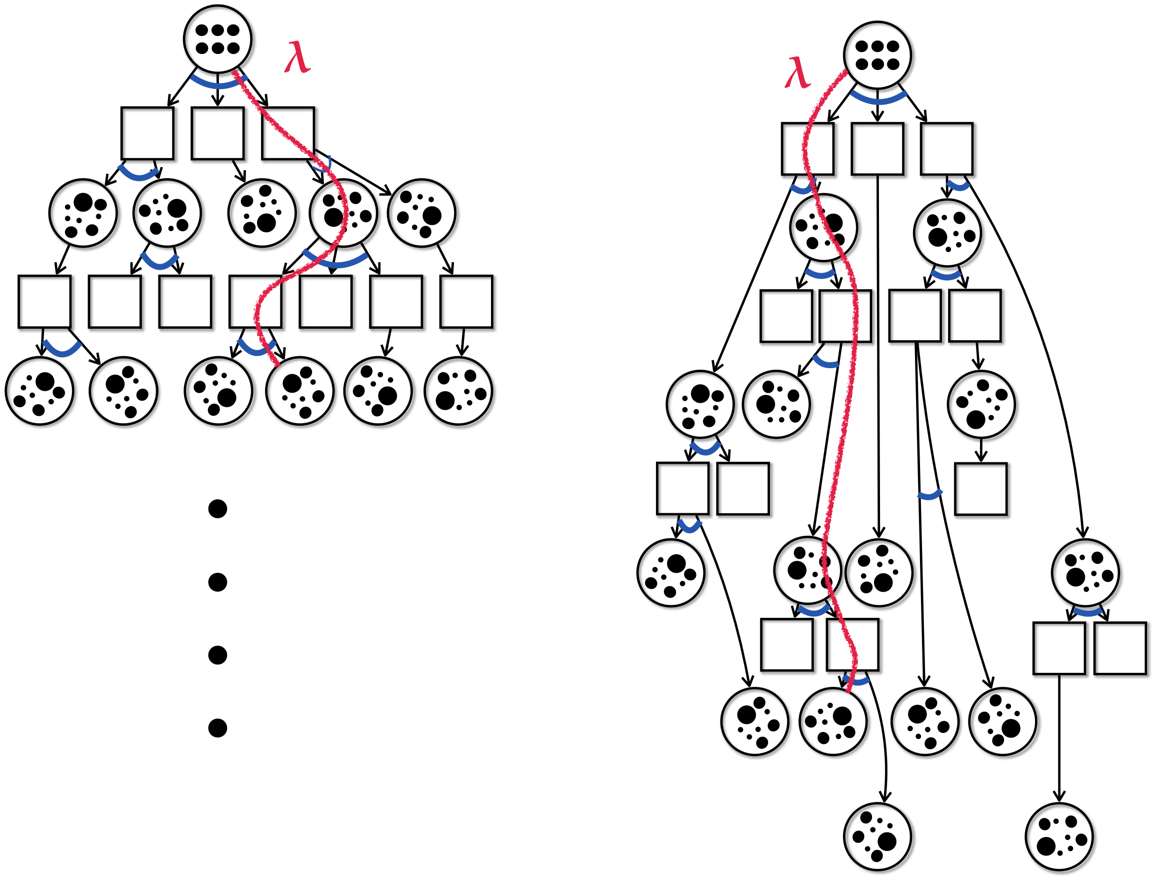
5.1 Introduction

Planning in robotics requires robust regard for safety, which often necessitates careful consideration of uncertainty. Two factors contributing to uncertainty include a) the true state of the robot and surrounding environment (*state* uncertainty), and b) how that state will evolve given robot actuation (*transition* uncertainty). Constrained partially observable Markov decision processes (CPOMDPs) provide a general mathematical framework for safe planning under state and transition uncertainty by imposing constraints [118].

While offline CPOMDP planning algorithms are able to build policies for discrete environments with thousands of possible states [118], building policies in many robotic domains that are typically large or continuous necessitates online planning. Online CPOMDP planning has been scaled to large or continuous state spaces by combining Monte Carlo tree search with Lagrangian exploration and dual ascent [122] and has recently been extended to domains with continuous action and observation spaces [149]. However, search-based planning with large or continuous action and observation spaces is still extremely difficult. Common techniques try to artificially limit the width of the search tree by restricting the sets of successor nodes [27], [58]. Unfortunately, it can often be difficult to guarantee the inclusion of promising actions in the reduced search set. This problem accelerates as searches deepen, which can be especially problematic when problems require deep searches to find promising action sequences. Models can be learned for biasing action selection [62]–[64], however, this requires generating data from past experience.

In many robotic planning applications, low-level controllers can be easily crafted for different high-level action primitives using domain expertise or commonly available tools (e.g. trajectory optimization). Decomposing search hierarchically over these action primitives (macro-actions, options) can *significantly* reduce the size of the search tree. Decomposition can reduce the space of actions to search over, but more importantly, reduces the search depth required to plan to the same horizon.

In this paper, we introduce Constrained Options Belief Tree Search (COBeTS),



(a) CPFT-DPW Tree [149]

(b) COBeTS Tree

Figure 5.1: In CPFT-DPW [149] (left), progressive widening (blue) is used to limit the branching factor of the Monte Carlo belief-state search tree, while dual parameters (red) are optimized to guide the search towards constraint satisfaction. COBeTS (right), leverages a hierarchy to decompose the partially observable planning problem, resulting in a search tree over options and semi-Markov belief transitions, with potentially far fewer nodes.

a Monte Carlo tree search algorithm that leverages hierarchical decomposition to scale online CPOMDP planning. COBeTS, depicted in Figure 5.1, combines the options framework to handle hierarchies [136], particle filter tree search to search over beliefs [58], progressive widening to limit the number of observation *sequences* emitted from each option node, and Lagrangian exploration with dual ascent to guide the search towards safe primitives [122]. We show that if options can satisfy assigned constraint budgets, COBeTS satisfies constraints anytime. If not, dual ascent will guide the search toward safety, satisfying constraints in the limit. In our experiments, we demonstrate COBeTS on a toy domain, a carbon sequestration problem, and two robot localization problems. In each of these problems, COBeTS significantly outperforms state-of-the-art baselines that plan without hierarchical decomposition. Additionally, we demonstrate that COBeTS can satisfy constraints anytime with feasible options, and COBeTS with many options can outperform baselines because of the overall reduction in tree size induced by a hierarchy. To our knowledge, this is the first work explicitly formulating hierarchical CPOMDP planning.

In summary, our contributions are to:

- introduce COBeTS to perform online CPOMDP planning in large or continuous domains by using hierarchical decomposition,
- examine its anytime safety properties and tree complexity reduction, and
- demonstrate COBeTS on four constrained partially observable problems, including two robotic problems where non-hierarchical baselines fail to plan successfully.

5.2 Background

5.2.1 Hierarchical Planning in CPOMDPs

As noted in previous sections, CPOMDP planning with large problem spaces is difficult. Hierarchical planning simplifies difficult planning problems by favorably decomposing them into more tractable subproblems [104], [106]. Recall the options framework discussed in Section 2.5. A partially observable options model augments an underlying POMDP problem with the set $\{\mathcal{I}_{\hat{a}}, \pi^L_{\hat{a}}, \beta_{\hat{a}}\}_{\hat{a}\in\hat{\mathcal{A}}}$, that for every option, defines a set of belief-states $\mathcal{I}$ from which it can be initialized, a low-level control policy $\pi^L(a \mid b)$ returning actions from the underlying action space $\mathcal{A}$, and a function β that returns the probability that an option will terminate in a given belief. An

```
1: procedure Execute($b_0, \hat{\mathbf{c}}$)
2:     $\hat{a} \leftarrow \emptyset, b \leftarrow b_0$
3:     while $\neg$Terminal($b$)
4:         if $\hat{a} = \emptyset \lor$ Terminate($\hat{a}, b$)
5:             $\hat{a} \leftarrow$ SelectOption($b, \hat{\mathbf{c}}$)
6:         $a \leftarrow$ Action($\hat{a}, b$)
7:         $o \leftarrow$ Step($b, a$)
8:         $\hat{\mathbf{c}} \leftarrow \left[\frac{\hat{\mathbf{c}} - \mathbf{C}(b,a)}{\gamma}\right]^{+}$
9:         $b \leftarrow$ UpdateBelief($b, a, o$)
```

Algorithm 5.1: Hierarchical execution in an options CPOMDP

implementation of hierarchical policy execution in CPOMDPs using the options framework is depicted in Algorithm 5.1. During execution, the low-level policy acts in the environment (lines 6–7), updates the cost budget with the expected instantaneous cost (line 8), and updates the state belief using a new observation (line 9). A high-level `SelectOption` policy chooses a new option whenever an executing option terminates (lines 4–5).

