Sequential Decision Making under Uncertainty: Optimality Guarantees, Compositional Learning, and Applications to Robotics and Ecology

By

Hyun Jae Lim

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Claire J. Tomlin, Chair
Assistant Professor Zachary N. Sunberg
Assistant Professor Benjamin W. Blonder
Professor Pieter Abbeel

Spring 2023

Sequential Decision Making under Uncertainty: Optimality Guarantees, Compositional Learning, and Applications to Robotics and Ecology

Abstract

Sequential Decision Making under Uncertainty: Optimality Guarantees, Compositional Learning, and Applications to Robotics and Ecology

by

Hyun Jae Lim

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Claire J. Tomlin, Chair

Sequential decision making under uncertainty problems often deal with partially observable Markov decision processes (POMDPs). POMDPs mathematically capture making decisions at each step while accounting for potential rewards and uncertainties an agent may encounter in the future, which make them desirable and flexible representations of many real world problems. However, such sequential decision making problems with various sources of uncertainty are notoriously difficult to solve, especially when the state and observation spaces are continuous or hybrid, which is often the case for physical systems. Furthermore, modern problem settings require sophisticated machine learning techniques to effectively handle complex data structures like image, text or audio inputs, while performing complicated reasoning such as localizing with noisy camera images or predicting intentions and locations of other agents. Modern approaches that involve artificial intelligence and machine learning methods provide powerful computational resources that can effectively manage the above challenges. Many of these decision making algorithms and machine learning techniques can either capture rigorous theoretical guarantees or empirical performance, but few capture both.

This dissertation aims to lay the foundations to study sequential decision making under uncertainty from multiple angles: theoretical guarantees, integration with learning, and real world applications. We strike a balance between mathematical analysis of the foundational framework of POMDPs, and enabling and deploying these techniques via integration with machine learning techniques through compositional learning.

We first begin the theoretical portion of the dissertation by analyzing novel POMDP solvers and their theoretical convergence properties. This portion introduces a several novel POMDP algorithms that serve as foundations for studying convergence properties of modern POMDP algorithms when dealing with continuous observation and action spaces. Then, we cover a more general result that provides theoretical guarantees and justification for solving the

particle belief approximation of POMDPs while retaining guarantees in the original POMDP. This result formally justifies a common POMDP approximation technique known as the particle likelihood weighting, which is the first-of-its-kind in theoretically explaining a family of modern POMDP algorithms that use this technique.

Then, we introduce approaches to integrate model-based planning with learning-based components via compositional learning for real world robotic settings. First, we study how to integrate the aforementioned POMDP planning algorithms with machine learning components by using deep generative models, which enables these algorithms to tackle visual navigation tasks. Second, we substantially extend a robotic arm manipulation algorithm for tabletop manipulation through reasoning with demonstration sequences and weighted multi-task learning.

Lastly, we propose a novel application area of sequential decision making in ecological subfield of community state navigation. Specifically, we focus on formulating the species coexistence navigation problem as an optimal path planning problem. This approach allows us to understand the population dynamics by analyzing small perturbations to the equilibrium states and subsequently find action sequences that allow efficient navigation. We also discuss the benefits and impact of applying sequential decision making framework to community state navigation problems and beyond.

Afterwards, we summarize the main contributions once again and contextualize the novel contributions. We also discuss some opportunities for future works in sequential decision making under uncertainty, in terms of new theoretical developments, alternative approaches for compositional learning, and other avenues for impactful real world applications.

Contents

List of Figures

List of Tables

Acknowledgments

I'd like to thank my advisors, Profs. Claire Tomlin and Zachary Sunberg, and my dissertation and qual exam committee members, Profs. Benjamin Blonder, Pieter Abbeel, and Venkatachalam Anantharam. I am infinitely grateful for your support and guidance.

My wonderful collaborators include: Claire J. Tomlin, Zachary N. Sunberg, Benjamin W. Blonder, Aleksandra Faust, Stefan Schaal, Mykel J. Kochenderfer, Andy Zeng, Brian Ichter, Maryam Bandari, Erwin Coumans, Tyler J. Becker, Sampada Deglurkar, and Johnathan Tucker. It has been a huge honor and pleasure working with every one on various projects.

The generous funding sources for the authors of the projects include: the National Science Foundation Graduate Research Fellowship under Grant Nos. DGE 1752814 and DGE 2146752, DARPA Assured Autonomy Grant, the SRC CONIX program, NSF CPS Frontiers, the University of Colorado Boulder, Google-BAIR Commons AI Research Grant, the ONR Embedded Humans MURI, ONR Multibody Control Systems Basic Research Challenge, and NSF Rules of Life program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any aforementioned organizations.

I am thankful for the following colleagues for valuable manuscript feedback, helping initial idea formulations, and engaging in fruitful discussions: Lasse Peters, Gregory Lawrence, Katharina Muelling, Simon Stepputtis, Azarakhsh Keipour, Chelsea Finn, Courtenay Ray, Pierre Gaüzère, David Ackerly, Carl Boettiger, Sampada Deglurkar, Shankar Deka, and many anonymous reviewers. I am grateful for my colleagues whose public datasets and codebases made many empirical analyses possible. I also thank John Mern for sharing the source code for BOMCP and the lunar lander environment and providing valuable insights as well as technical support for running and verifying these scripts, and Daniel Maynard for running additional analyses to provide us with empirical parameter sets.

Thank you family, advisors, collaborators, friends, and my extensive collection of guitars. I wouldn't be here without you.

Part I

Introduction and Background

Chapter 1

Introduction

Chapter Outline

This chapter provides a short introduction to the dissertation and its contributions. It also defines key notations and formalism that may be useful for reading each chapter of the dissertation.

1.1 This Work in Perspective

This dissertation is rooted in ideas from **sequential decision making under uncertainty**, most often in dealing with partially observable Markov decision processes (POMDPs), which is a general formalism that can represent many of these sequential decision making under uncertainty problems. POMDPs mathematically capture the challenge of making decisions at each step, while accounting for potential rewards and uncertainties an agent may encounter in the future. Consequently, sequential decision making under uncertainty problems require agents to reason about uncertainty that can come in many forms: How will the system evolve in the next step after taking a certain action? How can the agent distill information about our current location or configuration from noisy or incomplete information provided by the environment? What is the best action to complete the task while minimizing failure risk?

As modern problem settings become incredibly complex both in the environmental setup, such as dealing with image, text, or audio inputs, as well as the reasoning, such as navigating with noisy camera readings or predicting community state evolution, it is imperative that these decision making methods can adapt to reasoning about and eventually solving these problems. While modern technologies like artificial intelligence and machine learning methods provide powerful tools to handle many of these issues, it is important to provide some level of performance guarantees and assurances to be useful in many real world applications. Unfortunately, many decision making solvers and systems attain either empirical performance or theoretical guarantees, but few capture both. The challenge to simultaneously

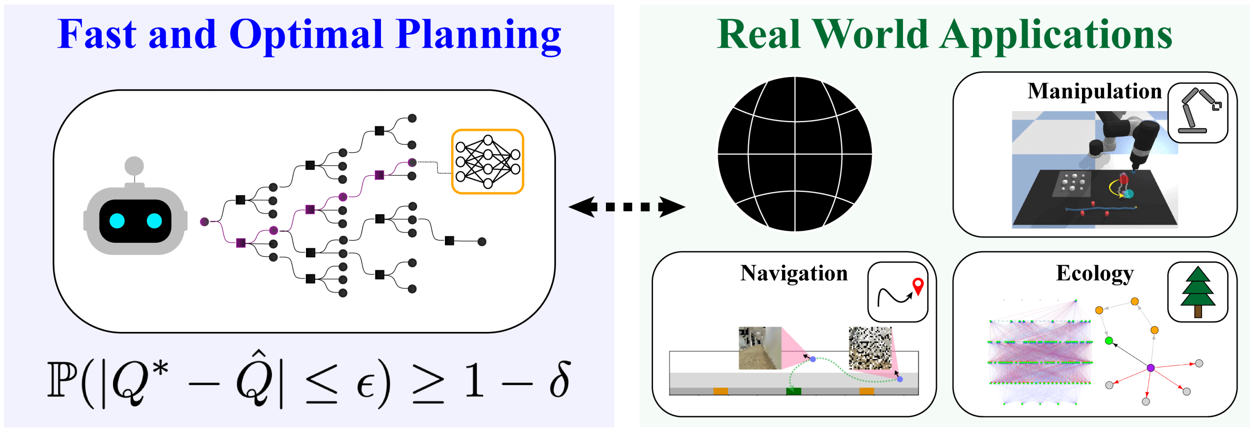

Fast and Optimal Planning
$\mathbb{P}(|Q^* - \hat{Q}| \leq \epsilon) \geq 1 - \delta$
Real World Applications
Manipulation
Navigation
Ecology

Figure 1.1: **Overview of the dissertation: sequential decision making under uncertainty.** We first theoretically motivate the sequential decision making under uncertainty, and then dive into various extensions using compositional learning and applications in robotics and ecology.

mathematically represent and formally justify many modern machine learning techniques and their applications to sequential decision making has become incredibly difficult, due to the complexity of these techniques as well as their applications.

The dissertation aims to strike a balance between mathematically justifying foundational framework of POMDPs, and enabling these techniques to be integrated with machine learning technologies in a way that preserves the system structure that enjoys these guarantees as much as possible. In essence, not only are the theoretical foundations of the finite sample optimality guarantees of decision making methods important, but also these methods should be made further useful through integration with modern machine learning and artificial intelligence technologies to enable applications to more realistic problem domains including robotics and ecology. In this sense, the approach of compositional learning is extremely promising in balancing theoretical rigor and computational power: the prior knowledge of the problem structure and domain knowledge is distilled into an appropriate sequential decision making problem, where the model components are bolstered by enabling individual and/or simultaneous learning of these components.

Consequently, the core philosophy of the dissertation is the following:

> *In order for fast and optimal sequential decision making under uncertainty methods to be useful in real world applications, they should be rigorously founded through mathematical proofs and guarantees and empowered through integration with modern technologies.*

Thus, we aim to devise solutions and systems that are both theoretically well-justified in the formulation, most often in the form of statistical guarantees, and computationally efficient and accurate. With these frameworks in mind, we also seek ways in which the sequential decision making framework could be useful in previously under-explored application areas, in particular ecology problems regarding species coexistence navigation.

1.2 Contributions and Outline

This dissertation provides three different lenses into the topic of decision making under uncertainty: theory, computation, and real world applications. In Part I Chapters 2 and 3, we will introduce the necessary mathematical background to understand the ideas in this dissertation, and provide additional materials that cover some of the more recent developments in the domain. The following parts and chapters outline the key contributions on approaching the subject of sequential decision making under uncertainty in three different lenses. Since this dissertation also serves as a compilation of the author's previously published works, we introduce each main part with an overview chapter that concisely outlines what the main contributions are, why they are significant, and how they fit into the overall narrative of studying sequential decision making under uncertainty.

Part II – Theory: Fast and Optimal Online Algorithms for POMDPs.
In Part II, we establish important theoretical properties of online algorithms for POMDPs. We present various theoretically motivated contributions on tackling POMDPs with continuous observations and continuous/hybrid actions. These techniques heavily utilize sampling-based techniques and the corresponding statistical analysis. We cover the following novel contributions:

- **Partially observable weighted sparse sampling (POWSS)**, which is a particle belief likelihood algorithm for continuous observation POMDPs with finite sample near-optimality results, justifying the commonly used particle likelihood weighting technique. Notably, this result does not rely on any discretization schemes nor directly depend on the size of the observation space.
- **A general result about Particle Belief MDP approximation of POMDPs**, which bridges the gap between theoretically solving POMDPs and practical POMDP algorithms that use simplifying POMDP representations. The theoretical results heavily leverage Sparse Sampling-ω, which is a simplified and updated version of POWSS that also enjoys much more desirable theoretical guarantees.
- **Voronoi Progressive Widening (VPW)**, which is the first known continuous space POMDP algorithm with theoretical guarantees (VOWSS algorithm) and practical extensions (VOMCPOW algorithm).

Part III – Learning and Robotics: Compositional Learning-based Planning with Algorithmic Priors.
In Part III, we integrate principles from sequential decision making and robotic motion

planning with modern machine learning techniques. Specifically, we discuss the *compositional learning* approach for decision making and planning, which strikes a balance between maintaining theoretical guarantees or domain knowledge structure, and leveraging the representation power of machine learning. We cover the following novel contributions:

- **Visual Tree Search**, which is a planning system that integrates particle filter and POMDP planners with learning-enabled components to tackle POMDPs with complex and noisy visual observations.
- **Sequence-Conditioned Transporter Networks**, which extends Transporter Networks system to reason with demonstration sequences through multi-task weighted learning to handle multi-task tabletop robotic arm manipulation problems.

Part IV – Ecology Application: Novel Ecological Applications of Optimal Path Planning.
In Part IV, we discuss novel applications of sequential decision making under uncertainty approach in the field of community state evolution. This section discusses some possible ways to formulate the species coexistence navigation problem as a sequential decision making problem, and encourage further collaborations between these two fields. We cover the following novel contributions:

- **Navigation between community states via shortcuts**, which enables community evolution via small sequential additions and deletions of species to nudge the dynamic system into desired basins and eventually the goal, through constructing the state diagram that represents possible community evolution pathways.

1.3 Notations and Formalism

Since this dissertation unifies three related yet distinct approaches to studying sequential decision making under uncertainty, each part and chapter has different notations and formalism. In order to remedy the notation discrepancy between different chapters, we provide Tables 1.1 and 1.2 of general notations that are most often followed in the chapters, and Table 1.3 of specific notations that are mostly used in specific chapters only.

Notation	Short Description
s, o, a, r	State, observation, action, reward
D	Horizon of the planning problem
d	Given depth of the tree/estimator.
	Mostly used for variables internal to the planner
t	Given time step of the environment.
	Mostly used for variables that interact with the environment
$b, b(s)$	Belief, belief state, or belief density
h	History of states, actions, and/or observations
$\bar{b}$	Particle belief, consisting of state particles and weights
w	Likelihood (importance sampling) weights for a particle
G	Generative model
ϵ	Concentration worst case error bound
α, β	Concentration worst case error bound for inductive steps (α - POMDP, β - PB-MDP)
δ	Concentration worst case probability bound
$\mathbb{P}, \mathbb{E}$	Probability and expectation operators
$\mathbb{R}^n$	n-dimensional Real space
V, Q	Value functions
$\mathcal{P}, \mathcal{Q}, \ldots$	Probability densities denoted with calligraphic font

Table 1.1: **General Notations - Variables and Quantities.** Summary of important notations and formalism used throughout this dissertation, for variables and quantities.

Notation	Short Description
$\square_d$	Step d of planning
$\square_t$	Time t of planning, estimation, or execution
$\square_i$	Variable or data indexed by i
$\square_{1:d}$	Product of the density from step 1 through d for a given sequence
$\square'$	Next quantity of $\square$, usually used to reason about next states and beliefs
$\square^*$	Optima of the quantity, usually the maximum
$\{\square\}$	Sequence of quantities
$\hat{\square}$	Statistical estimator of the quantity $\square$
$\tilde{\square}$	Denotes self-normalized importance sampling (SN) for weights and estimators

Table 1.2: **General Notations - Indexing and Superscripts.** Summary of important notations and formalism used throughout this dissertation, for indexing and superscripts.

Notation	Short Description
POWSS & Particle Belief MDPs (Chapters 5 and 6)	
C	Number of state/observation samples
$\mathbf{P}$	Denotes the given POMDP, or quantities related to it
$\mathbf{M_P}$	Denotes the corresponding particle belief MDP (PB-MDP) of a given POMDP, or quantities related to it
Voronoi Progressive Widening (Chapter 7)	
C_s	Number of state/observation samples, equivalent to C above. The distinction is necessary since actions also need to be finitely sampled for POMDPs with continuous action spaces
C_a	Number of action samples
$\mathcal{R}_{C_a}$	VOO agent regret bound using C_a queries
p	POWSS or Sparse Sampling bound tail probability
ω	VOO exploration probability
$D(\cdot, \cdot)$	Distance metric defined over the action space
$(k_a, \alpha_a, k_o, \alpha_o)$	Double Progressive Widening parameters for action a and observation o
Visual Tree Search (Chapter 9)	
T_ψ	Transition model of POMDP, neural net with parameters ψ
Z_θ	Observation density model of POMDP, neural net with parameters θ
P_ϕ	Particle proposer model of POMDP, neural net with parameters ϕ
G_ξ	Observation generative model of POMDP, neural net with parameters ξ
ζ	Neural network training hyperparameters
χ	Differentiable Particle Filter hyperparameters
ρ	Monte Carlo Tree Search hyperparameters
Transporter Networks (Chapter 10)	
$\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{pick}}$	SE(2) geometric pose coordinates for picking & placing actions
$\Phi_{\text{key}}, \Phi_{\text{query}}, \Phi_{\text{goal}}$	Transporter network visual input modules for key, query, and goal models
$\psi_{\text{key}}, \psi_{\text{query}}$	Transporter network intermediary modules for key and query models, with the goal information included.
$\mathcal{D}$	Data set that comprises of episode data ζ_i
ζ	Episode data $\zeta_i = \{(\mathbf{o}_1, \mathbf{a}_1), \cdots, (\mathbf{o}_{T_i}, \mathbf{a}_{T_i})\}$ of length T_i
Ecological Navigation (Chapter 12)	
X	State of the community of size n
E	Environmental variable of cardinality m
ξ	Fixed point of the community dynamics
A	A matrix of the GLV model, representing intra- & inter-species interactions
r	r vector of the GLV model, representing intra-species interactions
Δ	Action sequence of low-cost actions $\Delta = \{\delta_1, \delta_2, \dots\}$
ω	Cost sequence of low-cost actions $\omega = \{C_{q,1}, C_{q,2}, \dots\}$
$C_{q,i}$	Cost of action type q at time index i

Table 1.3: **Chapter Specific Notations.** Summary of important notations and formalism used mostly in specific parts of this dissertation.