Related previous work performed online hierarchical planning for *unconstrained* POMDPs [158] by combining partially observable MCTS with MaxQ [105], an alternative framework for hierarchical planning. Though not explicitly using hard constraints, additional recent work has focused on safe planning in large partially observable robotic domains by using a hierarchical information roadmap to manage local risks safely on large exploration missions [159]. For problems with favorable state partitions (e.g. path planning on a grid of neighboring states), work has also been done to solve large, *fully-observable*, constrained MDPs hierarchically by combining global CMDP solutions over coarse partitions with local solutions over underlying states [160].

5.3 Methodology

5.3.1 Preliminaries

We formulate hierarchical planning in a CPOMDP using the options framework. Options induce two processes: a low-level Markov process over the underlying state, action, and observation space, and a high-level semi-Markov process between successive option calls. With option policies defined a priori, the underlying model is a constrained partially observable semi-Markov decision process (CPOSMDP). Consequently, we now briefly cover the CSMDP, its generalization as CPOSMDP, and the CPOSMDP's equivalent belief-state CSMDP. More details can be found in related work such as by Vien and Toussaint [158].

A CSMDP is defined in a way similar to a CMDP, with the inclusion of a now semi-Markov transition function T that defines the joint probability of the successor state alongside the number of steps required to transition given a state and action, $p(s', \tau \mid s, a)$. Decisions are made at successive decision epochs e, each indexing a time step t_e when an action a_e begins executing and its duration τ_e where $t_{e+1} = t_e + \tau_e$.

Similarly, a CPOSMDP is defined with the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P, R, \mathbf{C}, \hat{\mathbf{c}}, \gamma)$, where P models the joint semi-Markov transition and observation functions $p(s', \mathbf{o}, \tau \mid s, a)$, where $\mathbf{o} \in \mathcal{O}^\tau$ is the sequence of emitted observations in a τ-step semi-Markov transition. As with POMDPs and their equivalent belief-state MDP representations, we can define a CPOSMDP equivalently as a belief-state CSMDP $(\tilde{\mathcal{S}}, \mathcal{A}, \tilde{T}, \tilde{R}, \tilde{\mathbf{C}}, \hat{\mathbf{c}}, \gamma)$. The belief states are $b \in \tilde{\mathcal{S}} = \triangle(\mathcal{S})$, with transitions $\tilde{T}(b, a, b' = ba\mathbf{o}, \tau) = \sum_{s,s' \in \mathcal{S}} b(s) b'(s') P(s, a, s', \mathbf{o}, \tau)$, rewards $\tilde{R}(b, a) = \sum_{s \in \mathcal{S}} b(s) R(s, a)$, and costs $\tilde{\mathbf{C}}(b, a) = \sum_{s \in \mathcal{S}} b(s) \mathbf{C}(s, a)$.

Proposition 1. *For all policies π, the reward value functions and cost value functions of the belief-state CSMDP are equal to those of the CPOSMDP, that is, for all $b \in \tilde{\mathcal{S}}$, $\tilde{V}_R^\pi(b) = V_R^\pi(b)$ and $\tilde{\mathbf{V}}_C^\pi(b) = \mathbf{V}_C^\pi(b)$.*

The proof follows directly from Theorem 2 of Vien and Toussaint [158], with vector costs and cost-values treated analogously as scalar rewards and reward-values in the original proof. This theoretical result lays the groundwork for COBeTS as it allows us to solve CPOSMDPs by solving their equivalent belief-state CSMDPs.

5.3.2 *Constrained Options Belief-Tree Search* (*COBeTS*)

The idea behind COBeTS is to select new options in a large CPOSMDP by planning over the equivalent belief-state CSMDP. COBeTS augments CPFT-DPW [149], a recent algorithm for online belief-state CMDP planning, with careful consideration for the options framework. That is, rather than search over actions a, COBeTS searches over options $\hat{a}$ and samples their induced semi-Markov belief-state transitions.

```
1:  procedure SelectOption($b, \hat{\mathbf{c}}$)
2:      $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}_0$
3:      for $i \in 1 : n$
4:          $Q_\lambda(b\hat{a}) := Q(b\hat{a}) - \boldsymbol{\lambda}^\top \mathbf{Q}_C(b\hat{a})$
5:          Simulate($b, \hat{\mathbf{c}}, d_{\max}$)
6:          $\hat{a} \leftarrow \arg\max_{\hat{a}} Q_\lambda(b\hat{a})$
7:          $\boldsymbol{\lambda} \leftarrow [\boldsymbol{\lambda} + \alpha_i(\mathbf{Q}_C(b\hat{a}) - \hat{\mathbf{c}})]^+$
8:      return $\arg\max_{\hat{a}} Q(b\hat{a})$ s.t. $\mathbf{Q}_C(b\hat{a}) \leq \hat{\mathbf{c}}$
9:  procedure OptionProgWiden($b, \hat{\mathbf{c}}$)
10:     if $|C(b)| \leq k_a N(b)^{\alpha_a}$
11:         $\hat{a} \leftarrow$ SampleNextOption($b, [\hat{\mathbf{c}}]^+$)
12:         $C(b) \leftarrow C(b) \cup \{\hat{a}\}$
13:     $Q_{\lambda UCB}(b\hat{a}) := Q_\lambda(b\hat{a}) + \kappa\sqrt{\frac{\log N(b)}{N(b\hat{a})}}$
14:     return $\arg\max_{\hat{a}} Q_{\lambda UCB}(b\hat{a})$
15: procedure Simulate($b, \hat{\mathbf{c}}, d$)
16:     if $d \leq 0$
17:         return $0, \mathbf{0}$
18:     $\hat{a} \leftarrow$ OptionProgWiden($b, \hat{\mathbf{c}}$)
19:     if $|C(b\hat{a})| \leq k_o N(b\hat{a})^{\alpha_o}$
20:         $b', \tilde{r}, \tilde{\mathbf{c}} \leftarrow G_{\mathrm{PF}(m)}(b,$ Action$(\hat{a}, b))$
21:         $\tau \leftarrow 1$
22:         while $\neg$Terminate$(\hat{a}, b') \wedge (d - \tau > 0)$
23:             $b', r, \mathbf{c} \leftarrow G_{\mathrm{PF}(m)}(b',$ Action$(\hat{a}, b'))$
24:             $\tilde{r} \leftarrow \tilde{r} + \gamma^\tau r$
25:             $\tilde{\mathbf{c}} \leftarrow \tilde{\mathbf{c}} + \gamma^\tau \mathbf{c}$
26:             $\tau \leftarrow \tau + 1$
27:         $C(b\hat{a}) \leftarrow C(b\hat{a}) \cup \{(b', \tilde{r}, \tilde{\mathbf{c}}, \tau)\}$
28:         $V', \mathbf{C}' \leftarrow$ Rollout$(b', \frac{\hat{\mathbf{c}} - \tilde{\mathbf{c}}}{\gamma^\tau}, d - \tau)$
29:     else
30:         $b', \tilde{r}, \tilde{\mathbf{c}}, \tau \leftarrow$ sample uniformly from $C(b\hat{a})$
31:         $V', \mathbf{C}' \leftarrow$ Simulate$(b', \frac{\hat{\mathbf{c}} - \tilde{\mathbf{c}}}{\gamma^\tau}, d - \tau)$
32:     $V \leftarrow \tilde{r} + \gamma^\tau V'$
33:     $\mathbf{C} \leftarrow \tilde{\mathbf{c}} + \gamma^\tau \mathbf{C}'$
34:     $N(b) \leftarrow N(b) + 1$
35:     $N(b\hat{a}) \leftarrow N(b\hat{a}) + 1$
36:     $Q(b\hat{a}) \leftarrow Q(b\hat{a}) + \frac{V - Q(b\hat{a})}{N(b\hat{a})}$
37:     $\mathbf{Q}_C(b\hat{a}) \leftarrow \mathbf{Q}_C(b\hat{a}) + \frac{\mathbf{C} - \mathbf{Q}_C(b\hat{a})}{N(b\hat{a})}$
38:     return $V, \mathbf{C}$
```

Algorithm 5.2: Option selection using Constrained Options Belief Tree Search (COBeTS)

The COBeTS option-selection procedure is outlined in Algorithm 5.2 with changes from CPFT-DPW highlighted in blue. The procedure is a recursive Monte Carlo tree search on a particle filter belief-state b. In `OptionProgWiden`, option selection (lines

13–14) is guided by a Lagrangian upper confidence bound heuristic that uses the current estimate of the dual parameters to trade off between reward and constraint objectives (line 4) and an exploration bonus based on visit counts. Dual parameters are updated between searches through dual ascent (line 7), in which constraint violations during search induce strengthening of the dual parameters and vice versa.