Chapter 2

Sequential Decision Making under Uncertainty

Chapter Outline

In this chapter, we give an overview of POMDPs and notable solution techniques. We formulate POMDP as an optimization problem, cover conventional techniques used to solve POMDP problems, in particular the MCTS technique, introduce more modern approaches based on deep learning, and discuss the progress in theoretical analyses of these algorithms in the form of convergence guarantees. Some discussions, results, and relevant figures are borrowed & repeated from the original works listed in Parts II and III for better storytelling and emphasis.

2.1 Optimization Objective

This thesis focuses on **sequential decision making under uncertainty**. In this setting, the agent is tasked to optimize some desired *objective*, which often comprises cumulative sum of rewards and costs for a given environment. Mathematically, this problem can be represented as

$$\max_{\pi} J(\pi) \tag{2.1}$$

where π represents a *policy*, which delineates what action an agent should take depending on the inputs, for some objective function J that takes into account the rewards and costs information. Often, the system that the agent plans in may have different forms of uncertainty. In this case, the optimization problem can be expressed in the expected value formulation

$$\max_{\pi} \mathbb{E}[J(\pi)] \tag{2.2}$$

that optimizes the objective function *on average*. There can be other objective formulations, which can optimize over the worst case behavior (min instead of $\mathbb{E}$), and using some notion of risk to interpolate between the minimum and expectation to strike a balance between average behavior and worst case behavior [35, 38]. Specifically in this work, we often assume the system uncertainties exist in the form of transition uncertainty and observation uncertainty, which will be defined in Sections 2.2 and 2.3 and studied further in Parts II and III.

Maintaining safety and acting efficiently in the midst of uncertainty is an important aspect in a diverse set of challenges from transportation [77, 172] to autonomous scientific exploration [25, 58], to healthcare [13] and ecology [123]. There are multiple ways of representing such safety requirements in the optimization framework. One way is to optimize a single objective J_0 while satisfying constraints $J_1, \ldots, J_n$:

$$\max_{\pi} J_0(\pi) \tag{2.3}$$

$$\text{s.t. } J_1(\pi) \geq D_1 \tag{2.4}$$

$$\vdots \tag{2.5}$$

$$J_n(\pi) \geq D_n. \tag{2.6}$$

While this formulation ensures that the constraints are met, many off-the-shelf decision making algorithms are not equipped to handle such constrained formulations, and the thresholds must be chosen a priori which may require additional domain knowledge or more development time.

Another formulation is similar to Lagrange multiplier idea, which incorporates the constraints or other objective functions directly into the optimization formalism by using a weighted sum:

$$\max_{\pi} J_0(\pi) + \lambda_1 J_1(\pi) + \cdots + \lambda_n J_n(\pi). \tag{2.7}$$

This formulation alleviates the aforementioned problems by implicitly dealing with the constraint objectives, but the relative weights must also be chosen beforehand which adds complexity.

2.2 Markov Decision Processes (MDPs)

The Markov decision process (MDP) is a mathematical formalism that can represent a wide range of sequential decision making problems [96]. In an MDP, an agent chooses actions based on current state to maximize the expectation of a cumulative reward signal. An important property of MDPs is the *Markov state* assumption, where the state information at the next time step depends only on the current state and not on any previous states:

$$\mathbb{P}(s_{t+1}|s_t, s_{t-1}, \ldots, s_0) = \mathbb{P}(s_{t+1}|s_t). \tag{2.8}$$

Formally an MDP is defined by the 5-tuple $(S, A, \mathcal{T}, R, \gamma)$. In this tuple, S is the set of all possible states and A is the sets of all possible actions. These sets can be discrete, e.g. $\{1, 2\}$, continuous, e.g. $\mathbb{R}^2$, or hybrid. The transition probability distribution is conditioned on the current state s and action a and is denoted $\mathcal{T}(s' \mid s, a)$. The reward function, $R(s, a)$, maps states and actions to an expected reward, and $\gamma \in [0, 1)$ is a discount factor. The agent plans starting from s_0, the initial state. Some MDP algorithms only require samples from the transition or reward models rather than explicit knowledge of $\mathcal{T}$ or R. Such samples can be produced using a so-called *generative model* [92] denoted with $s', r \leftarrow G(s, a)$.

The objective of an MDP is to find an optimal policy, π^*, that selects actions that maximizes the discounted sum of future rewards, with an appropriate tie-breaking method:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]. \tag{2.9}$$

In order to maximize this objective, the policy must take into account the immediate reward from taking the action in the current state and whether the action will lead to states favorable for attaining rewards in the future. For a state s and action a, the state-action value functions, defined as

$$Q^\pi(s, a) \equiv \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s,\ a_0 = a,\ a_t = \pi(s_t)\right], \tag{2.10}$$

take both of these factors into account. When π is also used for the current step, the expected accumulated reward is denoted with $V^\pi(s) = V^\pi(s) = Q^\pi(s, \pi(s))$. When π is an optimal policy, these value functions are denoted with Q^* and V^*. If Q^* can be calculated, an optimal policy π^* can simply be extracted with $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$.

2.3 Partially Observable Markov Decision Processes (POMDPs)

The partially observable Markov decision process (POMDP) is a mathematical formalism that can represent even wider range of sequential decision making problems than MDPs by also taking into account the *observation uncertainty* [96]. Similar to MDP, in a POMDP, an agent chooses actions based on observations to maximize the expectation of a cumulative reward signal.

A POMDP is defined by the 7-tuple $(S, A, O, \mathcal{T}, \mathcal{Z}, R, \gamma)$. In this tuple, S, A, and O are sets of all possible states, actions, and observations, respectively. These sets can be discrete, e.g. $\{1, 2\}$, continuous, e.g. $\mathbb{R}^2$, or hybrid. The conditional probability distributions $\mathcal{T}$ and $\mathcal{Z}$ define state transitions and observation emissions, respectively. The transition probability distribution is conditioned on the current state s and action a and is denoted $\mathcal{T}(s' \mid s, a)$. The observation probability is conditioned on the previous state and action and current state and denoted $\mathcal{Z}(o \mid s, a, s')$. In our analyses, we often relax this assumption to reason with observation densities that may only be conditioned on next step state s' and action a. The reward function, $R(s, a)$, maps states and actions to an expected reward, and $\gamma \in [0, 1)$ is a discount factor. The agent plans starting from b_0, the initial state distribution or the initial belief. Some POMDP algorithms only require samples from the transition, observation, or reward models rather than explicit knowledge of $\mathcal{T}$, $\mathcal{Z}$, or R. Such samples can be produced using a so-called *generative model* [92] denoted with $s', o, r \leftarrow G(s, a)$.

Similar to MDP, the objective of a POMDP is to find an optimal policy, π^*, that selects actions that maximizes the discounted sum of future rewards, with an appropriate tie-breaking method:

$$\pi^* = \operatorname*{argmax}_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]. \tag{2.11}$$

In general, since the observations are not Markov unlike the states, the actions may be chosen based on the entire history of actions and observations,

$$h_t \equiv (b_0, a_0, o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t). \tag{2.12}$$

However, because of the Markov property of the underlying state distribution, it can actually be shown that optimal decisions can be made based only on the conditional distribution of the state given the history [84], known as the belief,

$$b_t(s) \equiv \mathbb{P}(s_t = s \mid h_t). \tag{2.13}$$

This belief can be updated using Bayes's rule or an efficient approximation such as a Kalman filter or particle filter, and it is often more straightforward to determine actions based on beliefs rather than the history. Since the belief and history fulfill the Markov property, a POMDP is a Markov decision process (MDP) on the belief or history space, commonly referred to as the belief MDP [84]. Chapter 3 provides further details of filtering and estimation techniques for reasoning with beliefs.

The value functions for POMDPs are similarly defined as those for MDPs, except in most cases replacing the state variable with the history or belief variable. For a history h and a corresponding belief b, history-action and belief-action value functions are defined as

$$Q^\pi(h, a) \equiv Q^\pi(b, a) \equiv \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, b_0 = b,\ a_0 = a,\ a_t = \pi(b_t)\right]. \tag{2.14}$$

When π is also used for the current step, the expected accumulated reward from a history or belief is denoted with $V^\pi(h) = V^\pi(b) = Q^\pi(b, \pi(b))$. When π is an optimal policy, these value functions are denoted with Q^* and V^*. If Q^* can be calculated, an optimal policy π^* can simply be extracted with $\pi^*(h) = \operatorname{argmax}_a Q^*(h, a)$.

In our work, we often assume that we solve a POMDP that terminates after D steps. Then, the value functions can be defined as the following for a finite-horizon problem of horizon length D for $t \in [0, D-1]$, where bao indicates the belief b updated with (a, o):

$$V_t^\pi(b) = \mathbb{E}\left[\sum_{i=t}^{D-1} \gamma^{i-t} R(s_i, \pi(s_i)) \middle| b\right], \; V_D^\pi(b) = 0 \tag{2.15}$$

$$Q_t^\pi(b, a) = \mathbb{E}\left[R(s, a) + \gamma V_{t+1}^\pi(bao) | b\right]. \tag{2.16}$$

Accordingly, the optimal value functions for finite-horizon POMDPs satisfy the following:

$$V_t^*(b) = \max_{a \in A} Q_t^*(b, a) \tag{2.17}$$

$$\pi_t^*(b) = \arg\max_{a \in A} Q_t^*(b, a) \tag{2.18}$$

$$Q_t^*(b, a) = \mathbb{E}\left[R(s, a) + \gamma V_{t+1}^*(bao) | b\right]. \tag{2.19}$$

Early research [84, 158, 162] sought to find optimal solutions to POMDPs *offline*; that is, they attempted to optimize actions for every possible belief before interacting with the environment. However, since POMDPs are generally intractable [139], it is often impossible to find a complete solution for a POMDP offline. Thus, a common strategy for solving POMDPs involves iteratively improving estimates of Q^* denoted simply with Q for brevity, and we seek to compute solutions online only for the part of the problem that may be reached in the immediate future.

Another practical consideration or solving POMDPs is that since the agent only receives noisy observations of the state and never the true state itself, the sensing-planning-acting outer loop must have an observer system that also keeps a robust representation of the current belief. Such representations of the belief are described further in detail in Chapter 3. Thus, a typical closed-loop planning procedure for POMDP proceeds as following:

1. *Sensing*: The agent receives the corresponding observation and reward, which is then used to update the next step belief using a belief updater.
2. *Planning*: At the current belief state, plan for the next-step action using a POMDP policy/planner.
3. *Acting*: Perform the action chosen by the planner in the environment.

Fig. 2.1 demonstrates this closed-loop planning in a self-driving car example. In our work, we are mostly concerned with analyzing the *planning* step of the loop, but empirical studies performed in our work implements all three parts of the closed-loop.

2.4 Common Solution Approaches for POMDPs

2.4.1 Value Iteration for POMDPs

Value iteration is a simple offline method for solving MDPs, and can be extended to POMDPs as well. Value iteration for MDPs proceeds by utilizing the Bellman equation to characterize

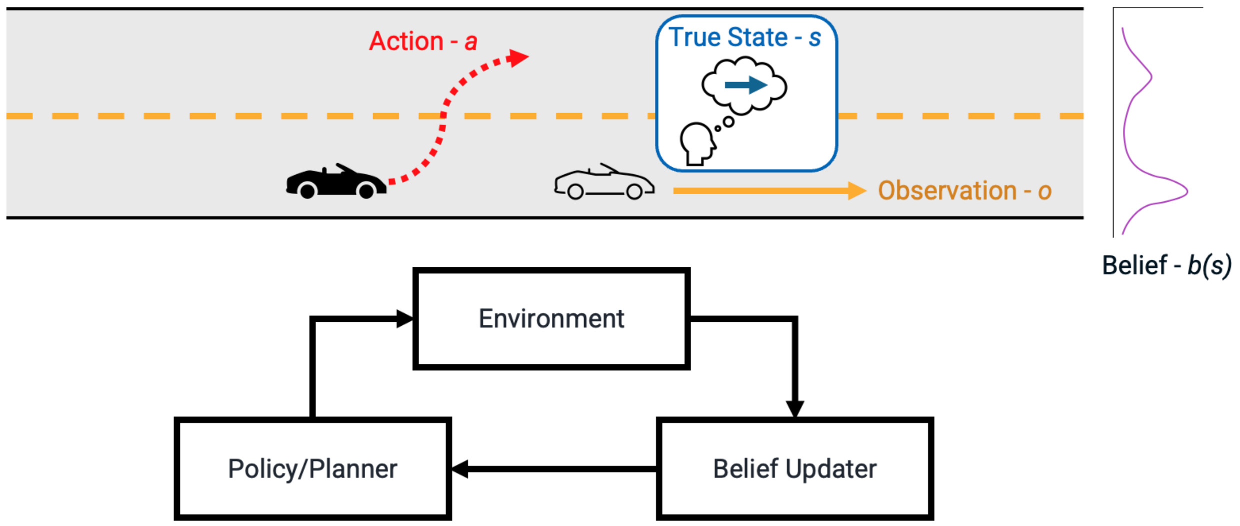


Figure 2.1: **Overview of the sensing-planning-acting procedure for closed-loop POMDP planning in a self-driving example.** The agent receives an observation (front car going forward) from the environment, which is used to update the belief about the true state distribution using a belief updater (sensing step). Then, the agent decides to change the lane using a policy/planner (planning step), and performs the lane change maneuver in the environment (acting step). The icons are generated from Microsoft PowerPoint 2023.

the optimal value function:

$$V^*(s) = \max_{a\in A} \left\{ R(s,a) + \mathbb{E}[V^*(s')|s,a] \right\}. \tag{2.20}$$

The Bellman operator applies this recursive definition:

$$\mathcal{B}[V](s) = \max_{a\in A} \left\{ R(s,a) + \mathbb{E}[V(s')|s,a] \right\}. \tag{2.21}$$

Specifically, for discrete problems, the Bellman operator can be used to find the unique solution to the Bellman equation through iterative application: $V_{k+1}(s) = \mathcal{B}[V_k](s)$. Due to the contraction mapping property of the Bellman operator, the uniqueness of fixed point and the convergence to such point is guaranteed. Such fixed point is the optimal value function, and the iterative process above is known as the value iteration [20]. Once the optimal value function is found, the optimal policy can be extracted by taking the maximizing argument of the Bellman operator:

$$\pi^*(s) = \arg\max_{a\in A} \left\{ R(s,a) + \mathbb{E}[V^*(s')|s,a] \right\}. \tag{2.22}$$

There is an analog of value iteration for POMDPs called α-*vectors* [84]. The main idea of α-vector algorithms is reasoning about belief distributions when performing Bellman-type backups through α-vectors that are linear representations of value functions. The algorithm utilizes the following facts. First, the policy value function of discrete POMDPs are defined as $V_p(b) = \sum_{s \in S} V_p(s)b(s)$, where $V_p(s)$ is the value function obtained by starting at initial state s and executing some policy $p \in P$. This state value function $V_p(s)$ can obtained via the policy tree in discrete settings, which is a tree that outlines all the possible outcomes and corresponding values, and it can be constructed via similar recurrence relationship as the Bellman update for MDPs. Second, the value function will be the supremum of the value functions that form a set of hyperplanes. This leads us to representing the value function as the set of value hyperplanes obtained from policy trees, which are defined to be the α-vectors:

$$\alpha_p(s) = (V_p(s_1), \ldots, V_p(s_1)). \tag{2.23}$$

Consequently, the value function can be obtained by taking the inner product with the belief vector:

$$V(b) = \max_{p \in P} b \cdot \alpha_p. \tag{2.24}$$

2.4.2 Approximate POMDP Solvers

Finding an optimal POMDP policy is computationally demanding because of the uncertainty introduced by imperfect observations [139]. One of the most popular approaches to deal with this computational challenge is to use *online* algorithms that look for local approximate policies as the agent interacts with the environment rather than computing a global policy that maps every possible outcome to an action.

One common method for solving POMDPs online is online tree search, which is attractive for several reasons. First, the approach scales to very large problems because it uses sampled trajectories, making it insensitive to the dimensionality of the state and observation spaces [92]. Second, since online computation focuses on the current states and states likely to be encountered in the future, it can reduce the need for offline computation and end-to-end training [48]. Third, tree search is applicable to a wide range of problems, for example hybrid continuous-discrete and non-convex problems, because it only depends on a minimal set of problem structure requirements. As such, many online POMDP algorithms are variants of Monte Carlo tree search (MCTS) [28, 159, 171] or other tree search variants [101, 191].

2.4.3 Monte Carlo Tree Search (MCTS)

Monte Carlo tree search (MCTS) is a common solution technique for Games, MDPs and POMDPs [28, 159]. In an MDP context, MCTS constructs a tree consisting of state and state-action nodes. In a POMDP context, each node corresponds to an action- or observation-terminated history node, estimating $Q(h, a)$ at each action-terminated history node. The most common variant is called partially observable upper confidence trees (PO-

UCT) or partially observable Monte Carlo planning (POMCP)[1] [159] and repeats the following four steps to construct the tree:

1. *Search.* A simulation proceeds from the root node with action selected according to the Upper Confidence Bound (UCB) exploration criterion,

$$UCB(h,a) = Q(h,a) + c_{UCB}\sqrt{\frac{\log(N(h))}{N(h,a)}}, \tag{2.25}$$

   where $N(h)$ and $N(h,a)$ are the number of times a history and action node have been visited, respectively. The first term encourages tree expansion towards parts of the history space that are likely to have high value, and the second term encourages exploration in parts of the space that have not been visited as often. The hyperparameter c_{UCB}, commonly referred to as the UCB critical factor, balances these two considerations. Once an action has been chosen, the next state, reward, and observation are generated by sampling from the POMDP stochastic model.

2. *Expansion.* Eventually, the simulation reaches a leaf node and it is expanded by creating new action nodes.

3. *Estimation.* After the expansion step, the value at the history node is estimated, either with a domain-specific heuristic, or with a rollout simulation.

4. *Update.* The estimates of $Q(h,a)$ at each of the visited action nodes are updated with the discounted reward and backed up value estimates from descendant nodes.

This approach builds a tree asymmetrically favoring regions of the history and action spaces that are likely to be visited when the optimal policy is executed.

In addition to the PO-UCT algorithm described above, several other approaches round out the online POMDP solver landscape. Early solvers attempted to use exact Bayesian belief updates on discrete state spaces [148], however, these are much less scalable than PO-UCT. Two other popular solvers with scalability similar to UCT are determinized sparse partially observable trees (DESPOT) [191] and adaptive belief trees (ABT) [101]. DESPOT uses a small number of determinized scenarios instead of independent random simulations to reduce variance and relies on heuristic tree search guided by upper and lower bounds rather than Monte Carlo tree search. ABT is designed to efficiently adapt to changes in the environment without throwing away previous computation.

2.4.4 QMDP

One notable approximation scheme of POMDPs is the QMDP approximation [84]. The QMDP approximation proceeds by solving the fully observable MDP problem of the POMDP,

[1]Strictly speaking, POMCP also includes a specialized unweighted particle filter update, but the term is often used informally as a synonym for PO-UCT.

by assuming no observation uncertainty starting from a given state, and obtaining its optimal state-action value function Q_{MDP}. This value function can be relatively easily calculated using value iteration or approximate MDP solvers, which are usually much less computationally demanding. Then, the QMDP value approximation is defined as the following:

$$Q_{\mathrm{MDP}}(b, a) = \mathbb{E}[Q_{\mathrm{MDP}}(s, a)|b]. \tag{2.26}$$

In essence, the QMDP approximation assumes that the observation uncertainty disappears after one action step due to taking the naive expectation over the fully observable MDP Q_{MDP} state-action value. Unfortunately, the QMDP approximation is known to be suboptimal in some contexts, as it assumes no observation uncertainty after one step and becomes overconfident as it is not incentivized to learn more about its current state.

Despite the suboptimality, QMDP approximation proves to be not only useful in many contexts due to its simplicity, but also can be used as baseline planner algorithm to be used as a rollout estimator. In this thesis, many experiments leverage the QMDP approximation as the rollout policy of tree search solvers in order to obtain a good estimate of the value function at leaf nodes.

2.4.5 Deep Reinforcement Learning

Many deep reinforcement learning algorithms are implicitly solving a POMDP, in a sense that the agent often needs to reason about complex state or observation inputs for a Markov environment, and must come up with an action or a policy by learning a neural network representation. While we will not dive too much into the details of all deep reinforcement learning methodologies, we outline the four major types of deep reinforcement learning solvers. These four characterizations are not definitive, and some modern algorithms integrate multiple of these components for better performance.

1. **Policy gradients**: Policy gradient methods directly differentiate the expected sum of rewards objective to obtain the policy. Some examples include REINFORCE [186], natural policy gradient [142], trust region policy optimization (TRPO) [151], and proximal policy optimization (PPO) [150].

2. **Value-based**: Value-based methods learn a value function network or a Q-function network that represents the optimal values, and use the value maximizing action as the policy. Some examples include Q-learning [184], deep Q-network (DQN) [131], temporal difference (TD) learning [173], and fitted Q-value iteration [145].

3. **Actor-critic**: Actor-critic algorithms learn a value function network or a Q-function network that represents the current policy values, and use that to improve the policy. Some examples include asynchronous advantage actor-critic (A3C) [130] and soft actor-critic (SAC) [68].

4. **Model-based**: Model-based algorithms estimate the model components of POMDP, usually the transition model, and then use it for planning, improve a policy, or perform other tasks to obtain the action. Most of our work in Part III and the related works discussed in those sections would fall under the model-based reinforcement learning category.

2.4.6 Compositional Learning

In compositional learning, a learning task is broken down into neural network components that each specialize in different tasks. These components are then integrated to learn complex relations, allowing for less data and training resources to be used overall. Compositional learning can be achieved either through provided compositional structure in the form of an algorithmic prior [6, 78], or by automatically discovering such structures [4, 95, 127, 146]. Specifically, algorithmic prior knowledge can be introduced by specifying neural network architectures or model components, which then allows for intelligent planning with minimal training from limited or specialized data. Furthermore, this paradigm has enjoyed success in various sequential decision making under uncertainty problems, and particularly reinforcement learning settings [48, 51, 67, 111, 129, 190]. Part III further discusses backgrounds for how these compositional learning methods can be used, and introduces our two new contributions that utilize compositional learning.

2.4.6.1 Vision POMDPs

Recently, there has been increased interest in solving POMDPs involving visual observations through deep learning [65, 88, 201], which can be viewed as instances of compositional learning. Model-free vision POMDP solvers such as QMDP-net [89] and Deep Variational Reinforcement Learning [80] maintain a latent belief vector whose update rule is learned via a neural network. They then learn corresponding value or policy networks in this latent belief state and action space. Furthermore, since these methods usually make minimal sets of assumptions, extensions of such techniques can also be seen in embodied artificial intelligence applications [3, 87]

Model-based vision POMDP works such as Differentiable Particle Filter (DPF) [83] allow conventional particle filtering techniques to be interfaced with complex visual observations. This algorithm contains multiple linked neural network components, including a particle proposer, an observation model, and a dynamics model, that are designed to be trained end-to-end. A recent work extends DPF with entropy regularization and provides convergence guarantees [41]. Dual Sequential Monte Carlo (DualSMC) [183] extends the DPF methods further to introduce an adversarial filtering objective and integrate in the SMC planner, making it a fully closed-loop POMDP solver. The DPF framework is extensively discussed in Section 3.2.3.

In Chapter 9, we break down the vision POMDP problem into a compositional learning problem involving learning *deep generative models*. Many models in deep learning are used to approximate probability distributions over high dimensional spaces such as spaces of images and videos. However, sampling from these distributions requires deep generative models. While some deep generative models, such as Generative Adversarial Networks (GANs), try to sample from distributions by making the samples seem as "realistic" as possible, other models such as Variational Autoencoders (VAEs) map complex distributions to simpler ones via latent space embedding. Conditional generative models condition on another variable to sample from conditional distributions [128, 202]. In our MCTS planner introduced in Chapter 9, we use deep generative models G to sample observations conditioned on the state, and P to propose state particles given the current observation.

2.4.6.2 Pick-and-Place Tasks

Recent works in pick-and-place tasks leverage learned models to generate pick-and-place policies that can successfully generalize to unseen objects [19, 52, 64, 79, 93, 154, 166, 189, 194, 199]. These vision-based agents produce per-pixel scores in an image, which are then mapped to physical pick-and-place coordinates. This approach is shown to be successful for tasks including robotic warehouse order picking [197], pushing and grasping [196], tossing objects [198], kit assembly [194], and mobile manipulation [188].

2.4.6.3 Multi-Task Learning

Multi-task learning approaches aim to improve learning efficiency through shared structure and data between tasks. While multi-task learning and systems have shown effectiveness across several domains [79], they often require careful choices of tasks, balanced sampling, and learning curricula [47, 66, 165]. Consequently, multi-task learning may not always improve learning, sometimes diminishing performances as shown in vision-based continuous control manipulation tasks [178]. Moreover, composing tasks together amplifies the problem of compounding errors [103, 137, 200] and the need for large number of demonstrations [55, 147]. Understanding task structure may be a key ingredient in scaling reinforcement learning methods to compositional tasks [108, 136, 164, 165], as leveraging task structures can successfully solve realistic compositional tasks including web navigation [66] and subtask dependency inference [165].

Sequence-conditioning combined with multi-task learning can also be viewed as a variant of one-shot meta-learning. Meta-learning approaches aim to learn policies that can quickly adapt and generalize to new task instances [55], and some of these approaches also condition the policies on demonstration videos or sequences [47, 193]. While most prior works in meta-learning focus on generalizing to new unseen tasks of short planning horizon, our work focuses on multiple seen tasks that result in long planning horizon.

2.5 Continuous Space POMDP Algorithms

2.5.1 Online Planning for Continuous Space Problems

Dealing with varying levels of continuous space assumptions make POMDP problems much more intangible to solve analytically. Here, we denote *continuous observation POMDPs* as POMDPs with continuous observation (and also often continuous state) space, and *continuous space POMDPs* as POMDPs with continuous state, action, and observation spaces.

2.5.2 Online Continuous Observation POMDP Solvers

There are many approaches for solving POMDPs or belief-space MDPs with continuous observation spaces. For example, a variant of Monte Carlo Value Iteration (MCVI) uses Monte Carlo sampling-based Bellman backup and leverages a classifier to deal with continuous observation spaces and provide convergence guarantees [14]. Others partition the observation space [75] or assume that the most likely observation is always received [143]. Many approaches are based on motion planning [2, 29], locally optimizing pre-computed trajectories [179], or optimizing open-loop plans [170]. McAllester and Singh [121] also extend the sparse sampling algorithm of Kearns *et al.* [92], but they use a belief simplification scheme instead of the particle sampling scheme.

Previous online approaches that use particle belief representations, such as ABT [101], POMCP [159], and DESPOT [191] have exhibited good performance in large discrete domains. However, many real-world domains, notably when a robot interacts with the physical world, have continuous observation spaces, and the algorithms mentioned above will not always converge to an optimal policy in problems with continuous or naively discretized observation spaces [171]. Since PO-UCT, DESPOT, and ABT all rely on an unweighted particle representation, they can be used with continuous state spaces. However, they will fail to find optimal policies in continuous observation spaces since the probability of generating the same observation twice, and hence creating beliefs with multiple particles, is zero [109, 171].

Partially observable Monte Carlo planning with observation widening (POMCPOW), approached the continuous observation challenge by introducing a weighted particle filter and the continuous action challenge with progressive widening [171]. DESPOT-α [61] incorporates a similar weighting scheme and uses the α-vector concept to generalize value estimates between sibling nodes. Adaptive online packing-guided search (AdaOPS) [187] fuses similar observation branches in the search tree to improve performance. Lazy Belief Extraction for Continuous Observation POMDPs (LABECOP) [74] builds the planning tree by re-weighting beliefs with particle weighting and extracting belief sequence values efficiently.

2.5.3 Online Continuous Space POMDP Solvers

There are also solvers that additionally focus on dealing with continuous or hybrid action spaces. Several techniques use double progressive widening (DPW) [43], originally designed to solve continuous space MDPs. Most notably, POMCPOW and PFT-DPW [171] extend POMCP and DPW to handle continuous space POMDPs. However, these algorithms use DPW as it is proposed with the inefficient action sampling that is not suitable for large problems.

One effective direction to handle continuous action spaces has been to use continuous bandits to sample new actions. Particularly, hierarchical optimistic optimization (HOO) and the corresponding HOOT algorithm [117] began work on continuous bandit algorithms applied to MCTS. This is followed by works such as HOLOP [185] that plans with open-loop trajectories instead of individual actions using HOO, POLY-HOOT [118] with polynomial guarantees, and VOO and VOOT [94] that we build on in Chapter 7.

Another direction is to use Bayesian optimization to efficiently sample new actions. CBTS [133] uses Gaussian processes (GP) to tackle the random action sampling problem. However, CBTS does not use progressive widening approach and uses a separate GP at each belief node, which limits its optimization scope and branching capabilities. Most recently, BOMCP [124] extends POMCPOW by posing the random action sampling step of DPW as a Bayesian optimization problem over all the belief and action nodes. Despite the effectiveness of BOMCP, GPs are very computationally expensive to fit especially over the joint belief and action space, and BOMCP trades off computation time for sample efficiency compared to POMCPOW.

Other directions include GPS-ABT [152] that uses generalized pattern search to find local optima, PA-POMCPOW that additionally incorporates score functions [125], and VG-UCT that calculates gradient of the value function [104].

2.5.4 Theoretical Guarantees

Recently, there has been a considerable amount of developments in providing guarantees for online POMDP algorithms. Some of the notable popular algorithms for large discrete POMDPs include POMCP [159] and DESPOT [191], which were foundational algorithms for both theoretical analysis and practical algorithm development. A lot of contemporary algorithms take inspiration from the belief representation using state particle insertions. However, these algorithms cannot effectively handle continuous state and observation spaces without relying on some discretization schemes, due to the fact that the planning trees become infinitely wide of depth 1, commonly referred to as bushes or shrubs. This phenomena is mainly due to the fact that in extremely large or continuous probability measures, the probability of sampling the same observation twice is zero [109, 171]. Since the state particle insertion scheme relies on revisiting the same observation many times and inserting state particles into it, it ends up generating depth 1 bushes as it is unable to visit an observation node more than once and cannot expand depth-wise. This results in a myopic planner that never reasons past the first step and cannot deal with long horizons.

Recently proposed POMDP tree search algorithms [14, 61, 74, 109, 110, 124, 171, 187] have been shown empirically to work on continuous state and observation spaces, and some come with finite sample performance guarantees. Specifically, several of these algorithms such as POMCPOW and PFT-DPW [171] and DESPOT-α [61] have employed a weighting scheme inspired by particle filtering to achieve good performance on realistic problems with large or continuous observation spaces. In our work, we discuss the POWSS algorithm [109] in Chapter 5, which not only formally justifies the necessity of such weighting scheme but also provides and algorithm with convergence guarantees that does not directly depend on the observation space size. Furthermore, the VPW algorithm [110] in Chapter 7 captures both theoretical guarantees and empirical performance when we additionally assume continuous or hybrid action space.

While there are many particle weighting-based algorithms that have performance guarantees [109, 110] and algorithms that perform well empirically [61, 74, 110, 124, 171, 187], there has been little progress on generalization of why such family of algorithms can enjoy performance guarantees. For instance, AdaOPS [187], a recent particle belief-based POMDP solver that we include in our analysis, comes very close to capturing both theory and practice, but it has a complex algorithmic structure and provides only algorithm specific guarantees with additional simplifying assumptions.

In our work, we discuss the particle belief approximation of POMDPs [112] in Chapter 6, which formally justifies that optimality guarantees in a finite sample particle belief MDP (PB-MDP) approximation of a POMDP/belief MDP yields optimality guarantees in the original POMDP as well. Notably, this convergence rate does not directly depend on the size of the state space nor the observation space, but rather depends on a statistical quantity that links the probabilities concerning state and observation trajectories and the planning horizon D. This fundamental bridge between PB-MDPs and POMDPs allows us to adapt any sampling-based MDP algorithm of choice to a POMDP by solving the corresponding particle belief MDP approximation and preserve the convergence guarantees in the POMDP.

Chapter 3

State Estimation and Representation

Chapter Outline

In this chapter, we give an overview of state estimation through belief representations. We cover some common methods of belief representations, dive deeper into particle beliefs and some modern techniques, and discuss some theoretical properties of particle beliefs by introducing importance sampling. Some discussions, results, and relevant figures are borrowed & repeated from the original works listed in Parts II and III for better storytelling and emphasis.

3.1 Belief Representations

When the agent is only provided with noisy or complex observations that reveal partial information about the state of the environment, the agent should take into account the entire observation history in order to plan optimally. This is sometimes known as the *curse of history*, as the agent is burdened with keeping the entire history of observations to intelligently plan over long horizons, as the observation sequence is no longer Markov. Furthermore, modern high dimensional observations such as images, videos, or texts require sophisticated machinery to estimate the state at any given point in time, for which a good latent representation that explicitly or implicitly encodes this history information is crucial.

3.1.1 Reasoning with Beliefs

One effective technique to handle such curse of history is to reason about the observation histories with *beliefs* or belief functions. Generally speaking, beliefs are mathematical objects that capture the state density given an initial density representation and some history of observations and actions:

$$b_t(s) \equiv b_t(s|b_0, \{a_i\}, \{o_i\}). \tag{3.1}$$

Beliefs are sufficient statistics for optimal decision making. This intuitively means that beliefs are the only object necessary to fully capture the observation history information that is relevant for optimal planning. Mathematically, there exists a policy π^* that derives the action policy $a_t = \pi^*(b_t)$, which maximizes the expected cumulative reward or the value function of the POMDP [84, 96]. For an exact analytical update, if the exact density form is analytically known, then it is possible to update the belief function with a new action and observation at a given time via Bayesian update due to the Markov property [84]. These subsequent updates of b_t given (a_t, o_{t+1}) can be calculated as following:

$$b_{t+1}(s_{t+1}) = \frac{\int_S \mathcal{Z}(o|s_t, a_t, s_{t+1})\mathcal{T}(s_{t+1}|s_t, a_t)b_t(s_t)\, ds_t}{\int_S \int_S \mathcal{Z}(o|s_t, a_t, s_{t+1})\mathcal{T}(s_{t+1}|s_t, a_t)b_t(s_t)\, ds_t\, ds_{t+1}}. \tag{3.2}$$

The integrals can be replaced by sums when the state space is discrete. Thus, beliefs are a compact way of keeping track of history information.

3.1.2 Belief MDP

Since the belief and history fulfill the Markov property, a POMDP can be viewed as a Markov decision process (MDP) on the belief or history space, commonly referred to as the belief MDP [84]. The corresponding belief MDP for a given POMDP problem $\mathbf{P} = (S, A, O, \mathcal{T}, \mathcal{Z}, R, \gamma)$ is the MDP $\mathbf{M}_B = (B, A, \mathcal{T}_B, R_B, \gamma)$ defined by the following elements:

- B: Space over belief densities b_d. The belief density is now a "state" in this MDP.
- A: Action space. Remains the same as the original action space.
- $\mathcal{T}_B$: Transition density $\mathcal{T}_B(b_{d+1} \mid b_d, a)$. The transition probability from b_d to b_{d+1} by taking the action a can be defined as:

$$\mathcal{T}_B(b_{d+1} \mid b_d, a) \equiv \int_O \mathbb{P}(b_{d+1} \mid b_d, a, o)\mathbb{P}(o \mid b_d, a)do. \tag{3.3}$$

- R_B: Reward function $R_B(b_d, a)$:

$$R_B(b_d, a) = \mathbb{E}_{s\sim b_d}[R(s, a)]. \tag{3.4}$$

- γ: Discount factor. Remains the same as the original discount factor.