After selecting a search option (line 18), COBeTS imagines executing that option until its termination, resulting in a semi-Markov belief transition (lines 20–27). Option execution uses sampled low-level actions and observations to update the m-state particle filter belief at every step (line 23) while tracking the discounted rewards and costs accumulated along the option trajectory. New leaf nodes are initialized with value estimates that can be generated from default policy rollouts or heuristics (`Rollout` in line 28). The simulated reward and cost values are used to make temporal difference updates and are then backpropagated (lines 32–38).

Searching across a large set of options or resulting transitions necessitates techniques to limit the size of the tree. Progressive widening artificially limits the branching factor of a node as a function of its visit count $N(b)$, limiting the number of children to $|Ch(b)| \approx kN(b)^{\alpha}$, where $k > 0$ and $\alpha \in (0, 1)$ are hyperparameters that control the shape of the widening [27], [58]. COBeTS implements progressive widening on the option space (lines 9–14) and on the semi-Markov belief-state transition space (line 19). Different option sampling strategies in `SampleNextOption` can ensure coverage of the option space [59]. Since the transition distribution $T = p(b', \tau \mid b, \hat{a})$ is often continuous and uncountable, COBeTS benefits greatly from progressive widening on its belief transitions for the same reasons as large continuous POMDPs [58].

Hierarchical decomposition provides computational advantages that can be estimated through the ratio in sizes between analogous CMDP and CSMDP search trees. Consider searching across a belief-state CMDP with average action branching of cardinality A and state transition branching with average cardinality O alongside a belief-state CSMDP with action branching of average cardinality $c_1 A$, and state transition branching with average cardinality $c_2 O$ after an average of τ steps.

With a fixed time horizon T, searching over the CSMDP instead of the CMDP improves computational complexity by a factor of $\mathcal{O}((c_1c_2)^{-T/\tau}(AO)^{T(\tau-1)/\tau})$, as the ratio in the computational complexity can be expressed as the ratio of the tree sizes:

$$\frac{\text{CSMDP size}}{\text{CMDP size}} = \mathcal{O}\left(\frac{(c_1c_2AO)^{\frac{T}{\tau}}}{(AO)^T}\right) = \mathcal{O}\left(\frac{(c_1c_2)^{\frac{T}{\tau}}}{(AO)^{\frac{T(\tau-1)}{\tau}}}\right). \tag{5.1}$$

This ratio allows us to analyze the significant improvement in computational complexity induced through a hierarchical decomposition. If our hierarchical decomposition had the same action and observation branching factors $(c_1 = c_2 = 1)$, it would result in a tree that is smaller by a factor of $(AO)^{\frac{T(\tau-1)}{\tau}}$. This gives significant leeway and allows us to, for example, compensate for designing a large number of many-step options.

5.3.3 *Maintaining Feasibility Anytime with Options*

Though combining Lagrangian Monte Carlo Tree Search with dual ascent guides search away from constraint violations in the limit, it does not guarantee anytime constraint satisfaction. When executing a hierarchical controller, runtime monitoring can be used to ensure safety online by terminating options and replanning in case of impending constraint violations. However, unsafe search could still lead to states where safe replanning is not possible. In this section, we show that when options are feasible when executed under an assigned budget for their decision epoch, COBeTS can maintain global feasibility anytime. To show this, we first define local feasibility, one-step global feasibility, and global feasibility.

Definition 1. *An option $\hat{a}_e$ chosen at decision epoch e in b_e is* locally feasible *given a budget $\hat{\mathbf{c}}_e$ if $\mathbf{Q}^{\pi}_{\mathbf{C}}(b_e, \hat{a}_e) \leq \hat{\mathbf{c}}_e$.*

Definition 2. *An option $\hat{a}_e$ chosen at decision epoch e in b_e is* one-step globally feasible *if $\mathbf{C}_{e-1} + \mathbf{Q}^{\pi}_{\mathbf{C},e}(b_e, \hat{a}_e) \leq \hat{\mathbf{c}}$, where $\mathbf{C}_{e-1} = \sum_{e'=0}^{e-1} \gamma^{t_{e'}} \tilde{\mathbf{c}}_{e'} = \sum_{t=0}^{t_e-1} \gamma^t \mathbf{C}(b_t, a_t)$.*

Definition 3. *An algorithm or policy π is said to be* globally feasible *if $\mathbf{V}^{\pi}_{\mathbf{C}}(b_0) \leq \hat{\mathbf{c}}$.*

Informally, a locally feasible option is guaranteed to satisfy a set cost budget while it is in control, a one-step globally feasible option can be applied once and is guaranteed to satisfy the global budget, and global feasibility states that the original constraints from **??** are satisfied.

With these definitions, Proposition 2 below shows that for a CPOSMDP, if a locally feasible option is chosen with a particular assignment of $\hat{\mathbf{c}}_e$, then it ensures that one-step is globally feasible.

Proposition 2. *For policy π and locally feasible option $\hat{a}_e$ at decision epoch e with accumulated costs $\mathbf{C}_{e-1}$, if $\hat{\mathbf{c}}_e = \frac{\hat{\mathbf{c}}-\mathbf{C}_{e-1}}{\gamma^{t_e}} \geq 0$ then $\hat{a}_e$ is one-step globally feasible.*

Proof. By definition of a locally feasible option and the choice of $\hat{\mathbf{c}}_e$:

$$\begin{aligned}
&\mathbf{Q_C}(b_e, \hat{a}_e) \leq \hat{\mathbf{c}}_e = \frac{\hat{\mathbf{c}} - \mathbf{C}_{e-1}}{\gamma^{t_e}} \\
&\mathbf{C}_{e-1} + \gamma^{t_e}\mathbb{E}\Big[\sum_{t=t_e}^{\infty} \gamma^{t-t_e}\mathbf{C}(b_t, a_t) \mid b_e, \hat{a}_e, \pi\Big] \leq \hat{\mathbf{c}} \\
&\mathbf{C}_{e-1} + \mathbf{Q}^{\pi}_{\mathbf{C},e}(b_e, \hat{a}_e) \leq \hat{\mathbf{c}}.
\end{aligned}$$

Thus, by Definition 2, $\hat{a}_e$ is one-step globally feasible. □

These results imply that COBeTS is globally feasible when its options are locally feasible, that is, when they satisfy the budgets passed to `SampleNextOption` (line 11).

Proposition 3. *COBeTS is globally feasible if all its options $\hat{a}_e$ are locally feasible given COBeTS assignments of $\hat{\mathbf{c}}_e \geq 0$ for all e.*

Proof. By construction. Consider any decision epoch e. As given, consider any COBeTS option $\hat{a}_e$. By definition of COBeTS, it assigns $\hat{\mathbf{c}}_e = \left[\frac{\hat{\mathbf{c}}-\mathbf{C}_{e-1}}{\gamma^{t_e}}\right]^+ \geq 0$. By Proposition 2, it is one-step globally feasible. Since this is true for all e, COBeTS is globally feasible. □

5.4 *Experiments*

Our experiments consider online planning in four large safety-critical, partially observable planning problems in order to empirically demonstrate the efficacy of

COBeTS. We compare COBeTS against different non-hierarchical solvers on our target domains, investigate its anytime properties, and show that it can yield better plans even when the number of options far exceeds the number of underlying actions.

We use Julia 1.9 and the POMDPs.jl framework for our experiments [155]. In the following sections, we outline the CPOMDP target problems, briefly describe their hierarchical decompositions, and discuss the main results from our experiments. For full experimentation details, including CPOMDP modeling details, the precise options crafted, and choices of hyperparameters, please refer to our code, which we have open-sourced at `github.com/sisl/COBTSExperiments`.

5.4.1 CPOMDP Problems and Option Policies

We highlight the CPOMDP problem domains used in our experiments below, whether their state, action, and observation spaces are (D)iscrete or (C)ontinuous, and provide an overview of the types of options crafted for execution.

Constrained LightDark [149] (C, D, C)

In this one-dimensional robot localization problem adapted from LightDark [55], [58], the robot must first safely localize itself before navigating to the goal. The robot can move in discrete steps of $\mathcal{A} = \{0, \pm 1, \pm 5, \pm 10\}$ in order to navigate to $s \in [-1, 1]$, take action 0, and receive $+100$ reward, but taking action 0 elsewhere accrues a -100 reward. The robot accrues a per-step reward of -1. The agent starts in the dark region, $b_0 = \mathcal{N}(2, 2^2)$, and can navigate towards the light region at $s = 10$ to help localize itself with less noisy observations. However, the robot must avoid entering a constraint region above $s = 12$ where it will receive a per-step cost of 1 and violate a budget of $\hat{c} = 0.1$. As such, taking the $+10$ action immediately would violate the constraint in expectation.

We template four types of options for this problem. `GoToGoal` greedily navigates the robot's mean position to the goal and terminates. `LocalizeFast` greedily navigates the robot's mean position to the light region until the belief uncertainty is sufficiently small. `LocalizeFromBelow` adjusts the navigation technique for localization so that the robot's mean position does not overshoot the light region. `LocalizeSafe` uses the robot's position uncertainty while localizing to minimize the risk that the robot violates the constraint.

Constrained Spillpoint [149] (C, C, C)

This problem models safe geological carbon capture and sequestration around uncertain subsurface geometries and properties. In the original POMDP [156], instances of CO_2 leaking through faults in the geometry are heavily penalized, both for the presence of a leak and for the total amount leaked. The constrained adaptation imposes a constraint of $\hat{c} = 1 \times 10^{-6}$ to ensure minimal CO_2 leakage.

The options for the spillpoint problem include `InferGeology` and `SafeFill`. The `InferGeology` begins by injecting 90% of the CO_2 volume of the lowest-volume instance of the geology according to the current belief. Then, a sequence of observations of conducted which provides information on the shape of the geology (these observations indicate the presence of CO_2 at various spatial locations). The `InferGeology` is templated so a variety of observation sequences can be selected. The `SafeFill` option involves injecting 90% of the CO_2 volume of the lowest-volume instance of the geology according to the current belief and then terminating the episode. The options (with five versions of `InferGeology`) were combined with the standard set of five individual actions for a total of 11 options.

Constrained Bumper Roomba (C, D, D)

Roomba models a robot with an uncertain initial pose in a fixed environment as it uses its sensors to navigate to a goal region while avoiding a penalty region [153]. In this work, we augment the problem to include a constraint region that the robot must avoid traveling through as it navigates to the goal. States are defined by the continuous pose of the robot on the map, actions allow the robot to turn or move by discrete amounts, and while the robot does not have access to its true pose, in Bumper Roomba, it receives a binary observation when it collides with a wall.

Bumper Roomba crafts three types of options: `TurnAndGo` options turn the robot a fixed amount then navigate until the robot collides with a wall, `GreedyGoToGoal` greedily navigates to the goal using the robot's mean pose, and `SafeGoToGoal` navigates to the goal while imposing a barrier function around the constraint region.

Constrained Lidar Roomba (C, D, C)

This CPOMDP augments Constrained Bumper Roomba with a Lidar sensor that noisily observes the distance to the nearest wall along the robot's heading, with noise proportional to the distance. Rather than using `TurnAndGo` options for localization, Constrained Lidar Roomba implements `Spin` options that turn the robot for different periods of time and with different turn radii to localize.

5.4.2 *Experiments and Discussion*

	LightDark		**Spillpoint**	
Model	$\hat{V}_R$	$\hat{V}_C$ $[\leq 0.1]$	$\hat{V}_R$	$\hat{V}_C$ $[\leq 10^{-6}]$
`COBeTS`	**68.6**±0.7	0.027±0.015	**3.40**±0.66	0.000±0.000
`CPFT-DPW`	5.9±7.7	0.000±0.000	1.50±0.39	0.000±0.000
`CPOMCPOW`	−9.2±8.0	0.032±0.015	1.51±0.40	$1.0 \cdot 10^{-5} \pm 0.9 \cdot 10^{-5}$
	Bumper Roomba		**Lidar Roomba**	
Model	$\hat{V}_R$	$\hat{V}_C$ $[\leq 0.1]$	$\hat{V}_R$	$\hat{V}_C$ $[\leq 0.1]$
`COBeTS`	**5.73**±0.67	0.038±0.038	**5.23**±0.67	0.020±0.019
`CPFT-DPW`	−4.76±0.00	0.036±0.036	−4.57±0.19	0.000±0.000
`CPOMCPOW`	−3.76±0.42	0.000±0.000	−4.55±0.20	0.000±0.000

Table 5.1: Online CPOMDP algorithm demonstrations comparing mean discounted cumulative rewards ($\hat{V}_R$) and costs ($\hat{V}_C$) across 100 LightDark simulations, 10 Spillpoint simulations, and 50 Bumper and Lidar Roomba simulations. COBeTS consistently satisfies constraints while outperforming both the CPFT-DPW baseline and the CPOMCPOW baseline.

CPOMDP algorithm comparison

To evaluate COBeTS, we measure the mean discounted cumulative reward and cost for different planning episodes on the aforementioned CPOMDP domains. We benchmark COBeTS against the closest non-hierarchical online CPOMDP planning algorithms, CPOMCPOW and CPFT-DPW [149], which perform Lagrangian MCTS on the state spaces and belief-state spaces respectively. Table 5.1 summarizes the performance of the algorithms on the different CPOMDP domains, averaged across 100 LightDark, 10 Spillpoint, and 50 Roomba simulations.

In summary, we see that COBeTS is able to significantly outperform against baselines on all domains while satisfying cost constraints. In both Roomba problems, baselines are unable to search deep enough to localize and get to the goal, and instead meander while accruing step penalties to avoid the risk of violating the constraint.

Anytime constraint satisfaction

To highlight the anytime constraint satisfaction with COBeTS we vary the number of search queries in the Constrained LightDark CPOMDP with the robot restricted to `GoToGoal` and two different `LocalizeSafe` options, all of which are feasible from the initial belief. The results averaged across 50 simulations are depicted in Figure 5.2. We see that even with low numbers of search queries, COBeTS satisfies the constraints while achieving high reward, while CPFT-DPW only satisfies the constraints as the number of queries is increased.

Searching over many options

Finally, we investigate the impact that the action branching factor has on search quality in LightDark. We vary the branching factor in the COBeTS search by adding options using different strategies. COBeTS-Unc. adds options that localize to different sampled uncertainties, while COBeTS-Random adds options that execute three randomly selected non-terminal actions. The results, depicted in Figure 5.3, show that COBeTS still finds quality plans with a significant number of options to search over, much greater than the number of actions in the underlying problem (seven). These results support the analysis presented in Equation (5.1), that search complexity reduction from hierarchical decomposition can compensate for increased action branching.

5.5 Discussion

Safe robotic planning under state and transition uncertainty can be naturally expressed as a CPOMDP, but CPOMDPs are extremely difficult to solve in large problem domains. Recent works scaled online search-based CPOMDP planning to large spaces [122], [149], but with limited scope or expertly crafted heuristics. In robotics, planning can often be favorably decomposed hierarchically, separating high-level action primitives and low-level control. In this work, we introduced Constrained Options Belief Tree Search (COBeTS) to improve online CPOMDP planning when favorable hierarchies exist by performing a belief-state Monte Carlo tree search over options. We showed that COBeTS with feasible options will satisfy constraints anytime and demonstrated its success on large planning domains where recent methods fail.

Limitations: A significant limitation of COBeTS is the necessity to craft low-level policies. Recent work uses language models to construct policies automatically [161] and if combined with COBeTS, could enable hierarchical constrained search to compensate for uncertainty in policy generation. A second limitation is that though COBeTS biases the search toward safety, it requires feasible options in order to satisfy constraints anytime. Future work can examine propagating cost bounds [121] or using search heuristics generated from past experience [62], [63], [123] or natural language priors [162] to achieve safety under more general conditions.

This chapter concludes the core contributions of this thesis. The next chapter will summarize the themes presented in this thesis, reiterate our contributions, and discuss possible directions for future work.