This type of reasoning will prove to be extremely useful for deriving the particle belief MDP approximation for POMDPs in Chapter 6. Particle belief MDP has a very similar definition as the belief MDP definition in order to effectively reason about the MDP structure over belief approximations using particles.

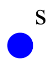
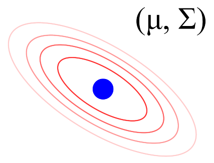
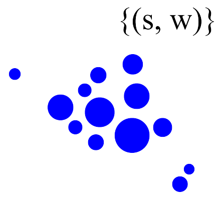
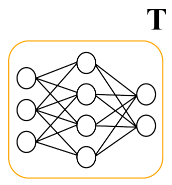
State (Mean)	**Gaussian**	**Particle**	**Tensor**
			
• Simple representation	• Simple representation • Captures uncertainty • Mathematically convenient	• Multi-modal uncertainty • Statistical guarantees • Models still applicable	• Complex uncertainty • Modern inputs • Interface w/ neural nets
• Certainty equivalence • No uncertainty info	• Uncertainty representation simple and uni-modal	• Extra computational factor $\mathcal{O}$(#particles) • Need obs. density	• Need neural net models • No guarantees • Difficulty training & representations

Table 3.1: **Overview of select belief representation methods.** The advantages of each method are listed on the top row, and the disadvantages on the bottom row.

3.1.3 Representations of Belief State

There are many ways to reason about the belief state. We outline some of the popular methods that represent the belief state, and their various properties in Table 3.1.

The simplest representation is using a single state directly as the representation of a belief. This can be done by either just using the observation directly as the state if the observation and state spaces are the same, or extracting the mean of some belief function if the belief function is externally managed, such as in the outer state observer. While this is a simple representation, there is no representation of uncertainty information whatsoever. This effectively assumes certainty equivalence, for which such assumption can be sometimes theoretically shown to be suboptimal [84]. However, the certainty equivalence principle actually holds up quite well in many robotic experimental scenarios, as the sensor noise is small enough.

Another simple representation is using a Gaussian distribution to represent the belief. While this is similar to the single state representation, the Gaussian belief now captures the uncertainty information through the covariance matrix. This additional information is still rather simple to manage, and the updates to the mean and covariance matrix can be done with techniques such as the Kalman filter and its variants. However, this uncertainty representation is still crude and uni-modal, which does not perform well in scenarios that require multi-modal representation of belief density.

The third option is the one we will extensively analyze in Part II, which is representing a belief with a set of particles and their corresponding weights. This representation strikes a balance between representational power that not only captures the multi-modality and the theoretical guarantees of finitely representing a belief density, but also being able to use the known POMDP model components to plan. However, since this uses a large number of particles, this induces an extra multiplicative computational factor of $\mathcal{O}(\#\text{particles})$, and it also needs an analytic model of the observation density function in order to correctly update the state particle likelihood weights via Bayesian updates. Despite the above shortcomings, the mathematical flexibility and plausibility makes it an ideal candidate for theoretical analysis.

The last option listed is the tensor representation of beliefs. This option is both explicitly and implicitly used in many modern reinforcement learning solvers that reason with complex inputs [69, 80, 89]. Since many modern machine learning methods utilize neural networks, which can distill complex information into tensor representation, modern neural network-based planners use tensors to reason about the observations. There is a wide variety of tensor representations: some reinforcement learning algorithms train solely on the observations, which may implicitly learn a belief representation internally in the neural network, while others may explicitly formulate latent space embedding, very similar to a belief representation as it often takes in some form of observation history as an input. However, it is very difficult to learn such high-dimensional embedding, and there is very little statistical guarantee in most real-world scenarios.

3.2 Particle Filtering

The particle filter is an application of Monte Carlo estimation to the task of Bayesian belief updating [97, 174]. In essence, it uses a set of states, and their weights if using a weighted particle filter, to keep track of likely hypotheses of states.

3.2.1 Principles

Unweighted particle filter. The simplest form is an unweighted particle filter, in which the belief is represented by a collection of N states, known as *particles*, $\tilde{b} = \{s_i\}_{i=1}^N$. The density is approximated by $\tilde{b}(s) \approx \sum_{i=1}^N \delta(s_i = s)$, where $\delta(\cdot)$ is a Dirac or Kronecker delta function depending on the form of the state space. At each step of the POMDP, after an action a is taken and an observation o is received, a new state s_i' and observation o_i is simulated once or more for each of the particles to create the new belief, $\tilde{b}' = \{s_i' : o_i = o\}$. In most cases, few particles will match o so it is difficult to maintain a large number of particles in the belief. Various domain-specific techniques can be used to reduce this problem, but it is difficult to solve completely in the unweighted particle filter.

Weighted particle filter. The weighted particle filter is usually much more effective. The belief is represented by a collection of state particles and corresponding weights, $\tilde{b} = \{(s_i, w_i)\}_{i=1}^N$. The density is approximated with $\tilde{b}(s) \approx \frac{\sum_{i=1}^N w_i \delta(s_i = s)}{\sum_{i=1}^N w_i}$. A belief update consists of simulating each particle once or more and then calculating the new weight according to the importance sampling correction: $w_i' = w_i \cdot Z(o \mid s, a, s_i')$. This weight will be referred

Algorithm 3.1 SIR Particle Filter vs. Generative Particle Filter

Algorithm: SIRPF($\bar{b}, a, o$)
Input: particle belief set $\bar{b} = \{(s_i, w_i)\}$, action a, observation o.
Output: New updated particle belief set $\bar{b}' = \{(s'_i, w'_i)\}$.

```
1: for $i = 1, \dots, C$ do
2:     $s_j \leftarrow$ sample $s_i$ from $\bar{b}$ w.p. $\frac{w_i}{\sum_i w_i}$
3:     $s'_i \leftarrow G(s_j, a)$
4:     $w'_i \leftarrow w_i \cdot \mathcal{Z}(o|a, s'_i)$
5: for $i = 1, \dots, C$ do
6:     $w'_i \leftarrow \frac{w'_i}{\sum_i w'_i}$
7: $\bar{b}' \leftarrow \left\{(s'_i, w'_i)\right\}_{i=1}^{C}$
8: return $\bar{b}'$
```

Algorithm: GenPF($\bar{b}, a$)
Input: particle belief set $\bar{b} = \{(s_i, w_i)\}$, action a.
Output: New updated particle belief set $\bar{b}' = \{(s'_i, w'_i)\}$, mean reward ρ.

```
1: $s_o \leftarrow$ sample $s_i$ from $\bar{b}$ w.p. $\frac{w_i}{\sum_i w_i}$
2: $o \leftarrow G(s_o, a)$
3: for $i = 1, \dots, C$ do
4:     $s'_i, r_i \leftarrow G(s_i, a)$
5:     $w'_i \leftarrow w_i \cdot \mathcal{Z}(o|a, s'_i)$
6: $\bar{b}' \leftarrow \left\{(s'_i, w'_i)\right\}_{i=1}^{C}$
7: $\rho \leftarrow \sum_i w_i r_i / \sum_i w_i$
8: return $\bar{b}', \rho$
```

	SIR Particle Filter	**Generative Particle Filter**
Observation Generation	✗	✓
Resampling	✓	✗
Purpose	State estimation	Value estimation
Used in	POMDP outer loop observer	POMDP action planner

Table 3.2: **Comparison of SIR particle filter and generative particle filter.**

to as the particle likelihood weight. Typically, the weights of a few particles grow much larger than the others, so a resampling step creates many particles from those with large weights and eliminates those with very small weights [97, 174]. A basic implementation of this weighted particle filter is given in Algorithm 3.1 – this algorithm is also known as the sequential importance resampling (SIR) particle filter. A more comprehensive survey of particle filtering and corresponding guarantees are available in [45].

3.2.2 Generative Particle Filter

The generative particle filter, which is a logic that is implicitly used a lot in POMDP planning, is very similar to the weighted particle filter mentioned in the previous section. However, the generative particle filter does not take in an observation as an input, but rather *samples* the next step observation. This is mostly to enable sampling from an approximate density of belief dynamics using observation and transition densities for sampling-based online planners. Specifically, the generative particle filter corresponds to the transition model of a particle belief approximation of POMDPs:

$$\tau(\bar{b}_{d+1} \mid \bar{b}_d, a) \equiv \int_O \mathbb{P}(\bar{b}_{d+1} \mid \bar{b}_d, a, o)\mathbb{P}(o \mid \bar{b}_d, a)do. \tag{3.5}$$

This allows treating the particle belief approximation as a form of MDP, which enables us to connect guarantees from MDP-type algorithms to POMDP-type algorithms. This connection is further described in Chapter 6. Thus, the main difference is that the generative particle filter seeks to sample the next possible particle belief using a valid observation and transition density for the purposes of *planning*, while the particle filter used for observers seeks to estimate the current state for the purposes of *localization*.

An example implementation of the generative particle filter is given in Algorithm 3.1, which is actually the exact generative particle filter routine used in Algorithm 6.1 to correctly define the particle belief approximation of POMDP. Since the above implementation of generative particle filter ensures that the state particle trajectories are independently generated between indices, it suffices to know the likelihood of each trajectory to be able to accurately approximate the full POMDP value function. Table 3.2 outlines the key differences between these concepts.

Note that the generative particle filter does not perform resampling. While it is certainly possible to perform resampling, it actually is less desirable to do so within our theoretical analyses. This is because the main purpose of generative particle filter is to be able to accurately approximate the value function at the root node, and adding resampling logic only adds another layer of complexity by entangling the state trajectory sequences together and also adds in another source of uncertainty which only makes the convergence speed worse. Furthermore, the observations are generated directly from one of the state particles, and this consistency ensures that at least one particle will have a high enough weight. On the other hand, since the SIR particle filter needs to address observations from the environment that may not correspond to any of the state particles in the belief, it must use resampling techniques to deal with particle depletion problems where none of the state particles are likely.

3.2.3 Differentiable Particle Filter

While particle filtering can help represent and solve a fairly large family of POMDP problems due to its flexibility and theoretical guarantees, it cannot handle modern problems that involve complex and high-dimensional inputs such as images, videos, and texts. One popular modern framework to handle the challenge is called differentiable particle filters [83]. Differentiable particle filters use learning-enabled components to represent the POMDP models. Specifically, it learns the transition, observation density, and particle proposer networks to interface particle filters with complex observations.

This in turn allows other planning methods to take advantage of the algorithmic structure (or the algorithmic prior) of particle filtering and particle belief representations. For instance,

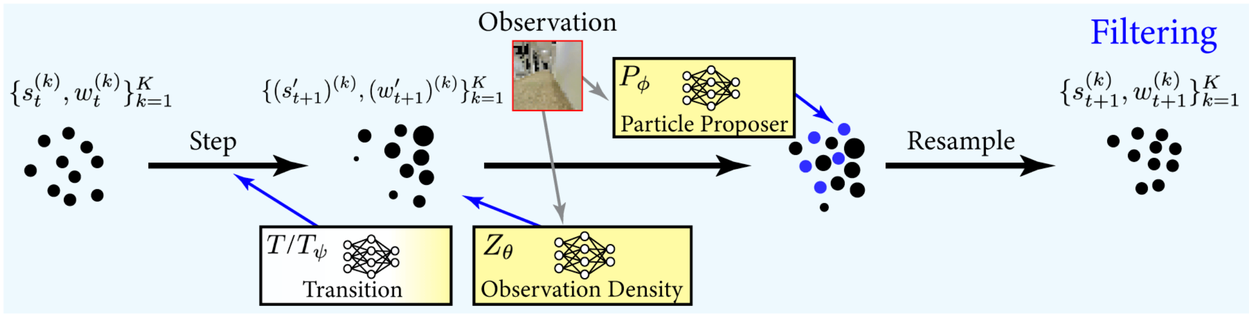

Observation
Filtering
$\{s_t^{(k)}, w_t^{(k)}\}_{k=1}^K$
$\{(s'_{t+1})^{(k)}, (w'_{t+1})^{(k)}\}_{k=1}^K$
$\{s_{t+1}^{(k)}, w_{t+1}^{(k)}\}_{k=1}^K$
Step
Resample
P_ϕ
Particle Proposer
T/T_ψ
Transition
Z_θ
Observation Density

Figure 3.1: **Overview of the differentiable particle filtering system.** The learned modules – transition, observation, and proposer – serve as the model-based representations of POMDP model components. This figure is a modification of Fig. 9.2.

Dual Sequential Monte Carlo (DualSMC) [183] modifies the differentiable particle filter algorithm by adversarially training the observation density and the particle proposer networks from collected data, and employs the Sequential Monte Carlo method as the planner. However, the planner learns state-dependent advantage function neural network, and uses a fixed planner that requires reward information, cannot adapt to different reward structures, needs to interact with the environment, and requires long training that sometimes fails.

Our work Visual Tree Search (VTS) [48] introduced in Chapter 9 interfaces tree search planning methods often used in POMDP literature, by learning an observation generative model neural network instead, which allows us to vastly reduce the training time, does not need to interact with the environment, and can leverage previously collected data without any reward information to be trained offline. This is a type of generative particle filter that uses deep image generative model. This results in an efficient, interpretable, and theoretically principled planner that can take in any reward structure of choice while not compromising on planning quality with short online planning.

Other works in capturing the belief information of the POMDP for high-dimensional inputs often explicitly or implicitly represent the belief in tensor form [69, 80]. The resulting algorithms do not assume any previous knowledge about the state space structure and are very general, but require extremely long training time with little insight into the reasoning behind the decisions they make.

3.3 Importance Sampling

In many real-world applications, updating the belief exactly based on a new action and observation is impractical. Fortunately, Monte Carlo methods provide simple and effective tools for approximate reasoning about distributions such as beliefs. In this section, we focus on the theoretical foundations of particle belief representation, which leverages likelihood weights of each state particles that is theoretically empowered by the tool of importance sampling.

3.3.1 Principles

We often need to reason about a random variable $X \sim \mathcal{P}$ based only on samples from another related random variable, $Y \sim \mathcal{Q}$. *Importance sampling* allows us to, among other tasks, calculate the expectation of a function by observing that

$$\mathbb{E}_{X\sim\mathcal{P}}[f(X)] = \int f(x)\mathcal{P}(x)dx = \int f(x)\frac{\mathcal{P}(x)}{\mathcal{Q}(x)}\mathcal{Q}(x)dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{\mathcal{P}(y_i)}{\mathcal{Q}(y_i)}f(y_i), \tag{3.6}$$

where $\{y_i\}_{i=1}^N$ are samples from distribution $\mathcal{Q}$. The ratio of the densities is known as the importance weights $w_{\mathcal{P}/\mathcal{Q}}(x) \propto \mathcal{P}(x)/\mathcal{Q}(x)$. Consequently, a standard assumption of importance sampling is that the $\mathcal{P}$ is *absolutely continuous* with respect to $\mathcal{Q}$, which means that $\mathcal{P}(x) = 0$ whenever $\mathcal{Q}(x) = 0$.

Since the importance weights are often assumed to be known only up to an unknown normalizing constant, we often work with the self-normalized importance sampling estimator (SN estimator), which additionally normalizes the importance weights with the sum of the available weights. While this produces a slightly biased estimator, it still maintains favorable convergence properties as shown in Theorem 5.1.

We define the following quantities:

$$\tilde{w}_{\mathcal{P}/\mathcal{Q}}(x) \equiv \frac{w_{\mathcal{P}/\mathcal{Q}}(x)}{\sum_{i=1}^{N} w_{\mathcal{P}/\mathcal{Q}}(x_i)} \tag{SN Importance Weight}$$

$$d_\alpha(\mathcal{P}||\mathcal{Q}) \equiv \mathbb{E}_{x\sim\mathcal{Q}}[w_{\mathcal{P}/\mathcal{Q}}(x)^\alpha] \tag{Rényi Divergence}$$

$$\tilde{\mu}_{\mathcal{P}/\mathcal{Q}} \equiv \sum_{i=1}^{N}\tilde{w}_{\mathcal{P}/\mathcal{Q}}(x_i)f(x_i). \tag{SN Estimator}$$

Of particular importance is the infinite Rényi Divergence, d_∞, which can be rewritten as an almost sure bound on the ratio of $\mathcal{P}$ and $\mathcal{Q}$:

$$d_\infty(\mathcal{P}||\mathcal{Q}) = \operatorname*{ess\,sup}_{x\sim\mathcal{Q}} w_{\mathcal{P}/\mathcal{Q}}(x). \tag{3.7}$$

3.3.2 Relationship to Particle Beliefs

Importance sampling is the backbone of particle belief representations, since the particle weights essentially are the importance weights of each state trajectories. In POMDP planning algorithms using particle belief representations, we wish to estimate the value function for beliefs conditioned on observation sequences while only being able to sample from the marginal distribution of states for a given action sequence. Following the formalism of importance sampling, $\mathcal{P}^d$ is the normalized measure incorporating the probability of observation sequence $\{o_{n,j}\}_{n=1}^d$ on top of the state sequence $\{s_{n,i}\}_{n=1}^d$ and action sequence $\{a_n\}_{n=0}^{d-1}$ until the node at depth d, and $\mathcal{Q}^d$ is the measure of the state sequence. We can think of $\mathcal{P}^d$ corresponding to the observation sequence $\{o_{n,j}\}$. For simplicity, we also denote $\mathcal{Z}_{1:d}$ as the product of observation likelihoods $\prod_{n=1}^{d} \mathcal{Z}(o_n \mid a_{n-1}, s_n)$ and $\mathcal{T}_{1:d}$ as the product of transition densities $\prod_{n=1}^{d} \mathcal{T}(s_n \mid s_{n-1}, a_{n-1})$. Then, for an arbitrary action sequence $\{a_n\}$, the following describes the densities necessary to define importance weighting:

$$\mathcal{P}^d = \mathcal{P}^d_{\{a_n, o_{n,j}\}}(\{s_{n,i}\}) = \frac{(\mathcal{Z}^{i,j}_{1:d})(\mathcal{T}^i_{1:d})b^i_0}{\int_{S^{d+1}}(\mathcal{Z}^j_{1:d})(\mathcal{T}_{1:d})b_0 ds_{0:d}} \tag{3.8}$$

$$\mathcal{Q}^d = \mathcal{Q}^d_{\{a_n\}}(\{s_{n,i}\}) = (\mathcal{T}^i_{1:d})b^i_0 \tag{3.9}$$

$$w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\}) = \frac{(\mathcal{Z}^{i,j}_{1:d})}{\int_{S^{d+1}}(\mathcal{Z}^j_{1:d})(\mathcal{T}_{1:d})b_0 ds_{0:d}}. \tag{3.10}$$

Here, the integral to calculate the normalizing constant is taken over S^{d+1}, the Cartesian product of the state space S over $d+1$ steps. Now, we claim that the recursive likelihood updating scheme produces valid likelihood weights up to a normalization. With our recursive definition of the empirical weights, we obtain the full weight of each state sequence for a fixed observation sequence:

$$w_{d,i} = w_{d-1,i} \cdot \mathcal{Z}(o_{d,j} \mid a_d, s_{d,i}) = \ldots = \mathcal{Z}^{i,j}_{1:d} \propto w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\}). \tag{3.11}$$

Therefore, this allows us to justify that the self-normalized importance sampling estimator that uses i.i.d. state particle trajectories and properly updated particle likelihood weights are SN estimators of the expected value of a desired function:

$$\tilde{\mu}_{\bar{b}_d}[f] = \frac{\sum_{i=1}^{C} w_{d,i} \cdot f(s_{d,i})}{\sum_{i=1}^{C} w_{d,i}} = \frac{\sum_{i=1}^{C} w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\}) \cdot f(s_{d,i})}{\sum_{i=1}^{C} w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\})} = \sum_{i=1}^{C} \tilde{w}_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\}) \cdot f(s_{d,i}). \tag{3.12}$$

3.3.3 Convergence Property

Assuming $d_\infty(\mathcal{P}||\mathcal{Q})$ is finite, we can prove an estimator concentration bound in Theorem 5.1, which builds upon the derivation in Proposition D.3 of [126].

Theorem 5.1 (SN d_∞-Concentration Bound)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be two probability measures on the measurable space $(\mathcal{X}, \mathcal{F})$ with $\mathcal{P}$ absolutely continuous w.r.t. $\mathcal{Q}$ and $d_\infty(\mathcal{P}||\mathcal{Q}) < +\infty$. Let $x_1, \ldots, x_N$ be independent identically distributed random variables (i.i.d.r.v.) with distribution $\mathcal{Q}$, and $f : \mathcal{X} \to \mathbb{R}$ be a bounded function ($\|f\|_\infty < +\infty$). Then, for any $\lambda > 0$ and N large enough such that $\lambda > \|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})/\sqrt{N}$, the following bound holds with probability at least $1 - 3\exp(-N \cdot t^2(\lambda, N))$:*

$$|\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}}| \leq \lambda, \tag{3.13}$$

$$t(\lambda, N) \equiv \frac{\lambda}{\|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})} - \frac{1}{\sqrt{N}}. \tag{3.14}$$

This result is important, since it then allows us to establish a general result about function estimation using state particles with likelihood weights, as a precursor to Theorem 6.2. This is useful because the inductive proof for showing Sparse Sampling-ω convergence in Lemma 6.2 relies heavily upon this SN estimator concentration inequality, where estimating the reward/value function at a given particle belief node can be shown to be nearly accurate through this lemma.

Lemma 6.2 (Particle Likelihood SN Estimator Convergence)**.** *Suppose a function f is bounded by a finite constant $\|f\|_\infty \leq f_{\max}$, and a particle belief state $\bar{b}_d = \{(s_{d,i}, w_{d,i})\}_{i=1}^C$ at depth d represents b_d with particle likelihood weighting that is recursively updated as $w_{d,i} = w_{d-1,i} \cdot \mathcal{Z}(o_d \mid a, s_d)$. Then, for all $d = 0, \dots, D-1$, the following weighted average is the SN estimator of f under the belief b_d corresponding to the actions $\{a_n\}_{n=0}^{d-1}$ and observations $\{o_n\}_{n=1}^{d}$, for all beliefs $b_d \in B$ that are realizable given the initial belief b_0:*

$$\tilde{\mu}_{\bar{b}_d}[f] = \frac{\sum_{i=1}^C w_{d,i} f(s_{d,i})}{\sum_{i=1}^C w_{d,i}}, \tag{3.15}$$

and the following concentration bound holds with probability at least $1-3\exp(-C \cdot t_{\max}^2(\lambda, C))$,

$$|\mathbb{E}_{s\sim b_d}[f(s)] - \tilde{\mu}_{\bar{b}_d}[f]| \leq \lambda, \tag{3.16}$$

$$t_{\max}(\lambda, C) \equiv \frac{\lambda}{f_{\max} d_\infty^{\max}} - \frac{1}{\sqrt{C}}. \tag{3.17}$$

Part II

Theory: Fast and Optimal Online Algorithms for POMDPs

Chapter 4

Overview

Chapter Outline

In this chapter, we give an overview of all the works discussed in Part II. We introduce sparse sampling algorithm variants for POMDPs, which constitute the theoretical contributions for continuous observation POMDP algorithms. Then, we discuss the particle belief approximation of POMDPs, which provides a theoretically valid way of extending MDP planning algorithms into POMDP algorithms. Lastly, we summarize our contributions for theoretical and algorithmic developments on solving continuous space POMDPs that additionally deal with continuous or hybrid action spaces. Some discussions, results, and relevant figures are borrowed & repeated from the original works listed in each chapter.

4.1 Sparse Sampling for POMDPs

4.1.1 Sparse Sampling for MDPs

Sparse sampling algorithm [92] is a continuous state MDP algorithm that fully expands the decision tree by recursively exploring all the actions and sampling a fixed number of states C by using the generative model G. The sparse sampling logic of limiting the visit of number of states and fully expanding the sampled subtree provides a powerful analytic tool to reason about convergence guarantees when dealing with continuous states and observations.

The contribution of this algorithm at the time was that there exists a simple yet elegant tree search algorithm that has convergence guarantees to the optimal policy. The convergence guarantee is independent of the state space dimensions while scaling with the number of states sampled and the problem horizon instead, shifting the planning requirement paradigm. This gives an exponentially converging worst-case probability while requiring $\mathcal{O}((|A|C)^D)$ samples, including the contribution from the size of the action space. Even though the sparse sampling algorithm is meant to be a theoretical tool to justify some of the sampling

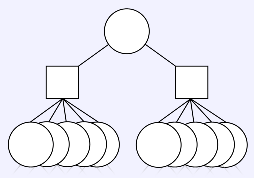
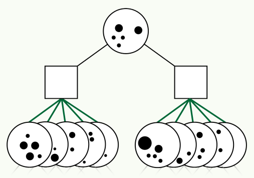
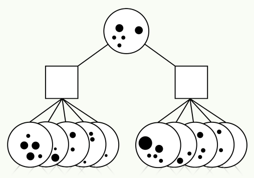
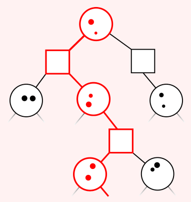
Sparse Sampling	POWSS	Sparse Sampling-ω	Sparse-PFT
			
• MDP • Theoretical • Convergence for cont. state MDPs • Foundational algorithm	• POMDP • Theoretical • Convergence for cont. obs. POMDPs • State particle insertion • Weighted sum	• POMDP • Theoretical • Bridge between PB-MDPs & POMDPs • Particle belief transition • Monte Carlo sum	• POMDP • Practical • Sparse-UCT adapted to PB-MDPs

Table 4.1: **Summary of Sparse Sampling related algorithms discussed in this dissertation.** These are designed to tackle continuous state MDPs or continuous observation POMDPs.

based solvers to tackle the challenging continuous state MDPs, the relative ease of algorithm definition and the concise mathematical guarantees that it enjoys makes it an ideal candidate to further take inspiration from to analyze POMDP planning algorithms.

In this dissertation, we actively leverage the concept of sparse sampling to now generalize our results to continuous observation POMDPs, which additionally takes into account the observation uncertainty in POMDPs. This leads to our two sparse sampling-based algorithms for continuous observation POMDPs: the Partially Observable Weighted Sparse Sampling (POWSS) algorithm and the Sparse Sampling-ω algorithm.

4.1.2 Partially Observable Weighted Sparse Sampling (POWSS)

In Chapter 5, we introduce the Partially Observable Weighted Sparse Sampling (POWSS) algorithm [109]. POWSS algorithm is a continuous observation POMDP algorithm with near-optimality guarantees, without relying on any discretization schemes. This convergence guarantee justifies the particle likelihood weighting scheme and provides performance guarantees that do not directly depend on size of the state and observation spaces. Specifically, the algorithm leverages ideas from sparse sampling to limit the number of observation samples, and then using importance sampling principles to iteratively insert the state particles in each of the observation nodes by adjusting their likelihood weights.

The particle likelihood weighting technique is essential for solving continuous observation POMDPs. Fig. 5.1 illustrates the qualitative behaviors of a solver that uses observation matching and does not use particle likelihood weighting, and a solver that uses particle likelihood weighting to insert particles. The observation matching logic requires that when the state and observation are generated from a generative model, the state can only be inserted into the observation nodes in the planner tree that have matching observations. Solvers that only rely on observation matching often produce observation nodes with only one state particle inside as the probability of generating the same observation more than once is 0 for a continuous observation space. This leads to a certainty equivalent planner, which ends up planning in a QMDP fashion that is known to be sometimes suboptimal as we show in Section 5.4. Many previous notable POMDP solvers such as POMCP [159] and DESPOT [191] rely on observation matching, and thus cannot effectively handle continuous observations.

On the other hand, particle likelihood weighting provides us with a way to reason about hypothetically inserting a state particle in an observation node. By adjusting the likelihood weight of the state particle, the state particle can be inserted into an observation node even if it did not generate the same observation. This style of reasoning is very similar to how state particles are handled in many modern continuous observation POMDP solvers, such as POMCPOW [171], LABECOP [74], and DESPOT-α [61]. Thus, POWSS provides a theoretical foundation for which the particle likelihood weighting is not only necessary but also optimal.

Such combination of sparse sampling infused with particle likelihood-based belief gives rise to a continuous observation POMDP algorithm with similar performance guarantees with worst-case probability that scales as $\mathcal{O}((|A|C)^D \exp(-t \cdot C))$, including the contribution from the size of the action space for some problem specific constant t. Similar to sparse sampling, POWSS convergence rate does not directly depend on the state nor observation space size, but rather the Rényi divergence relating the observation sequence density and state transition density. While the convergence rate is rather crude, POWSS provides performance guarantees that do not directly depend on size of the state and observation spaces. POWSS is mostly meant as a theoretical tool to justify the particle likelihood weighting scheme, and is only applicable to small problems.

4.1.3 Sparse Sampling-ω

In Chapter 6, we introduce the Sparse Sampling-ω algorithm [112]. Despite the theoretical significance of POWSS, there are a few shortcomings of the POWSS algorithm. First, the algorithm definition is rather tedious, since all the state particles must be inserted into the observation node one-by-one, which makes the analysis and the definition complex. Second, POWSS is not exactly analogous to the sparse sampling algorithm in a POMDP context due to such iterative insertion of state particles. Third, POWSS only validates theoretical properties of its own convergence, and does not provide any more general guarantees for continuous observation POMDP algorithms that use particle likelihood weighting.

To address the above shortcomings, we then present Sparse Sampling-ω [112]. Sparse Sampling-ω works similarly to POWSS, since it can be thought of as the Monte Carlo equivalent of POWSS – the observations are not generated iteratively for every state particle, but rather through Monte Carlo random sampling according to the likelihood weights of the state particles in a given particle belief set. However, this rather small algorithmic difference brings many benefits:

1. Sparse Sampling-ω algorithm is directly analogous to the original sparse sampling algorithm. The particle belief states are the state nodes instead, and the transition model is now a generative particle filter.

2. Sparse Sampling-ω algorithm is much easier to define algorithmically and analyze mathematically.

3. Sparse Sampling-ω algorithm enjoys more general convergence guarantees, where it is near-optimal in both the original POMDP problem as well as the corresponding particle belief approximation of the POMDP.

The last point is incredibly significant, as this coupled convergence property enables us to prove a much more general statement: Planning in the particle belief approximation of the POMDP enables us to make similarly near-optimal decisions in the original POMDP problem. Thus, the Sparse Sampling-ω algorithm serves as a theoretical bridge that justifies much more general scheme of reasoning about particle belief approximations.

4.2 Particle Belief MDPs

In Chapter 6, we introduce general theoretical properties of the particle belief MDP approximation of POMDPs [112]. An important component of being able to make these particle belief approximations of POMDPs is the concept of particle belief MDP. The corresponding particle belief MDP for a given POMDP problem $\mathbf{P} = (S, A, O, \mathcal{T}, \mathcal{Z}, R, \gamma)$ is the MDP $\mathbf{M_P} = (\Sigma, A, \tau, \rho, \gamma)$ defined by the following elements:

- Σ: State space over particle beliefs $\bar{b}_d$. An element in this set, $\bar{b}_d \in \Sigma$, is a particle collection, $\bar{b}_d = \{(s_{d,i}, w_{d,i})\}_{i=1}^{C}$, where $s_{d,i} \in S$, $w_{d,i} \in \mathbb{R}^+$.

- A: Action space. Remains the same as the original action space.

- τ: Transition density $\tau(\bar{b}_{d+1} \mid \bar{b}_d, a)$: We define the likelihood weights $w_{d,i}$ of particles $s_{d,i}$ to be updated through unnormalized Bayes rule:

$$w_{d+1,i} = w_{d,i} \cdot \mathcal{Z}(o \mid a, s_{d+1,i}). \tag{4.1}$$

  Then, the transition probability from $\bar{b}_d$ to $\bar{b}_{d+1}$ by taking action a can be defined as:

$$\tau(\bar{b}_{d+1} \mid \bar{b}_d, a) \equiv \int_O \mathbb{P}(\bar{b}_{d+1} \mid \bar{b}_d, a, o)\mathbb{P}(o \mid \bar{b}_d, a) do. \tag{4.2}$$

- ρ: Reward function $\rho(\bar{b}_d, a)$:

$$\rho(\bar{b}_d, a) = \frac{\sum_i w_{d,i} \cdot R(s_{d,i}, a)}{\sum_i w_{d,i}}. \tag{4.3}$$

- γ: Discount factor. Remains the same as the original discount factor.

This mathematical definition of a corresponding particle belief MDP, which is further detailed in Section 6.2, not only provides a rigorous and general theoretical justification of using particle-based planners to solve POMDPs, but also allows us to directly adapt any sampling-based MDP algorithms to approximately solve a POMDP by only changing the transition generative model into a generative particle filter. As we demonstrate later, PB-MDP approximation allows Q-value convergence guarantees of the MDP algorithms to translate nicely into solving the POMDP. This powerful generality only comes at an additional multiplicative compute factor of $\mathcal{O}(C)$ at each depth, with C as the number of particles we use to approximate a belief.

Consequently, this brings us to Theorem 6.3, which we repeat below:

Theorem 6.3 (Particle Belief MDP Q-Value Approximation Optimality)**.** *Given a finite horizon POMDP* $\mathbf{P}$ *and its corresponding particle belief MDP* $\mathbf{M_P}$*, there exists a number of particles* C *for which the optimal* Q*-value of the POMDP problem* $Q^*_{\mathbf{P}}(b,a)$ *can be approximated by the optimal* Q*-value of the particle belief MDP problem* $Q^*_{\mathbf{M_P}}(\bar{b},a)$ *with arbitrary precision. Namely, under the regularity conditions (i)-(vi), the following bound holds for a given realizable belief* b*, corresponding sampled particle belief* $\bar{b}$*, and all available actions* a *with probability at least* $1-\delta_{\mathbf{M_P}}$ *for a desired accuracy* $\epsilon_{\mathbf{M_P}}$*:*

$$|Q^*_{\mathbf{P}}(b,a) - Q^*_{\mathbf{M_P}}(\bar{b},a)| \leq \epsilon_{\mathbf{M_P}}. \tag{4.4}$$

This theorem essentially claims that the optimal Q values of the POMDP and the corresponding particle belief approximation of the POMDP are close to each other with high probability, that scales with the number of particles. Therefore, Theorem 6.3 justifies the usage of particle belief approximations in general, irrespective of the sampling-based planning algorithm choice. Fig. 6.1 illustrates this relationship.

Using the above principle, we promote a sampling-based MDP planning algorithm Upper Confidence Tree (UCT) into Sparse Particle Filter Tree (Sparse-PFT) and retain similar convergence guarantees for the POMDP [21, 98, 157]. This results in an algorithm that is simple to implement and enjoys both theoretical guarantees and high performance in practice.

4.3 Voronoi Progressive Widening

In Chapter 7, we introduce the Voronoi Progressive Widening (VPW) technique [110]. Voronoi Progressive Widening is a generalization and integration of two techniques that handle continuous action spaces: Voronoi Optimistic Optimization (VOO) and Progressive Widening (PW).

4.3.1 Voronoi Optimistic Optimization

Voronoi optimistic optimization (VOO) [94] is a continuous multi-armed bandit algorithm that adaptively explores the sampling space by partitioning the space into Voronoi cells. Essentially, VOO tackles the problem of exploration-exploitation trade-off in *sampling new actions* by either deciding to sample randomly uniformly (exploration) or sample within the best Voronoi cell with the highest Q value estimate (exploitation). VOO enjoys nice theoretical guarantees in the form of expected value bounds using multi-armed bandit theory. Furthermore, it is relatively easy to implement and scales well to higher dimensions, since sampling within the best Voronoi cell only requires performing a rejection sampling based on a specified distance metric. VOO has previously been extended to Voronoi Optimistic Optimization Tree (VOOT) [94], but only for deterministic MDPs.