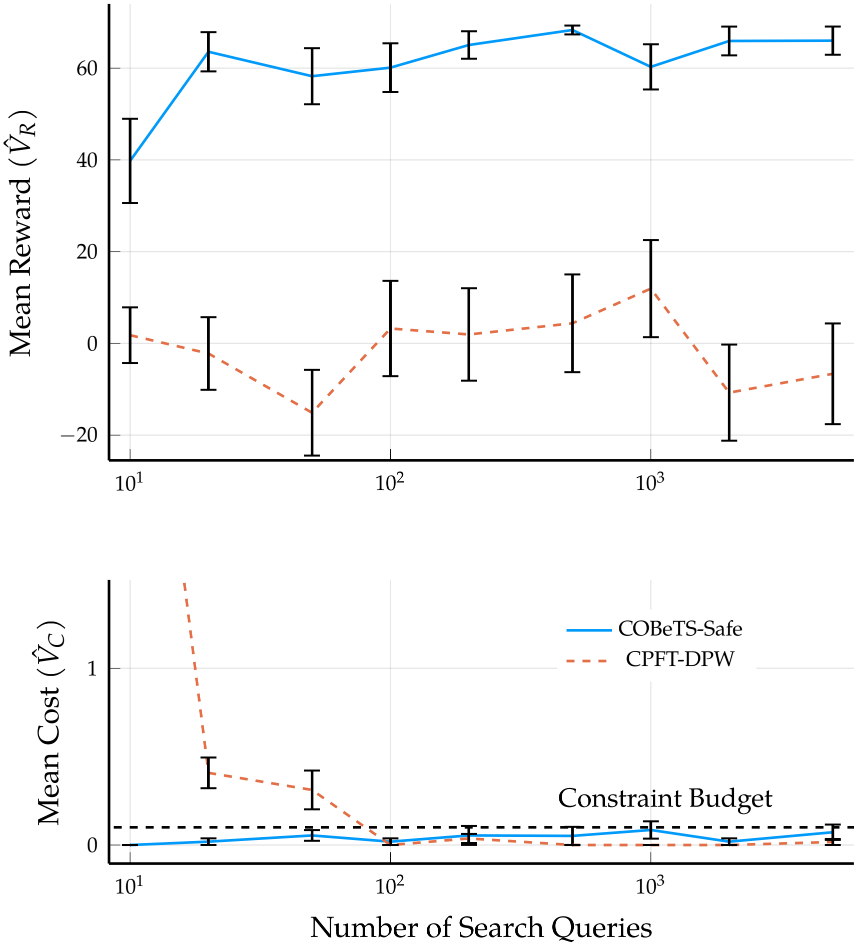


Figure 5.2: Mean cumulative discounted rewards (above) and costs (below) vs. number of tree queries across 50 Constrained LightDark simulations when using COBeTS with feasible options. COBeTS stays safe anytime while CPFT-DPW only satisfies constraints in the limit.

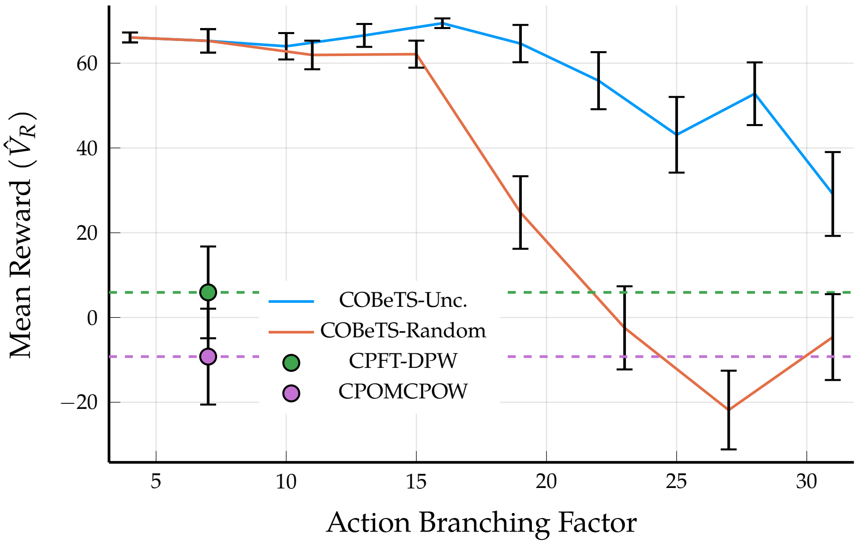


Figure 5.3: Mean cumulative discounted rewards for different numbers of options averaged over 50 Constrained LightDark simulations. All costs are feasible (not shown). COBeTS can retain high reward at larger action branching factors because hierarchy induces a smaller overall tree size.

6 Conclusion

In this chapter, we conclude by overviewing the motivation and main ideas discussed in this thesis, summarizing our primary contributions, and recommending possible directions for future work.

6.1 Summary

Autonomous systems promise to improve efficiency, augment human capabilities, and advance scientific discovery. The ability to reason about environments and plan sequences of decisions is a necessary pillar of autonomy. In complex environments, safe autonomy often requires reasoning about the uncertainty that can arise from environment dynamics and imperfect actuation (outcome uncertainty), incomplete information (state uncertainty), and ambiguous objectives (objective uncertainty). However, even with good models of uncertainty, guaranteeing safe plans is difficult. To improve safe planning under uncertainty, our thesis investigated novel applications of constraints and hierarchies to different problem domains that exhibited outcome, state, and/or objective uncertainty.

First, we investigated imitation learning methods that overcome outcome and objective uncertainty by leveraging expert decision-making demonstration data. We observed that in applications such as autonomous driving, experts demonstrate low-level decisions, while autonomous systems plan hierarchically. As such, we introduced a methodology that relied on low-level demonstrations, information about safe low-level controllers, and an environment simulator in order to infer policies that imitate high-level decision-making. We built a simulator based on real-world driving data in complex environments (roundabouts) to simulate driving counterfactuals. We observed empirically that our method was able to improve safety in these systems, especially in scenarios that were unobserved in the training data.

In the remainder of the thesis, we investigated methods for planning online given predictive models for outcome and state uncertainty. We modeled safety using constraints, framing the safe planning under uncertainty using constrained partially observable Markov decision processes. First, we introduced algorithms that use Monte Carlo tree search to plan online in continuous CPOMDPs by combining progressive widening to limit tree width with dual ascent to satisfy constraints. We showed that these algorithms can plan effectively on continuous CPOMDPs of limited problem scope, including in a safety-critical carbon sequestration domain. Next, we used hierarchical decomposition to scale online continuous CPOMDP planning to even larger problem domains by searching over predefined macro-actions (e.g. low-level control options) and sampling simulated resulting beliefs. Notably, we were able to scale CPOMDP planning to continuous robotic localization domains with safety constraints where non-hierarchical baselines failed to plan successfully. We also showed that under the right conditions, hierarchical CPOMDP planning enables anytime constraint satisfaction, a desirable property for planning in safety-critical systems.

6.2 Contributions

Our main contributions are summarized below.

A method for safety-aware imitation learning that scales to complex problem domains.

In Chapter 3, we introduced Safety-aware Hierarchical Adversarial Imitation Learning (SHAIL) in order to learn how to choose high-level control policies (e.g. macro-actions) that ultimately imitate low-level trajectory data. We build a roundabout simulator using human driving demonstrations from the Interaction dataset to simulate driving in complex environments. We empirically compared the performance of SHAIL against other popular imitation learning methods, demonstrating that it can conveniently plug-and-play a safety layer to significantly improve performance in scenarios not seen during training.

New methods for online planning in large constrained POMDPs.

In Chapter 4, we investigated the application of constraints to POMDP planning in continuous domains. We formulated tree search algorithms that plan online by sampling actions and outcomes to artificial limit tree branching and trend towards constraint satisfaction through dual ascent. We demonstrated that given enough planning time, our algorithms are able to satisfy constraints in large or continuous planning problems, including a safety-critical carbon sequestration problem.

A method for scaling planning to even larger constrained POMDPs using hierarchies.

In Chapter 4, we investigated inducing a hierarchy in order to extend continuous CPOMDP planning to even larger planning problems. We introduce a hierarchical belief-tree search algorithm that was able to solve difficult planning algorithms that our continuous CPOMDP algorithms could not. Importantly, we showed that when low-level controllers can satisfy constraints assigned to them, that COBeTSis guaranteed to satisfy constraints regardless of planning time. Finally, we showed that the reduction in search tree size induced by hierarchical planning allows for search over many more actions than in analogous non-hierarchical problems, motivating it as a method to induce safety in automatically-generated control policies.

6.3 Future Work

This thesis contributed methods for achieving runtime safety by imposing constraints and hierarchies when planning with principled frameworks for modeling uncertainty. Below, following the themes of this thesis, we discuss areas where answers to open questions could significantly improve safe runtime planning under uncertainty in autonomous systems.

Generalization of data-driven decision-making

Chapter 3 discussed data-driven decision-making in autonomous driving, where human demonstration data collected in different scenarios could inspire policy-making in those scenarios. However, a large open question remains in how to ensure that those inferred objectives and learned policies can transfer to entirely different scenarios. One direction of future work addresses generalization through foundation models for decision-making trained on large datasets of decisions in different problems [163]. However, unlike with other fields like natural language, the space of possible decision-making environments and optimal decisions is unbounded, meaning foundation models would likely could likely only be useful in problems related to those they were trained with. Regardless, questions remain on how to ensure that inferred or transferred policies are safe.

Open-loop planning in continuous CPOMDPs

Chapters 4 and 5 discussed methods to scale online closed-loop planning to large POMDPs that imposed safety through constraints. However, in many continuous problems, open-loop planning can often sufficiently approximate closed-loop planning, which can be advantageous as open-loop planning also often provides significant computational advantages. Future work could investigate open-loop planning in continuous CPOMDPs by planning in continuous belief space and imposing constraints using sequential convex programming.

Reasoning about and planning under partial model uncertainty

In this thesis, we often planned while assuming accurate models of uncertainty. However, in many real-world problems, models of uncertainty may be more or less accurate depending on the preconditions. Explicitly reasoning over model accuracy could improve plan safety at runtime, when model errors could skew safety in plans. Future work could investigate safe planning while explicitly reasoning over accuracy about uncertainties, for example through constrained Bayesian reinforcement learning [164] or model-lite planning [165].

Computation for closed-loop planning

Finally, many of the recent advancements in machine learning and control have been facilitated by significant improvements in computational efficiency. Namely, efficient implementations on graphical processing units have enabled significant speedups in the inference of deep neural networks as well as the optimization of open-loop trajectory plans. While many works already address parallelization in closed-loop planning [61], [166], improvements would significantly scale our safe planning capabilities.

Bibliography

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[2] R. Bellman, *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company, 1978.

[3] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[4] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[5] A. K. Jónsson, P. H. Morris, N. Muscettola, and K. Rajan, "Planning in interplanetary space: Theory and practice," in *International Conference on Artificial Intelligence Planning Systems (AIPS)*, 2000.

[6] J. Barreiro, M. Boyce, M. Do, *et al.*, "EUROPA: A platform for AI planning, scheduling, constraint programming, and optimization," in *International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)*.

[7] I. Klein, P. Jonsson, and C. Bäckström, "Efficient planning for a miniature assembly line," *Artificial Intelligence in Engineering*, vol. 13, no. 1, pp. 69–81, 1999.

[8] S. E. Cross and E. Walker, “DART: Applying knowledge-based planning and scheduling to crisis action planning,” *Intelligent Scheduling*, M. Zweben and M. Fox, Eds., pp. 711–29, 1994.

[9] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[10] R. A. Howard, “Dynamic programming and Markov processes.,” 1960.

[11] R. J. Williams and L. C. Baird III, “Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems,” Citeseer, Tech. Rep., 1993.

[12] S. P. Singh and R. C. Yee, “An upper bound on the loss from approximate optimal-value functions,” *Machine Learning*, vol. 16, pp. 227–233, 1994.

[13] H. Kwakernaak and R. Sivan, *Linear Optimal Control Systems*. Wiley-Interscience, 1972, vol. 1.

[14] W. Li and E. Todorov, “Iterative linear quadratic regulator design for nonlinear biological movement systems,” in *International Conference on Informatics in Control, Automation, and Robotics*, vol. 2, 2004.

[15] D. Mayne, “A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems,” *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.

[16] J. T. Betts, “Survey of numerical methods for trajectory optimization,” *AIAA Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[17] T. A. Howell, B. E. Jackson, and Z. Manchester, “ALTRO: A fast solver for constrained trajectory optimization,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[18] A. Mesbah, “Stochastic model predictive control: An overview and perspectives for future research,” *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 30–44, 2016.

[19] A. Y. Ng and M. Jordan, “PEGASUS: A policy search method for large mdps and pomdps,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.

[20] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control*, A. Garulli, A. Tesi, and A. Vicino, Eds., 2007, pp. 207–226.

[21] S. Garatti and M. C. Campi, "Modulating robustness in control design: Principles and algorithms," *IEEE Control Systems Magazine*, vol. 33, no. 2, pp. 36–51, 2013.

[22] M. Wytock, N. Moehle, and S. Boyd, "Dynamic energy management with scenario-based robust MPC," in *American Control Conference (ACC)*, 2017.

[23] D. S. Nau, V. Kumar, and L. Kanal, "General branch and bound, and its relation to a* and ao*," *Artificial Intelligence*, vol. 23, no. 1, pp. 29–58, 1984.

[24] M. Kearns, Y. Mansour, and A. Y. Ng, "A sparse sampling algorithm for near-optimal planning in large Markov decision processes," *Machine learning*, vol. 49, pp. 193–208, 2002.

[25] C. B. Browne, E. Powley, D. Whitehouse, *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[26] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European Conference on Machine Learning (ECML)*, 2006.

[27] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous upper confidence trees," in *International Conference on Learning and Intelligent Optimization (LION)*, 2011.

[28] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[29] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 2018.

[30] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," vol. 33, 2020.

[31] D. P. Bertsekas and J. N. Tsitsiklis, "An analysis of stochastic shortest path problems," *Mathematics of Operations Research*, vol. 16, no. 3, pp. 580–595, 1991.

[32] B. Bonet and H. Geffner, "Labeled RTDP: Improving the convergence of real-time dynamic programming.," in *International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 3, 2003.

[33] E. A. Hansen and S. Zilberstein, "Lao*: A heuristic search algorithm that finds solutions with loops," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 35–62, 2001.

[34] C. I. Connolly and R. A. Grupen, "The applications of harmonic functions to robotics," *Journal of Robotic Systems*, vol. 10, no. 7, pp. 931–946, 1993.

[35] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[36] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[37] R. Kalman, "A new approach to linear filtering and prediction problems," *ASME Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960.

[38] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium*, 2000.

[39] "Novel approach to nonlinear/non-gaussian bayesian state estimation," in *IEE Proceedings F (Radar and Signal Processing)*, vol. 140, 1993.

[40] A. Doucet, N. De Freitas, N. J. Gordon, *et al.*, *Sequential Monte Carlo Methods in Practice*. Springer, 2001, vol. 1.

[41] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of Markov decision processes," *Mathematics of Operations Research*, vol. 12, no. 3, pp. 441–450, 1987.

[42] E. J. Sondik, "The optimal control of partially observable markov processes over the infinite horizon: Discounted costs," *Operations Research*, vol. 26, no. 2, pp. 282–304, 1978.

[43] M. T. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for POMDPs," *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.

[44] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[45] H. Kurniawati, D. Hsu, and W. S. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Robotics: Science and Systems*, 2008.

[46] E. A. Hansen, "Solving POMDPs by searching in policy space," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1998.

[47] P. Poupart and C. Boutilier, "Bounded finite state controllers," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 16, 2003.

[48] C. Amato, D. S. Bernstein, and S. Zilberstein, "Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

[49] N. Meuleau, K.-E. Kim, L. P. Kaelbling, and A. R. Cassandra, "Solving POMDPs by searching the space of finite policies," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999.

[50] H. Bai, D. Hsu, W. S. Lee, and V. A. Ngo, "Monte Carlo value iteration for continuous-state POMDPs," in *Algorithmic Foundations of Robotics IX: Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics*, 2011.

[51] H. Bai, D. Hsu, M. J. Kochenderfer, and W. S. Lee, "Unmanned aircraft collision avoidance using continuous-state POMDPs," in *Robotics: Science and Systems*, 2011.

[52] P. Poupart and C. Boutilier, "Value-directed compression of POMDPs," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 15, 2002.

[53] N. Roy, G. Gordon, and S. Thrun, "Finding approximate POMDP solutions through belief compression," *Journal of Artificial Intelligence Research*, vol. 23, pp. 1–40, 2005.

[54] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference (ACC)*, 2005.

[55] R. Platt Jr, R. Tedrake, L. Kaelbling, and T. Lozano-Perez, "Belief space planning assuming maximum likelihood observations," in *Robotics: Science and Systems*, 2010.

[56] S. Ross, B. Chaib-Draa, *et al.*, "AEMS: An anytime online search algorithm for approximate policy refinement in large POMDPs," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[57] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2010.