4.3.2 Progressive Widening

Progressive widening (PW) [43] tackles continuous state and action spaces by gradually expanding the state and action sample sets. PW approaches the problem of exploration-exploitation trade-off in *evaluating current actions* by either deciding to sample new action candidates (exploration) or proceed further down the previously visited actions (exploitation). PW dynamically limits the number of sampled children as a function of the number of visits. Many modern MCTS solvers utilize progressive widening to handle large or continuous action spaces [124, 171] due to ease of implementation and theoretical guarantees under mild conditions [12]. However, since PW samples new actions uniformly from the action space, it is rather sample inefficient and does not take into account any information gained from previous samples.

4.3.3 Voronoi Progressive Widening

Voronoi Progressive Widening (VPW) generalizes VOO and PW: VPW progressively widens with a VOO sample instead of a uniform sample. With this formulation, PW is simply VPW with $\omega = 1$, and VOO is VPW with $k_a \cdot N^{\alpha_a} = +\infty$. This leads to a simple yet effective modification to the progressive widening logic that can handle exploration on both evaluating the currently visited actions and sampling new action candidates. VPW is a general technique that can be applied to most MCTS-based solvers that build the tree sequentially. Fig. 7.1 showcases the VPW logic.

Specifically, we introduce two types of algorithms that use VPW, Voronoi Optimistic Weighted Sparse Sampling (VOWSS) and Voronoi Optimistic Monte Carlo Planning with Observation Weighting (VOMCPOW):

- **VOWSS**: VOWSS is POWSS modified with VPW, which is the first known continuous space POMDP solver with convergence guarantees.

- **VOMCPOW**: VOMCPOW is POMCPOW modified with VPW, which is a state-of-the-art practical POMDP that rivals and outperforms other solvers.

Chapter 5

Partially Observable Weighted Sparse Sampling

Chapter Outline

This chapter is based on the paper "Sparse Tree Search Optimality Guarantees in POMDPs with Continuous Observation Spaces" [109], written in collaboration with Claire Tomlin and Zachary Sunberg.

5.1 Introduction

The partially observable Markov decision process (POMDP) is a flexible mathematical framework for representing sequential decision problems where knowledge of the state is incomplete [20, 84, 96]. The POMDP formalism can represent a wide range of real world problems including autonomous driving [15, 171], cancer screening [13], spoken dialog systems [192], and others [34]. In one of the most successful applications, an approximate POMDP solution is being used in a new aircraft collision avoidance system that will be deployed worldwide [77].

A POMDP is an optimization problem for which the goal is to find a policy that specifies actions that will control the state to maximize the expectation of a reward function. One of the most popular ways to deal with the challenging computational complexity of finding such a policy [139] is to use online tree search algorithms [101, 159, 171, 191]. Instead of attempting to find a global policy that specifies actions for every possible outcome of the problem, *online* algorithms look for local approximate solutions as the agent is interacting with the environment.

Previous online approaches such as ABT [101], POMCP [159], and DESPOT [191] have exhibited good performance in large discrete domains. However, many real-world domains, notably when a robot interacts with the physical world, have continuous observation spaces, and the algorithms mentioned above will not always converge to an optimal policy in problems with continuous or naively discretized observation spaces [171].

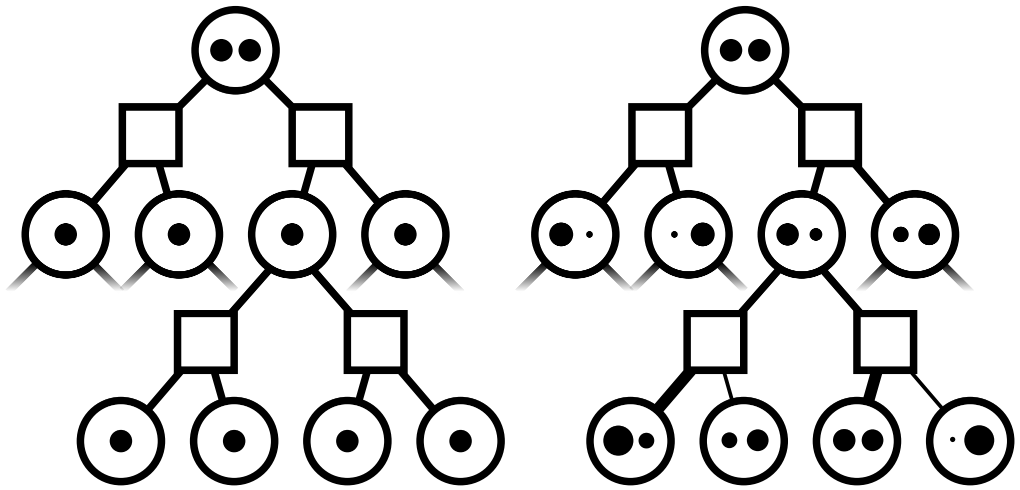


Figure 5.1: **Visual comparison of trees generated from partially observable sparse sampling (POSS) algorithm (left), and partially observable weighted sparse sampling (POWSS) algorithm (right) with depth $D = 2$ and width $C = 2$, for a continuous-observation POMDP.** Nodes below the fading edges are omitted for clarity. Square nodes correspond to actions, filled circles to state particles with size representing weight, and unfilled circles to beliefs.

Two recent approaches, POMCPOW [171] and DESPOT-α [61], have employed a weighting scheme inspired by particle filtering to achieve good performance on realistic problems with large or continuous observation spaces. However, there are currently no theoretical guarantees that these algorithms will find optimal solutions in the limit of infinite computational resources.

A convergence proof for these algorithms must have the following two components: (1) A proof that the particle weighting scheme is sound, and (2) a proof that the heuristics used to focus search on important parts of the tree are sound. This chapter tackles the first component by analyzing a new simplified algorithm that expands every node of a sparse search tree.

First, we naively extend the sparse sampling algorithm of Kearns *et al.* [92] to the partially observable domain in the spirit of POMCP and explain why this algorithm, known as partially observable sparse sampling (POSS), will converge to a suboptimal solution when the observation space is continuous. Then, we introduce appropriate weighting that results in the partially observable weighted sparse sampling (POWSS) algorithm. We prove that the value function estimated by POWSS converges to the optimal value function at a rate of $\mathcal{O}(C^D \exp(-t \cdot C))$, where C is the planning width and number of particles, D is the depth, and t is a constant specific to the problem and desired accuracy. This yields a policy that can be made arbitrarily close to optimal by increasing the computation.

POWSS is proven to converge to a globally optimal policy for POMDPs with continuous observation spaces, most notably with performance guarantees and algorithm logic that do not directly depend on size of the state and observation spaces. Since POWSS fully expands

Algorithm 5.1 Routines common to POWSS and POSS

```
Algorithm: SelectAction($b_0, \gamma, G, C, D$)
Input: Belief state $b_0$, discount $\gamma$, generative model $G$, width $C$, max depth $D$.
Output: An action $a^*$.
1: From the initial belief distribution $b_0$, sample $C$ particles and store it in $\bar{b}_0$. For POWSS, weights are also initialized with $w_i = 1/C$.
2: For each of the actions $a \in A$, calculate:
       $\hat{Q}^*_0(\bar{b}_0, a) = \text{EstimateQ}(\bar{b}_0, a, 0)$
3: Return $a^* = \arg\max_{a \in A} \hat{Q}^*_0(\bar{b}_0, a)$
```

```
Algorithm: EstimateV($\bar{b}, d$)
Input: Belief particles $\bar{b}$, current depth $d$.
Output: A scalar $\hat{V}^*_d(\bar{b})$ that is an estimate of $V^*_d(b)$.
1: If $d \geq D$ the max depth, then return 0.
2: For each of the actions $a \in A$, calculate:
       $\hat{Q}^*_d(\bar{b}, a) = \text{EstimateQ}(\bar{b}, a, d)$
3: Return $\hat{V}^*_d(\bar{b}) = \max_{a \in A} \hat{Q}^*_d(\bar{b}, a)$
```

all nodes in the sparse tree, it is not computationally efficient and is only practically applicable to toy problems. However, the convergence guarantees justify the weighting schemes in state-of-the-art efficient continuous observation POMDP algorithms like DESPOT-α and POMCPOW that solve realistic problems by only constructing the most important parts of the search tree, and suggest that these algorithm performances may not be as severely impacted by the size of the state and observation spaces.

The remainder of this chapter proceeds as follows: First, Section 5.2 presents an overview of the POSS and POWSS algorithms. Next, Section 5.3 contains an importance sampling result used in subsequent sections. Section 5.4 contains the main contribution, a proof that POWSS converges to an optimal policy using induction from the leaves to the root to prove that the value function estimate will eventually be accurate with high probability at all nodes. Finally, Section 5.5 empirically shows convergence of POWSS on a modified tiger problem [84].

5.2 Partially Observable Weighted Sparse Sampling

We first define the algorithmic elements shared by POSS and POWSS, SelectAction and EstimateV, in Algorithm 5.1. SelectAction is the entry point of the algorithm, which selects the best action for a belief b_0 according to the Q-function by recursively calling EstimateQ. EstimateV is a subroutine that returns the value, V, for an estimated belief, by calling EstimateQ for each action and returning the maximum. We use belief particle set $\bar{b}$ at every step d, which contain pairs (s_i, w_i) that correspond to the generated sample and its corresponding weight. The weight at initial step is uniformly normalized to $1/C$, as the samples are drawn directly from b_0. In Algorithms 5.1 to 5.3, we omit γ, G, C, D in the subsequent recursive calls for convenience since they are fixed globally.

We define EstimateQ functions in Algorithm 5.2 for POSS and Algorithm 5.3 for POWSS, where both methods perform sampling and recursive calls to EstimateV to es-

Algorithm 5.2 POSS

```
Algorithm: EstimateQ($\bar{b}, a, d$)
Input: Belief particles $\bar{b}$, action $a$, current depth $d$.
Output: A scalar $\hat{Q}^*_d(\bar{b}, a)$ that is an estimate of $Q^*_d(b, a)$.
1: For each particle $s_i$ in $\bar{b}$, generate $s'_i, o_i, r_i = G(s_i, a)$. If $i > |\bar{b}|$, use $s_{i \bmod |\bar{b}|}$.
2: For each unique observation $o_j$ from previous step, insert all $s'_i$ that satisfy $o_i = o_j$ to a
   new belief particle set $\overline{bao_j}$.
3: Return
   $$\hat{Q}^*_d(\bar{b}, a) = \frac{1}{C}\sum_{i=1}^{C}(r_i + \gamma \cdot \textsc{EstimateV}(\overline{bao_i}, d+1))$$
```

timate the Q-function at a given step. The crucial difference between these algorithms is shown in Fig. 5.1.

POSS naively samples the next s'_i, o_i, r_i via the generating function for each state s_i in the belief particle set $\bar{b}$, at a given step d. Then, for each unique observation o_j generated from the sampling step, POSS inserts the states s'_i into the next-step belief particle set $\overline{bao_j}$ only if the generated observation o_i matches o_j. This behavior is similar to POMCP, DESPOT, or a particle filter that uses rejection and can quickly lead to particle depletion when there are many unique observations. Finally, POSS returns the naive average of the Q-functions calculated via recursive calculation of EstimateV for each of the next-step beliefs.

On the other hand, POWSS uses particle weighting rather than using only unweighted particles with matching observation histories as in POSS. POWSS samples the next s'_i, o_i, r_i via the generating function for each state-weight pair (s_i, w_i) in the belief particle set $\bar{b}$. Now, for each observation o_j generated from the sampling step, POWSS inserts all the states s'_i and the new weights $w'_i = w_i \cdot Z(o_j|a, s'_i)$ into the next-step belief particle set $\overline{bao_j}$. These weights are the adjusted probability of hypothetically obtaining o_j from state s'_i. POWSS then returns the weighted average of the Q-functions.

5.3 Importance Sampling

We begin the theoretical portion of this work by stating important properties about self-normalized importance sampling estimators (SN estimators). One goal of importance sampling is to estimate an expected value of a function $f(x)$ where x is drawn from distribution $\mathcal{P}$, while the estimator only has access to another distribution $\mathcal{Q}$ along with the importance weights $w_{\mathcal{P}/\mathcal{Q}}(x) \propto \mathcal{P}(x)/\mathcal{Q}(x)$. This is crucial for POWSS because we wish to estimate the reward for beliefs conditioned on observation sequences, while only being able to generate the marginal distribution of states with correct probability for an action sequence.

Algorithm 5.3 POWSS

```
Algorithm: EstimateQ($\bar{b}, a, d$)
Input: Belief particles $\bar{b}$, action $a$, current depth $d$.
Output: A scalar $\hat{Q}^*_d(\bar{b}, a)$ that is an estimate of $Q^*_d(b, a)$.
1: For each particle-weight pair $(s_i, w_i)$ in $\bar{b}$, generate $s'_i, o_i, r_i$ from $G(s_i, a)$.
2: For each observation $o_j$ from previous step, iterate over $i = \{1, \cdots, C\}$ to insert $(s'_i, w_i \cdot Z(o_j|a, s'_i))$ to a new belief particle set $\overline{bao_j}$.
3: Return
   $$\hat{Q}^*_d(\bar{b}, a) = \frac{\sum_{i=1}^{C} w_i(r_i + \gamma \cdot \textsc{EstimateV}(\overline{bao_i}, d+1))}{\sum_{i=1}^{C} w_i}$$
```

We define the following quantities:

$$\tilde{w}_{\mathcal{P}/\mathcal{Q}}(x) \equiv \frac{w_{\mathcal{P}/\mathcal{Q}}(x)}{\sum_{i=1}^{N} w_{\mathcal{P}/\mathcal{Q}}(x_i)} \qquad \text{(SN Importance Weight)}$$

$$d_\alpha(\mathcal{P}||\mathcal{Q}) \equiv \mathbb{E}_{x\sim\mathcal{Q}}[w_{\mathcal{P}/\mathcal{Q}}(x)^\alpha] \qquad \text{(Rényi Divergence)}$$

$$\tilde{\mu}_{\mathcal{P}/\mathcal{Q}} \equiv \sum_{i=1}^{N} \tilde{w}_{\mathcal{P}/\mathcal{Q}}(x_i) f(x_i) \qquad \text{(SN Estimator)}$$

Theorem 5.1 (SN d_∞-Concentration Bound)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be two probability measures on the measurable space $(\mathcal{X}, \mathcal{F})$ with $\mathcal{P} \ll \mathcal{Q}$ and $d_\infty(\mathcal{P}||\mathcal{Q}) < +\infty$. Let $x_1, \cdots, x_N$ be i.i.d.r.v. sampled from $\mathcal{Q}$, and $f : \mathcal{X} \to \mathbb{R}$ be a bounded Borel function ($\|f\|_\infty < +\infty$). Then, for any $\lambda > 0$ and N large enough such that $\lambda > \|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})/\sqrt{N}$, the following bound holds with probability at least $1 - 3\exp(-N \cdot t^2(\lambda, N))$:*

$$|\,\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}}| \leq \lambda \tag{5.1}$$

where $t(\lambda, N)$ is defined as:

$$t(\lambda, N) \equiv \frac{\lambda}{\|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})} - \frac{1}{\sqrt{N}} \tag{5.2}$$

Theorem 5.1 builds upon the derivation in Proposition D.3 of Metelli *et al.* [126], which provides a polynomially decaying bound by assuming d_2 exists. Here, we compromise by further assuming that the infinite Rényi divergence d_∞ exists and is bounded to get an exponentially decaying bound: $d_\infty(\mathcal{P}||\mathcal{Q}) = \operatorname{ess\,sup}_{x\sim\mathcal{Q}} w_{\mathcal{P}/\mathcal{Q}}(x) < +\infty$. The proof of Theorem 5.1 is given in Appendix A.1.

This exponential decay is important for the proofs in Section 5.4.2. We need to ensure that all nodes of the POWSS tree at all depths d reach convergence. The branching of the tree induces a factor proportional to N^D. To offset this, we need a probabilistic bound at each depth that decays exponentially with N. Intuitive explanation of the d_∞ assumption is given at the beginning of Section 5.4.2.

5.4 Convergence Analysis

5.4.1 POSS Convergence to QMDP

We present a short informal argument for the convergence of the Q-value estimates of POSS to the QMDP value (Definition 5.1) in continuous observation spaces. Sunberg and Kochenderfer [171] provide a formal proof for a similar algorithm.

Definition 5.1 (QMDP value)**.** *Let $Q_{\text{MDP}}(s, a)$ denote the optimal Q-function evaluated at state s and action a for the fully observable MDP relaxation of a POMDP. Then, the* QMDP value *at belief b, $Q_{\text{MDP}}(b, a)$, is $\mathbb{E}_{s\sim b}\left[Q_{\text{MDP}}(s, a)\right]$.*

Since the observations o_i are drawn from a continuous distribution, the probability of obtaining duplicate o_i values in EstimateQ, line 1 is 0. Consequently, when evaluating EstimateQ, all the belief particle sets after the root node only contain a single state particle each (Fig. 5.1, left), which means that each belief node is merely an alias for a unique state particle. Therefore, EstimateV performs a rollout exactly as if the current state became entirely known after taking a single action, identical to the QMDP approximation. Since QMDP is sometimes suboptimal [84], POSS is suboptimal for some continuous-observation POMDPs.

5.4.2 Near-Optimality of POWSS

On the other hand, we claim that the POWSS algorithm can be made to perform arbitrarily close to the optimal policy by increasing the width C.

In analyzing near-optimality of POWSS, we view POWSS Q-function estimates as SN estimators, and we apply the concentration inequality result from Theorem 5.1 to show that POWSS estimates at every node have small errors with high probability. Through the near-optimality of the Q-functions, we conclude that the value obtained by employing POWSS policy is also near-optimal with further assumptions on the closed-loop POMDP system.

5.4.3 Assumptions for Analyzing POWSS

The following assumptions are needed for the proof:

(i) S and O are continuous spaces, and the action space has a finite number of elements, $|A| < +\infty$.