[58] Z. N. Sunberg and M. J. Kochenderfer, "Online algorithms for POMDPs with continuous state, action, and observation spaces," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.

[59] M. H. Lim, C. J. Tomlin, and Z. N. Sunberg, "Voronoi progressive widening: Efficient online solvers for continuous state, action, and observation POMDPs," in *IEEE Conference on Decision and Control (CDC)*, 2021.

[60] A. Somani, N. Ye, D. Hsu, and W. S. Lee, *DESPOT: Online POMDP planning with regularization*, 2013.

[61] P. Cai, Y. Luo, D. Hsu, and W. S. Lee, "HyP-DESPOT: A hybrid parallel algorithm for online planning under uncertainty," *International Journal of Robotics Research*, vol. 40, no. 2-3, pp. 558–573, 2021.

[62] P. Cai and D. Hsu, "Closing the planning–learning loop with application to autonomous driving," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 998–1011, 2022.

[63] R. J. Moss, A. Corso, J. Caers, and M. J. Kochenderfer, "BetaZero: Belief-state planning for long-horizon POMDPs using learned approximations," 2023. arXiv: `2306.00249 [cs.AI]`.

[64] J. Mern, A. Yildiz, Z. Sunberg, T. Mukerji, and M. J. Kochenderfer, "Bayesian optimized Monte Carlo planning," in *AAAI Conference on Artificial Intelligence* (*AAAI*), vol. 35, 2021.

[65] S. C. Ong, S. W. Png, D. Hsu, and W. S. Lee, "Planning under uncertainty for robotic tasks with mixed observability," *International Journal of Robotics Research*, vol. 29, no. 8, pp. 1053–1068, 2010.

[66] M. Araya, O. Buffet, V. Thomas, and F. Charpillet, "A POMDP extension with belief-dependent rewards," in *Advances in Neural Information Processing Systems* (*NIPS*), vol. 23, 2010.

[67] L. Dressel and M. Kochenderfer, "Efficient decision-theoretic target localization," in *International Conference on Automated Planning and Scheduling* (*ICAPS*), vol. 27, 2017.

[68] J. Fischer and Ö. S. Tas, "Information particle filter tree: An online algorithm for POMDPs with belief-based rewards on continuous domains," in *International Conference on Machine Learning* (*ICML*), 2020.

[69] M. Chen, E. Frazzoli, D. Hsu, and W. S. Lee, "POMDP-lite for robust robot planning under uncertainty," in *IEEE International Conference on Robotics and Automation* (*ICRA*), 2016.

[70] J. S. Dibangoye, C. Amato, O. Buffet, and F. Charpillet, "Optimally solving Dec-POMDPs as continuous-state MDPs," *Journal of Artificial Intelligence Research*, vol. 55, pp. 443–497, 2016.

[71] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," 2016. arXiv: `1606.06565 [cs.AI]`.

[72] G. Swamy, S. Choudhury, J. A. Bagnell, and S. Wu, "Of moments and matching: A game-theoretic framework for closing the imitation gap," in *International Conference on Machine Learning (ICML)*, 2021.

[73] S. K. S. Ghasemipour, R. Zemel, and S. Gu, "A divergence minimization perspective on imitation learning methods," in *Conference on Robot Learning (CoRL)*, 2020.

[74] I. Kostrikov, O. Nachum, and J. Tompson, "Imitation learning via off-policy distribution matching," in *International Conference on Learning Representations (ICLR)*, 2020.

[75] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2004.

[76] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, "Maximum entropy inverse reinforcement learning.," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 8, 2008.

[77] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Human behavior modeling with maximum entropy inverse optimal control.," in *AAAI Spring Symposium: Human Behavior Modeling*, vol. 92, 2009.

[78] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[79] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adverserial inverse reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2018.

[80] D. Garg, S. Chakraborty, C. Cundy, J. Song, and S. Ermon, "IQ-learn: Inverse soft-Q learning for imitation," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 4028–4039, 2021.

[81] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys*, vol. 50, no. 2, pp. 1–35, 2017.

[82] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun, "Survey of imitation learning for robotic manipulation," *International Journal of Intelligent Robotics and Applications*, vol. 3, pp. 362–369, 2019.

[83] V. Alexiadis, J. Colyar, J. Halkias, R. Hranac, and G. McHale, "The next generation simulation program," *Institute of Transportation Engineers (ITE) Journal*, vol. 74, no. 8, p. 22, 2004.

[84] W. Zhan, L. Sun, D. Wang, *et al.*, "INTERACTION Dataset: An INTERNational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps," *arXiv:1910.03088 [cs, eess]*, Sep. 2019.

[85] J. Bock, R. Krajewski, T. Moers, S. Runde, L. Vater, and L. Eckstein, "The InD dataset: A drone dataset of naturalistic road user trajectories at german intersections," in *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[86] P. Sun, H. Kretzschmar, X. Dotiwalla, *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[87] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Tech. Rep., 2018.

[88] T. Kanade, C. Thorpe, and W. Whittaker, "Autonomous land vehicle project at cmu," in *ACM Conference on Computer Science*, 1986.

[89] E. D. Dickmanns and A. Zapp, "Autonomous high speed road vehicle guidance by computer vision," pp. 221–226, 1987.

[90] D. A. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems (NIPS)*, 1989.

[91] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," 2016. arXiv: `1605.06450 [cs.LG]`.

[92] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer, "Imitating driver behavior with generative adversarial networks," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[93] J. Song, H. Ren, D. Sadigh, and S. Ermon, "Multi-agent generative adversarial imitation learning," vol. 31, 2018.

[94] R. P. Bhattacharyya, D. J. Phillips, B. Wulfe, J. Morton, A. Kuefler, and M. J. Kochenderfer, "Multi-agent imitation learning for driving simulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[95] R. P. Bhattacharyya, D. J. Phillips, C. Liu, J. K. Gupta, K. Driggs-Campbell, and M. J. Kochenderfer, "Simulating emergent properties of human driving behavior using multi-agent reward augmented imitation learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[96] S. Javdani, S. S. Srinivasa, and J. A. Bagnell, "Shared autonomy via hindsight optimization," 2015.

[97] H. Le, N. Jiang, A. Agarwal, M. Dudik, Y. Yue, and H. Daumé III, "Hierarchical imitation and reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2018.

[98] M. Jing, W. Huang, F. Sun, *et al.*, "Adversarial option-aware hierarchical imitation learning," in *International Conference on Machine Learning (ICML)*, vol. 139, 2021.

[99] E. Bronstein, M. Palatucci, D. Notz, *et al.*, "Hierarchical model-based imitation learning for planning in autonomous driving," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[100] D. Xu, Y. Chen, B. Ivanovic, and M. Pavone, "BITS: Bi-level imitation for traffic simulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[101] N. J. Nilsson, *The Quest for Artificial Intelligence*. Cambridge University Press, 2009.

[102] S. Kammel, J. Ziegler, B. Pitzer, *et al.*, "Team AnnieWAY's autonomous system for the 2007 DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 615–639, 2008.

[103] R. E. Fikes, P. E. Hart, and N. J. Nilsson, "Learning and executing generalized robot plans," *Artificial intelligence*, vol. 3, pp. 251–288, 1972.

[104] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 10, 1997.

[105] T. G. Dietterich, "The MAXQ method for hierarchical reinforcement learning.," in *International Conference on Machine Learning (ICML)*, vol. 98, 1998.

[106] E. D. Sacerdoti, "Planning in a hierarchy of abstraction spaces," *Artificial Intelligence*, vol. 5, no. 2, pp. 115–135, 1974.

[107] E. D. Sacerdoti, "The nonlinear nature of plans," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1975.

[108] A. Tate, "Interacting goals and their use," in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 10, 1975.

[109] A. Tate, "Generating project networks," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1977.

[110] K. Currie and A. Tate, "O-PLAN: The open planning architecture," *Artificial Intelligence*, vol. 52, no. 1, pp. 49–86, 1991.

[111] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," *Journal of Artificial Intelligence Research*, vol. 48, pp. 67–113, 2013.

[112] V. Pareto, *Manuale di Economia Politica*. Società Editrice Libraria, 1906.

[113] K. Wray, S. Zilberstein, and A.-I. Mouaddib, "Multi-objective MDPs with conditional lexicographic reward preferences," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 29, 2015.

[114] J. D. Isom, S. P. Meyn, and R. D. Braatz, "Piecewise linear dynamic programming for constrained POMDPs," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2008.

[115] K. H. Wray, S. Tiomkin, M. J. Kochenderfer, and P. Abbeel, "Multi-objective policy gradients with topological constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[116] E. Altman, *Constrained Markov Decision Processes*. CRC Press, 1999, vol. 7.

[117] D. Kim, J. Lee, K.-E. Kim, and P. Poupart, "Point-based value iteration for constrained POMDPs," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[118] P. Poupart, A. Malhotra, P. Pei, K.-E. Kim, B. Goh, and M. Bowling, "Approximate linear programming for constrained partially observable Markov decision processes," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 29, 2015.

[119] E. Walraven and M. T. Spaan, "Column generation algorithms for constrained POMDPs," *Journal of Artificial Intelligence Research*, vol. 62, pp. 489–533, 2018.

[120] K. H. Wray and K. Czuprynski, "Scalable gradient ascent for controllers in constrained POMDPs," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[121] A. Undurti and J. P. How, "An online algorithm for constrained POMDPs," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[122] J. Lee, G.-H. Kim, P. Poupart, and K.-E. Kim, "Monte-Carlo tree search for constrained POMDPs," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, 2018.

[123] D. Parthasarathy, G. Kontes, A. Plinge, and C. Mutschler, "C-MCTS: Safe planning with Monte Carlo tree search," 2023. arXiv: `2305.16209 [cs.LG]`.

[124] S. Thiébaux, B. Williams, *et al.*, "RAO*: An algorithm for chance-constrained POMDPs," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 30, 2016.

[125] J. Chen, Y. Shimizu, L. Sun, M. Tomizuka, and W. Zhan, "Constrained iterative LQG for real-time chance-constrained gaussian belief space planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[126] A. Majumdar and M. Pavone, "How should a robot assess risk? towards an axiomatic theory of risk in robotics," in *International Symposium of Robotic Research*, 2020.

[127] A. Hakobyan, G. C. Kim, and I. Yang, "Risk-aware motion planning and control using CVaR-constrained optimization," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3924–3931, 2019.

[128] A. Jamgochian, E. Buehrle, J. Fischer, and M. J. Kochenderfer, "SHAIL: Safety-aware hierarchical adversarial imitation learning for autonomous driving in urban environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[129] B. Mirchevska, M. Hügle, G. Kalweit, M. Werling, and J. Boedecker, "Amortized Q-learning with model-based action proposals for autonomous driving on highways," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[130] D. Kamran, Y. Ren, and M. Lauer, "High-level decisions from a safe maneuver catalog with reinforcement learning for safe and cooperative automated merging," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2021.

[131] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[132] S. Krishnan, A. Garg, R. Liaw, L. Miller, F. T. Pokorny, and K. Goldberg, "HIRL: Hierarchical inverse reinforcement learning for long-horizon tasks with delayed rewards," 2016. arXiv: `1604.06508 [cs.RO]`.

[133] M. Sharma, A. Sharma, N. Rhinehart, and K. M. Kitani, "Directed-Info GAIL: Learning hierarchical policies from unsegmented demonstrations using directed information," in *International Conference on Learning Representations*, 2018.

[134] P. Sharma, D. Pathak, and A. Gupta, "Third-person visual imitation learning via decoupled hierarchical controller," in *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[135] R. Fox, R. Berenstein, I. Stoica, and K. Goldberg, "Multi-task hierarchical imitation learning for home automation," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2019.

[136] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[137] F. Belletti, D. Haziza, G. Gomes, and A. M. Bayen, "Expert level control of ramp metering based on multi-task deep reinforcement learning," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2017.

[138] D. Kamran, C. Fernandez Lopez, M. Lauer, and C. Stiller, "Risk-aware high-level decisions for automated driving at occluded intersections with reinforcement learning," in *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[139] J. Chen, B. Yuan, and M. Tomizuka, "Model-free deep reinforcement learning for urban autonomous driving," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[140] J. Hawke, R. Shen, C. Gurau, *et al.*, "Urban driving with conditional imitation learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[141] J. Chen, Z. Wang, and M. Tomizuka, "Deep hierarchical reinforcement learning for autonomous driving with distinct behaviors," in *IEEE Intelligent Vehicles Symposium (IV)*, 2018.

[142] P. Henderson, W.-D. Chang, P.-L. Bacon, D. Meger, J. Pineau, and D. Precup, "OptionGAN: Learning joint reward-policy options using generative adversarial inverse reinforcement learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[143] L. Wang, Y. Hu, L. Sun, W. Zhan, M. Tomizuka, and C. Liu, "Transferable and adaptable driving behavior prediction," *arXiv:2202.05140 [cs]*, 2022.

[144] L. Sun, C. Peng, W. Zhan, and M. Tomizuka, "A fast integrated planning and control framework for autonomous driving via imitation learning," in *Dynamic Systems and Control Conference*, 2018.

[145] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning (ICML)*, 2015.

[146] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347 [cs]*, 2017.

[147] G. Brockman, V. Cheung, L. Pettersson, *et al.*, "OpenAI gym," *arXiv:1606.01540 [cs]*, 2016.

[148] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical Review E*, vol. 62, no. 2, pp. 1805–1824, 2000.

[149] A. Jamgochian, A. Corso, and M. J. Kochenderfer, "Online planning for constrained POMDPs with continuous spaces through dual ascent," in *International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 33, 2023.

[150] M. J. Kochenderfer, T. A. Wheeler, and K. H. Wray, *Algorithms for Decision Making*. MIT Press, 2022.

[151] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa, "Online planning algorithms for POMDPs," *Journal of Artificial Intelligence Research*, vol. 32, pp. 663–704, 2008.

[152] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. V. D. Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte Carlo tree search," *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.

[153] C. Wu, G. Yang, Z. Zhang, *et al.*, "Adaptive online packing-guided search for POMDPs," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 28 419–28 430, 2021.

[154] A. B. Piunovskiy and X. Mao, "Constrained Markovian decision processes: The dynamic programming approach," *Operations Research Letters*, vol. 27, no. 3, pp. 119–126, 2000.

[155] M. Egorov, Z. N. Sunberg, E. Balaban, T. A. Wheeler, J. K. Gupta, and M. J. Kochenderfer, "POMDPs.jl: A framework for sequential decision making under uncertainty," *Journal of Machine Learning Research*, vol. 18, no. 26, pp. 1–5, 2017.

[156] A. Corso, Y. Wang, M. Zechner, J. Caers, and M. J. Kochenderfer, "A POMDP model for safe geological carbon sequestration," in *Advances in Neural Information Processing Systems (NeurIPS)*, ser. Tackling Climate Change with Machine Learning Workshop, 2022.

[157] A. Jamgochian, H. Buurmeijer, K. H. Wray, A. Corso, and M. J. Kochenderfer, "Constrained hierarchical monte carlo belief-state planning," 2023. arXiv: 2310.20054 [cs.AI].

[158] N. A. Vien and M. Toussaint, "Hierarchical Monte-Carlo planning," in *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 29, 2015.

[159] S.-K. Kim, A. Bouman, G. Salhotra, *et al.*, "PLGRIM: Hierarchical value learning for large-scale exploration in unknown environments," in *International Conference on Automated Planning and Scheduling (ICAPS)*, vol. 31, 2021.

[160] S. Feyzabadi and S. Carpin, "Planning using hierarchical constrained Markov decision processes," *Autonomous Robots*, vol. 41, pp. 1589–1607, 2017.

[161] J. Liang, W. Huang, F. Xia, *et al.*, "Code as policies: Language model programs for embodied control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[162] Z. Zhao, W. S. Lee, and D. Hsu, "Large language models as commonsense knowledge for large-scale task planning," 2023. arXiv: 2305.14078 [cs.RO].

[163] S. Yang, O. Nachum, Y. Du, J. Wei, P. Abbeel, and D. Schuurmans, "Foundation models for decision making: Problems, methods, and opportunities," 2023. arXiv: `2303.04129 [cs.AI]`.

[164] J. Lee, Y. Jang, P. Poupart, and K.-E. Kim, "Constrained bayesian reinforcement learning via approximate linear programming.," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[165] H. H. Zhuo and S. Kambhampati, "Model-lite planning: Case-based vs. model-based approaches," *Artificial Intelligence*, vol. 246, pp. 1–21, 2017.

[166] G. M. J. B. Chaslot, M. H. M. Winands, and H. J. van Den Herik, "Parallel Monte-Carlo tree search," in *International Conference on Computers and Games*, 2008.