(ii) For any observation sequence $\{o_n\}_j$, the densities $\mathcal{Z}, \mathcal{T}, b_0$ are chosen such that the Rényi divergence of the target distribution $\mathcal{P}^d$ and sampling distribution $\mathcal{Q}^d$ (Eqs. (5.15) and (5.16)) is bounded above by $d_\infty^{\max} < +\infty$ a.s. for all $d = 0, \cdots, D-1$:

$$d_\infty(\mathcal{P}^d || \mathcal{Q}^d) = \text{ess sup}_{x \sim \mathcal{Q}^d} w_{\mathcal{P}^d/\mathcal{Q}^d}(x) \leq d_\infty^{\max}$$

(iii) The reward function R is Borel and bounded by a finite constant $||R||_\infty \leq R_{\max} < +\infty$ a.s., and $V_{\max} \equiv \frac{R_{\max}}{1-\gamma} < +\infty$.

(iv) We can sample from the generating function G and evaluate the observation probability density $\mathcal{Z}$.

(v) The POMDP terminates after $D < \infty$ steps.

Intuitively, condition (ii) means that the ratio of the conditional observation probability to the marginal observation probability cannot be too high. Additionally, our results still hold even when either of S, O are discrete, as long as it doesn't violate condition (ii), by appropriately switching the integrals to Riemann sums.

While we restrict our analysis to the case when $\gamma < 1$ for a finite horizon problem, the authors believe that similar results can be derived for either when $\gamma = 1$ or when dealing with infinite horizon problems.

Theorem 5.2 (Accuracy of POWSS Q-Value Estimates)**.** *Suppose conditions (i)-(v) are satisfied. Then, for a given $\epsilon > 0$, choosing constants C, λ, δ that satisfy:*

$$\lambda = \epsilon(1-\gamma)^2/5, \ \delta = \lambda/(V_{\max} D(1-\gamma)^2) \tag{5.3}$$

$$\delta \geq 3|A|(3|A|C)^D \exp(-C \cdot t_{\max}^2) \tag{5.4}$$

$$t_{\max}(\lambda, C) = \frac{\lambda}{3V_{\max} d_\infty^{\max}} - \frac{1}{\sqrt{C}} > 0 \tag{5.5}$$

The Q-function estimates obtained for all depths $d = 0, \cdots, D-1$ and all actions a are near-optimal with probability at least $1-\delta$:

$$\left| Q_d^*(b_d, a) - \hat{Q}_d^*(\bar{b}_d, a) \right| \leq \frac{\lambda}{1-\gamma} \tag{5.6}$$

Theorem 5.3 (POWSS Policy Convergence)**.** *In addition to conditions (i)-(v), assume that the closed-loop POMDP Bayesian belief update step is exact. Then, for any $\epsilon > 0$, we can choose a C such that the value obtained by POWSS is within ϵ of the optimal value function at b_0 a.s.:*

$$V^*(b_0) - V^{\text{POWSS}}(b_0) \leq \epsilon \tag{5.7}$$

Theorems 5.2 and 5.3 are proven sequentially in the following subsections. We generally follow the proof strategy of Kearns *et al.* [92] but with significant additions to account for the belief-based POMDP calculations rather than state-based MDP calculations. We use induction to prove a concentration inequality for the value function at all nodes in the tree, starting at the leaves and proceeding up to the root.

5.4.3.1 Value Convergence at Leaf Nodes

First, we reason about the convergence at nodes at depth $D-1$ (leaf nodes). In the subsequent analysis, we abbreviate some terms of interest with the following notation:

$$\mathcal{T}^i_{1:d} \equiv \prod_{n=1}^{d} \mathcal{T}(s_{n,i}|s_{n-1,i}, a_n) \tag{5.8}$$
$$\mathcal{Z}^{i,j}_{1:d} \equiv \prod_{n=1}^{d} \mathcal{Z}(o_{n,j}|a_n, s_{n,i})$$

Here d denotes the depth, i denotes the index of the state sample, and j denotes the index of the observation sample. Absence of indices i, j means that $\{s_n\}$ and/or $\{o_n\}$ appear as regular variables. Intuitively, $\mathcal{T}^i_{1:d}$ is the transition density of state sequence i from the root node to depth d, and $\mathcal{Z}^{i,j}_{1:d}$ is the conditional density of observation sequence j given state sequence i from the root node to depth d. Additionally, b^i_d denotes $b_d(s_{d,i})$, $r_{d,i}$ the reward $R(s_{d,i}, a_d)$, and $w_{d,i}$ the weight of $s_{d,i}$.

Since the problem ends after D steps, the Q-function for nodes at depth $D-1$ is simply the expectation of final reward and the POWSS estimate has the following form:

$$Q^*_{D-1}(b_{D-1}, a) = \int_S R(s_{D-1}, a) b_{D-1} ds_{D-1} \tag{5.9}$$
$$\hat{Q}^*_{D-1}(\bar{b}_{D-1}, a) = \frac{\sum_{i=1}^{C} w_{D-1,i} r_{D-1,i}}{\sum_{i=1}^{C} w_{D-1,i}} \tag{5.10}$$

Lemma 5.1 (SN Estimator Leaf Node Convergence). *$\hat{Q}^*_{D-1}(\bar{b}_{D-1}, a)$ is an SN estimator of $Q^*_{D-1}(b_{D-1}, a)$, and the following leaf-node concentration bound holds with probability at least $1 - 3\exp(-C \cdot t^2_{\max}(\lambda, C))$,*

$$|Q^*_{D-1}(b_{D-1}, a) - \hat{Q}^*_{D-1}(\bar{b}_{D-1}, a)| \leq \lambda \tag{5.11}$$

Proof. First, we show that $\hat{Q}^*_{D-1}(\bar{b}_{D-1}, a)$ is an SN estimator of $Q^*_{D-1}(b_{D-1}, a)$. By following the recursive belief update, the belief term can be fully expanded:

$$b_{D-1}(s_{D-1}) = \frac{\int_{S^{D-1}} (\mathcal{Z}_{1:D-1})(\mathcal{T}_{1:D-1}) b_0 ds_{0:D-2}}{\int_{S^D} (\mathcal{Z}_{1:D-1})(\mathcal{T}_{1:D-1}) b_0 ds_{0:D-1}} \tag{5.12}$$

Then, $Q^*_{D-1}(b_{D-1}, a)$ is equal to the following:

$$Q^*_{D-1}(b_{D-1}, a) = \int_S R(s_{D-1}, a) b_{D-1} ds_{D-1} \tag{5.13}$$
$$= \frac{\int_{S^D} R(s_{D-1}, a)(\mathcal{Z}_{1:D-1})(\mathcal{T}_{1:D-1}) b_0 ds_{0:D-1}}{\int_{S^D} (\mathcal{Z}_{1:D-1})(\mathcal{T}_{1:D-1}) b_0 ds_{0:D-1}} \tag{5.14}$$

We approximate the Q^* function with importance sampling by utilizing problem requirement (iv), where the target density is b_{D-1}. First, we sample the sequences $\{s_n\}_i$ according to the joint probability $(\mathcal{T}_{1:D-1})b_0$. Afterwards, we weight the sequences by the corresponding observation density $\mathcal{Z}_{1:D-1}$, obtained from the generated observation sequences $\{o_n\}_j$. For now, we assume the observation sequences $\{o_n\}_j$ are fixed.

Applying the importance sampling formalism to our system for all depths $d = 0, \cdots, D-1$, $\mathcal{P}^d$ is the normalized measure incorporating the probability of observation sequence j on top of the state sequence i until the node at depth d, and $\mathcal{Q}^d$ is the measure of the state sequence. We can think of $\mathcal{P}^d$ being indexed by the observation sequence $\{o_n\}_j$.

$$\mathcal{P}^d = \mathcal{P}^d_{\{o_n\}_j}(\{s_n\}_i) = \frac{(\mathcal{Z}^{i,j}_{1:d})(\mathcal{T}^i_{1:d})b^i_0}{\int_{S^{d+1}}(\mathcal{Z}^j_{1:d})(\mathcal{T}_{1:d})b_0 ds_{0:d}} \tag{5.15}$$

$$\mathcal{Q}^d = \mathcal{Q}^d(\{s_n\}_i) = (\mathcal{T}^i_{1:d})b^i_0 \tag{5.16}$$

$$w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_n\}_i) = \frac{(\mathcal{Z}^{i,j}_{1:d})}{\int_{S^{d+1}}(\mathcal{Z}^j_{1:d})(\mathcal{T}_{1:d})b_0 ds_{0:d}} \tag{5.17}$$

The weighing step is done by updating the self-normalized weights given in POWSS algorithm. We define $w_{d,i}$ and $r_{d,i}$ as the weights and rewards obtained at step d for state sequence i from POWSS simulation. With our recursive definition of the empirical weights, we obtain the full weight of each state sequence i for a fixed observation sequence j:

$$w_{D-1,i} = w_{D-2,i} \cdot Z(o_{D-1,j}|a_{D-1}, s_{D-1,i}) \tag{5.18}$$

$$\propto \mathcal{Z}^{i,j}_{1:D-1} \tag{5.19}$$

Realizing that the marginal observation probability is independent of indexing by i, we show that $\hat{Q}^*_{D-1}(\bar{b}_{D-1}, a)$ is an SN estimator of $Q^*_{D-1}(b_{D-1}, a)$:

$$\hat{Q}^*_{D-1}(\bar{b}_{D-1}, a) = \frac{\sum_{i=1}^{C}(\mathcal{Z}^{i,j}_{1:D-1})R(s_{D-1,i}, a)}{\sum_{i=1}^{C}(\mathcal{Z}^{i,j}_{1:D-1})} \tag{5.20}$$

$$= \frac{\sum_{i=1}^{C} \frac{(\mathcal{Z}^{i,j}_{1:D-1})}{\int_{S^D}(\mathcal{Z}^j_{1:D-1})(\mathcal{T}_{1:D-1})b_0 ds_{0:D-1}} R(s_{D-1,i}, a)}{\sum_{i=1}^{C} \frac{(\mathcal{Z}^{i,j}_{1:D-1})}{\int_{S^D}(\mathcal{Z}^j_{1:D-1})(\mathcal{T}_{1:D-1})b_0 ds_{0:D-1}}} \tag{5.21}$$

$$= \frac{\sum_{i=1}^{C} w_{\mathcal{P}^{D-1}/\mathcal{Q}^{D-1}}(\{s_n\}_i)R(s_{D-1,i}, a)}{\sum_{i=1}^{C} w_{\mathcal{P}^{D-1}/\mathcal{Q}^{D-1}}(\{s_n\}_i)} \tag{5.22}$$

$$= \sum_{i=1}^{C} \tilde{w}_{\mathcal{P}^{D-1}/\mathcal{Q}^{D-1}}(\{s_n\}_i)R(s_{D-1,i}, a) \tag{5.23}$$

Since $\{s_n\}_1, \cdots \{s_n\}_C$ are i.i.d.r.v. sequences of depth D sampled from $\mathcal{Q}^{D-1}$, and R is a bounded function from problem requirement (iii), we can apply the SN concentration bound in Theorem 5.1 to prove Lemma 5.1. Detailed finishing steps of the proof are given in Appendix A.2. □

5.4.3.2 Induction from Leaf to Root Nodes

Now, we want to show that nodes at all depths have convergence guarantees via induction.

Lemma 5.2 (SN Estimator Step-by-Step Convergence). *$\hat{Q}^*_d(\bar{b}_d, a)$ is an SN estimator of $Q^*_d(b_d, a)$, and for all $d = 0, \cdots, D-1$ and a, the following holds with probability at least $1 - 3|A|(3|A|C)^D \exp(-C \cdot t^2_{\max})$:*

$$|Q^*_d(b_d, a) - \hat{Q}^*_d(\bar{b}_d, a)| \le \alpha_d \tag{5.24}$$

$$\alpha_d \equiv \lambda + \gamma\alpha_{d+1};\ \alpha_{D-1} = \lambda \tag{5.25}$$

Proof. First, we set C such that $C > (3V_{\max} d^{\max}_{\infty}/\lambda)^2$ to satisfy $t_{\max}(\lambda, C) > 0$, which ensures that the SN concentration inequality holds with probability $1 - 3\exp(-C \cdot t^2_{\max}(\lambda, C))$ at any given step d and action a. Furthermore, we multiply the worst-case union bound factor $(3|A|C)^D$, since we want the function estimates to be within their respective concentration bounds for all the actions $|A|$ and child nodes C at each step $d = 0, \cdots, D-1$, for the 3 times we use SN concentration bound in the induction step. We once again multiply the final δ by $|A|$ to account for the root node Q-value estimates also satisfying their respective concentration bounds for all actions.

Following our definition of EstimateQ, the value function estimates at step d are given as the following:

$$\hat{V}^*_d(\bar{b}_d) = \max_{a \in A} \hat{Q}^*_d(\bar{b}_d, a) \tag{5.26}$$

$$\hat{Q}^*_d(\bar{b}_d, a) = \frac{\sum_{i=1}^{C} w_{d,i} \left(r_{d,i} + \gamma \hat{V}^*_{d+1}(\overline{b_d a o_i}) \right)}{\sum_{i=1}^{C} w_{d,i}} \tag{5.27}$$

The base case $d = D-1$ holds by Lemma 5.1. Then for the inductive step, we assume Eq. (5.24) holds for all actions at step $d+1$. Using the triangle inequality for step d, we split the difference into two terms, the reward estimation error (A) and the next-step value estimation error (B):

$$|Q^*_d(b_d, a) - \hat{Q}^*_d(\bar{b}_d, a)| \le \underbrace{\left| \mathbb{E}[R(s_d, a)|b_d] - \frac{\sum_{i=1}^{C} w_{d,i} r_{d,i}}{\sum_{i=1}^{C} w_{d,i}} \right|}_{\text{(A)}} \tag{5.28}$$

$$+ \gamma \underbrace{\left| \mathbb{E}[V^*_{d+1}(bao)|b_d] - \frac{\sum_{i=1}^{C} w_{d,i} \hat{V}^*_{d+1}(\overline{b_d a o_i})}{\sum_{i=1}^{C} w_{d,i}} \right|}_{\text{(B)}} \tag{5.29}$$

Each of the error terms are bound by $(A) \le \frac{R_{\max}}{3V_{\max}}\lambda$ and $(B) \le \frac{1}{3}\lambda + \frac{2}{3\gamma}\lambda + \alpha_{d+1}$. We provide a detailed justification of these bounds in Appendix A.3, which uses the SN concentration bound 3 times. Combining (A) and (B), we prove the inductive hypothesis:

$$\begin{aligned} |Q_d^*(b_d, a) - \hat{Q}_d^*(\bar{b}_d, a)| &\leq \frac{R_{\max}}{3V_{\max}}\lambda + \gamma[\frac{1}{3}\lambda + \frac{2}{3\gamma}\lambda + \alpha_{d+1}] \\ &\leq \lambda + \gamma\alpha_{d+1} = \alpha_d \end{aligned} \tag{5.30}$$

Therefore, Eq. (5.24) holds for all $d = 0, \cdots, D-1$ with probability at least $1 - 3|A|(3|A|C)^D \exp(-C \cdot t_{\max}^2)$. □

Proof. (Theorem 5.2) First, we choose constants C, λ, δ and densities $\mathcal{Z}, \mathcal{T}, b_0$ that satisfy the conditions in Theorem 5.2. Since $\alpha_d \leq \alpha_0$, the following holds for all $d = 0, \cdots, D-1$ with probability at least $1 - \delta$ through Lemmas 5.1 and 5.2:

$$|Q_d^*(b_d, a) - \hat{Q}_d^*(\bar{b}_d, a)| \leq \alpha_0 = \sum_{d=0}^{D-1} \gamma^d \lambda \leq \frac{\lambda}{1-\gamma} \tag{5.31}$$

Note that the convergence rate δ is $\mathcal{O}(C^D \exp(-tC))$, where $t = (\lambda/(3V_{\max} d_\infty^{\max}))^2$. □

5.4.3.3 Near-Optimal Policy Performance

We have proven in the previous subsection that the planning step results in a near-optimal Q-value for a given belief. Assuming further that we have a perfect Bayesian belief update in the outer observe-plan-act loop, we prove Theorem 5.3, which states that the closed-loop POMDP policy generated by POWSS at each planning step results in a near-optimal policy. The proof given in Appendix A.4 combines Theorem 5.2 with results from Kearns *et al.* [92]; Singh and Yee [161]:

$$V^*(b_0) - V^{\text{POWSS}}(b_0) \leq \epsilon \tag{5.32}$$

5.5 Experiments

The simple numerical experiments in this section confirm the theoretical results of Section 5.4. Specifically, they show that the value function estimates of POSS converge to the QMDP approximation and the value function estimates of POWSS converge to the optimal value function for a toy problem.

5.5.1 Continuous Observation Tiger Problem

We consider a simple modification of the classic tiger problem [84] that we refer to as the continuous observation tiger (CO-tiger) problem. In the CO-tiger problem, the agent is presented with two doors, left (L) and right (R). One door has a tiger behind it ($S = \{\texttt{Tiger L}, \texttt{Tiger R}\}$). In the classic problem, the agent can either open one of the doors or listen, and the CO-tiger problem has an additional wait action to illustrate the suboptimality of QMDP estimates ($A = \{\texttt{Open L}, \texttt{Open R}, \texttt{Wait}, \texttt{Listen}\}$). If the agent opens a door, the problem terminates immediately; If the tiger is behind that door, a penalty of -10 is received, but if not, a reward of 10 is given. Waiting has a penalty of -1 and listening has a penalty of -2. If the agent waits or listens, a noisy continuous observation between 0 and 1 is received ($O = [0, 1]$). In the wait case, this observation is uniformly distributed, independent of the tiger's position, yielding no information. In the listen case, the observation distribution is piecewise uniform. An observation in $[0, 0.5]$ corresponds to a tiger behind the left door and $(0.5, 1]$ the right door. Listening yields an observation in the correct range 85% of the time. The discount is 0.95, and the terminal depth is 3.

The optimal solution to this problem may be found by simply discretizing the observation space so that any continuous observation in $[0, 0.5]$ is treated as a `Tiger L` observation, and any continuous observation in $(0.5, 1]$ is treated as a `Tiger R` observation. This fully discrete version of the problem may be easily solved by a classical solution method such as the incremental pruning method of Cassandra *et al.* [33]. Given an evenly-distributed initial belief, the optimal action is `Listen` with a value of 4.65, and the `Wait` action has a value of 3.42. The QMDP estimate for `Wait` is 8.5 and for `Listen` is 7.5.

While the CO-tiger problem is too small to be of practical significance, it serves as an empirical demonstration that POWSS converges toward the optimal value estimates and that POSS converges toward the QMDP estimates. In fact, the QMDP estimates generated by POSS are suboptimal in this example and lead to picking the suboptimal `Wait` action. Both POWSS and POSS were implemented using the POMDPs.jl framework, [53] and open-source code can be found at `https://github.com/JuliaPOMDP/SparseSampling.jl`.

5.5.2 Results

The results plotted in Fig. 5.2 show the Q-value estimates of POWSS converging toward the optimal Q-values as the width C is increased. Each data point represents the mean Q-value from 200 runs of the algorithm from a uniformly-distributed belief, with the standard deviation plotted as a ribbon. The estimates for POSS have no uncertainty bounds since the estimates in this problem are the same for all C.

With $C = 1$, POWSS suffers from particle depletion and, because of the particular structure of this problem, finds the QMDP Q-values. As C increases, one can observe that both bias and variance in the Q-value estimates significantly decrease in agreement with our theoretical results, while POSS continues to yield incorrect estimates.

Some estimates by POMCPOW are also included. These are not directly comparable since POMCPOW is parameterized differently. For these tests, the double progressive widening parameters $k_o = C$, $\alpha_o = 0$ were used to limit the tree width, with $n = C^3$ iterations to keep the particle density high in wider trees (see Sunberg and Kochenderfer [171] for parameter definitions). POMCPOW's value estimates are strongly biased downwards by exploration actions, but the estimated value for `Listen` action is much higher than the estimated value for the `Wait` action, which is too low to appear on the plot. Thus the correct

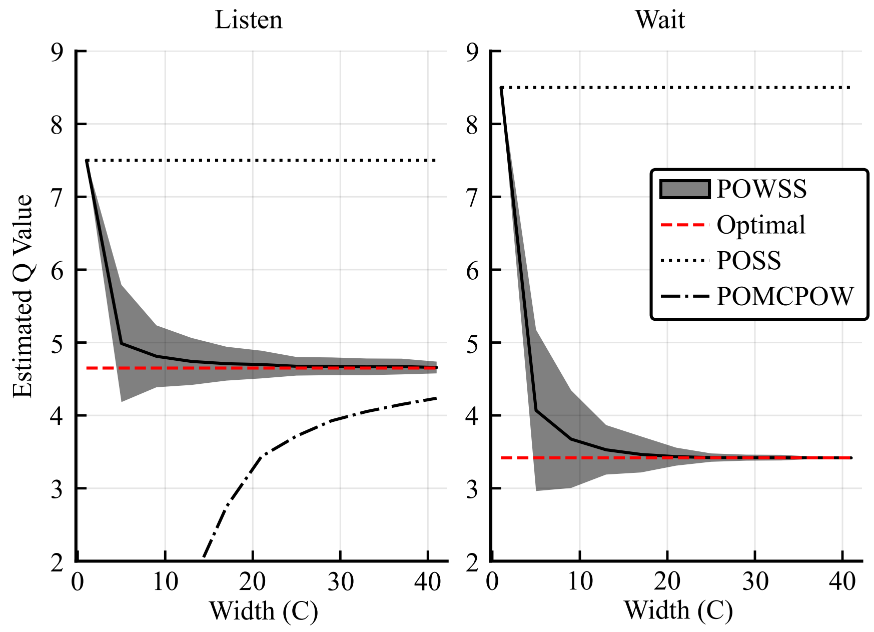


Figure 5.2: **Numerical convergence of Q-value estimates for POSS, POWSS, and POMCPOW in the CO-tiger problem.** Ribbons indicate standard deviation.

action will usually still be chosen. At $C = 41$, POMCPOW is about an order of magnitude faster than POWSS.

5.6 Conclusion

This work has proposed two new POMDP algorithms and analyzed their convergence in POMDPs with continuous observation spaces. Though these algorithms are not computationally efficient and thus not suitable for realistic problems, this work lays the foundation for analysis of more complex algorithms, rigorously justifying the observation likelihood weighting used in POWSS, POMCPOW, and DESPOT-α.

There is a great deal of future work to be done along this path. Most importantly, the theory presented in this work should be extended to more computationally efficient and hence practical algorithms. Before extending to POMCPOW and DESPOT-α, it may be beneficial to apply these techniques to an algorithm that is less conceptually complex, such as a modification of Sparse-UCT [21] extended to partially observable domains. Such an algorithm could enjoy strong theoretic guarantees, ease of implementation, and good performance on large problems.

Moreover, the proof techniques in this work may yield insight into which problems are difficult for sparse tree search techniques. For example, the Rényi divergence between the marginal and conditional state distributions (assumption (ii)) may be a difficulty indicator for likelihood-weighted sparse tree solvers, similar to the covering number of the optimal reachable belief space for point-based solvers [105].

Chapter 6

Optimality Guarantees for Particle Belief Approximation of POMDPs

Chapter Outline

This chapter is based on the paper "Optimality Guarantees for Particle Belief Approximation of POMDPs" [112], written in collaboration with Tyler J. Becker, Mykel J. Kochenderfer, Claire J. Tomlin, and Zachary N. Sunberg.

6.1 Introduction

Maintaining safety and acting efficiently in the midst of uncertainty is an important aspect in a diverse set of challenges from transportation [77, 172] to autonomous scientific exploration [25, 58], to healthcare [13] and ecology [123]. The partially observable Markov decision process (POMDP) is a flexible framework for sequential decision making in uncertain environments.

One common method for solving POMDPs is online tree search, which is attractive for several reasons. First, the approach scales to very large problems because it uses sampled trajectories, making it insensitive to the dimensionality of the state and observation spaces [92]. Second, since online computation focuses on the current states and states likely to be encountered in the future, it can reduce the need for offline computation and end-to-end training [48]. Third, tree search is applicable to a wide range of problems, for example hybrid continuous-discrete and non-convex problems, because it only depends on a minimal set of problem structure requirements.

Recently proposed POMDP tree search algorithms [61, 74, 109, 110, 124, 171, 187] have been shown empirically to work on continuous state and observation spaces. Theoretical analysis, however, has lagged behind. While there are algorithms that have performance guarantees [109, 110] and algorithms that perform well empirically [61, 74, 110, 124, 171, 187], there has been little progress on generalization of why such family of algorithms can enjoy performance guarantees. For instance, AdaOPS [187], a recent particle belief-based POMDP solver that we include in our analysis, comes very close to capturing both theory and practice, but it has a complex algorithmic structure and provides only algorithm specific guarantees with additional simplifying assumptions. The considerable gap in the connection between POMDPs and practical approximations using particle methods still remains, only partially answered by a few algorithm-specific guarantees.

This chapter formally justifies that optimality guarantees in a finite sample particle belief MDP (PB-MDP) approximation of a POMDP/belief MDP yields optimality guarantees in the original POMDP as well. We accomplish this by showing that the Q-values of the POMDP and PB-MDP are close with high probability by using an intermediary theoretical algorithm called Sparse Sampling-ω. Specifically, we prove that the Sparse Sampling-ω Q-value estimates are close to both optimal Q-values of the POMDP and PB-MDP with high probability. This in turn implies that since there exists an algorithm that approximates both Q-values accurately with high probability, the optimal Q-values of the POMDP and PB-MDP themselves must be close to each other with high probability that scales as $1 - \mathcal{O}(C^D \exp(-t \cdot C))$ where C is the number of particles, D is the planning depth, and t is a number determined by the POMDP reward function and probability distribution, number of particles, and desired accuracy. Notably, this convergence rate does not directly depend on the size of the state space nor the observation space, but rather depends on the Rényi divergence that links the probabilities concerning state and observation trajectories and the planning horizon D.

This fundamental bridge between PB-MDPs and POMDPs allows us to adapt any sampling based MDP algorithm of choice to a POMDP by solving the corresponding particle belief MDP approximation and preserve the convergence guarantees in the POMDP. Practically, this means additionally assuming we have an explicit observation model $\mathcal{Z}$, and then simply swapping out the state transition generative model with a particle filtering-based model, which only increases the computational complexity of transition generation by a factor of $\mathcal{O}(C)$, with C the number of particles in a particle belief state. This allows us to devise algorithms such as Sparse Particle Filter Tree (Sparse-PFT), which enjoys algorithmic simplicity, theoretical guarantees and practicality, as it is equivalent to upper confidence trees (UCT) [21, 157], with particle belief states.

The remainder of this chapter proceeds as follows: First, Section 6.2 formalizes the notion of particle belief MDPs (PB-MDPs) and gives detailed mathematical treatment. Then, Section 6.3 introduces Sparse Sampling-ω algorithm, and proves its coupled convergence towards the optimal Q-values of a POMDP and its corresponding PB-MDP in Theorem 6.2. Section 6.4 formally bridges the gap between POMDPs and PB-MDPs by leveraging the coupled convergence of Sparse Sampling-ω. In this section, we present two main theorems: Theorem 6.3 shows the optimal Q-value bounds between POMDP and PB-MDP, and Theorem 6.4 shows the near-optimality of planning in PB-MDP to solve a POMDP in closed-loop. We also introduce Sparse-PFT, a practical example of generating a PB-MDP approximation algorithm from an MDP algorithm. Finally, Section 6.5 empirically shows the performance

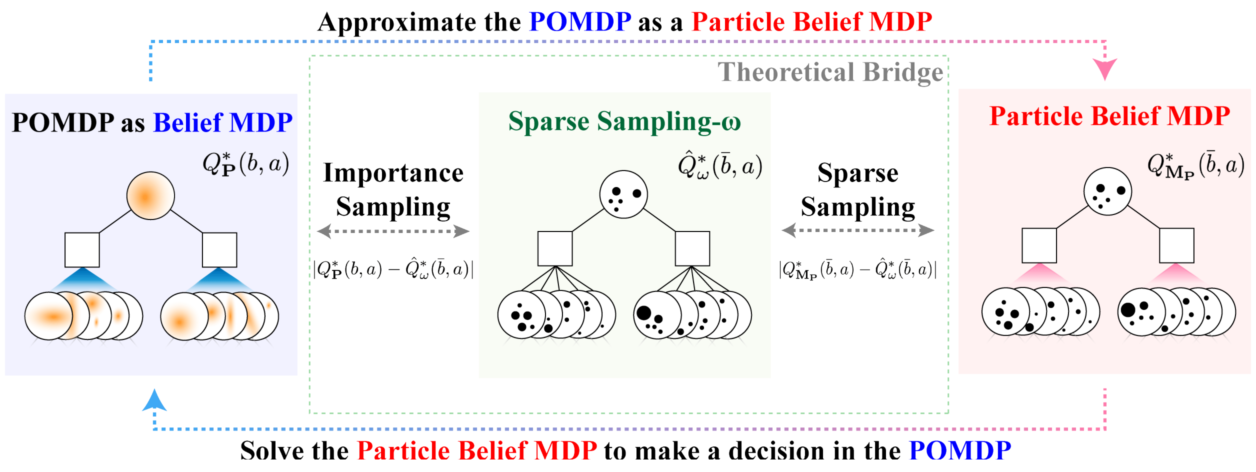

Approximate the POMDP as a Particle Belief MDP
Theoretical Bridge
POMDP as Belief MDP
$Q^*_{\mathbf{P}}(b,a)$
Importance Sampling
$|Q^*_{\mathbf{P}}(b,a) - \hat{Q}^*_{\omega}(\bar{b},a)|$
Sparse Sampling-ω
$\hat{Q}^*_{\omega}(\bar{b},a)$
Sparse Sampling
$|Q^*_{\mathbf{M_P}}(\bar{b},a) - \hat{Q}^*_{\omega}(\bar{b},a)|$
Particle Belief MDP
$Q^*_{\mathbf{M_P}}(\bar{b},a)$
Solve the Particle Belief MDP to make a decision in the POMDP

Figure 6.1: **Illustration of Particle Belief MDP approximation of POMDPs and its theoretical justification.** Since Sparse Sampling-ω algorithm Q-value estimator converges to both the optimal Q-values of POMDP and PB-MDP, such an existence of algorithm implies that the optimal Q-values of POMDP and PB-MDP are also close to each other with high probability. This enables us to approximate the POMDP problem as a PB-MDP, and then solve the PB-MDP with an MDP algorithm to make a decision in the original POMDP while retaining the guarantees and computational efficiencies of the original MDP algorithm.

of Sparse-PFT and other practical continuous observation POMDP algorithms over five different simulation experiments, and validates the improvements in performance of PB-MDP approximation with the increase in number of particles C while keeping other hyperparameters fixed.

6.2 Particle Belief MDPs (PB-MDPs)

In this section, we define the corresponding particle belief MDP (PB-MDP) for a given POMDP. Deriving the corresponding particle belief MDP of a POMDP is equivalent to approximating the belief MDP with a finite number of particles.

Definition 6.1 (Particle Belief MDP)**.** *The corresponding particle belief MDP for a given POMDP problem* $\mathbf{P} = (S, A, O, \mathcal{T}, \mathcal{Z}, R, \gamma)$ *is the MDP* $\mathbf{M_P} = (\Sigma, A, \tau, \rho, \gamma)$ *defined by the following elements:*

- Σ: State space over particle beliefs $\bar{b}_d$. An element in this set, $\bar{b}_d \in \Sigma$, is a particle collection, $\bar{b}_d = \{(s_{d,i}, w_{d,i})\}_{i=1}^{C}$, where $s_{d,i} \in S$, $w_{d,i} \in \mathbb{R}^+$.[1] Note that this space is not permutation invariant over the particles, meaning that particles of different orders are considered different elements in Σ.

[1]The d subscript, the number of steps, is included for subscript order consistency with the rest of the chapter, but is not meaningful in this context.

- A: Action space. Remains the same as the original action space.
- τ: Transition density $\tau(\bar{b}_{d+1} \mid \bar{b}_d, a)$: We define the likelihood weights $w_{d,i}$ of particles $s_{d,i}$ to be updated through unnormalized Bayes rule:

$$w_{d+1,i} = w_{d,i} \cdot \mathcal{Z}(o \mid a, s_{d+1,i}). \tag{6.1}$$

Then, the transition probability from $\bar{b}_d$ to $\bar{b}_{d+1}$ by taking the action a can be defined as:

$$\tau(\bar{b}_{d+1} \mid \bar{b}_d, a) \equiv \int_O \mathbb{P}(\bar{b}_{d+1} \mid \bar{b}_d, a, o)\mathbb{P}(o \mid \bar{b}_d, a)do. \tag{6.2}$$

The first term in the integrand product $\mathbb{P}(\bar{b}_{d+1} \mid \bar{b}_d, a, o)$ is the conditional transition density given some observation o. Note that the likelihood weight Bayesian update step is deterministic given $s_{d,i}, s_{d+1,i}, a$ and o. Combining with the facts that the state transition density update for each particle is independent and that the case when $\bar{b}_{d+1} = \{s'_i, w'_i\}$ results in the only nonzero integrand, such likelihood calculation simplifies into the product of the transition densities for each i-th index particle if such transition is valid:

$$\begin{align}
\mathbb{P}(\bar{b}_{d+1} \mid \bar{b}_d, a, o) &= \int_\Sigma \mathbb{P}(\bar{b}_{d+1} \mid \bar{b}_d, a, o, \{s'_i, w'_i\})\mathbb{P}(\{s'_i, w'_i\} \mid \bar{b}_d, a, o)d\sigma \tag{6.3}\\
&= \int_\Sigma \delta\left[\bar{b}_{d+1} = \{s'_i, w'_i\}\right] \prod_{i=1}^{C} \mathcal{T}(s'_i \mid s_{d,i}, a)d\sigma \tag{6.4}\\
&= \begin{cases} \prod_{i=1}^{C} \mathcal{T}(s_{d+1,i} \mid s_{d,i}, a) & \text{if } w_{d+1,i} = w_{d,i} \cdot \mathcal{Z}(o \mid a, s_{d+1,i}) \ \forall i \\ 0 & \text{otherwise.} \end{cases} \tag{6.5}
\end{align}$$

The second term in the integrand product $\mathbb{P}(o \mid \bar{b}_d, a)$ is the observation likelihood given a particle belief and an action. This is equivalent to weighted sum of observation likelihoods conditioning on the observation having been generated from the respective i-th particle:

$$\begin{align}
\mathbb{P}(o \mid \bar{b}_d, a) = \mathbb{P}(o \mid \{s_{d,i}, w_{d,i}\}, a) &= \frac{\sum_{i=1}^{C} w_{d,i} \cdot \mathbb{P}(o \mid s_{d,i}, a)}{\sum_{i=1}^{C} w_{d,i}} \tag{6.6}\\
&= \frac{\sum_{i=1}^{C} w_{d,i} \cdot [\int_S \mathcal{Z}(o \mid a, s')\mathcal{T}(s' \mid s_{d,i}, a)ds']}{\sum_{i=1}^{C} w_{d,i}}. \tag{6.7}
\end{align}$$

Note that this density τ is usually impossible or very difficult to calculate explicitly. However, it is rather easy to sample from it using generative models.

- ρ: Reward function $\rho(\bar{b}_d, a)$:

$$\rho(\bar{b}_d, a) = \frac{\sum_i w_{d,i} \cdot R(s_{d,i}, a)}{\sum_i w_{d,i}}. \tag{6.8}$$

Note that if R is bounded by $R_{\max}$, ρ is also bounded with $||\rho||_\infty \leq R_{\max}$, since the normalized weights sum to 1.

- γ: Discount factor. Remains the same as the original discount factor.

The significance of defining a corresponding particle belief MDP is that we can directly adapt any sampling-based MDP algorithms to approximately solve a POMDP by only changing the transition generative model. The transition generative model will now be a sampler based on particle filtering, as the particle belief MDP deals with particle belief states. Furthermore, this allows Q-value convergence guarantees of the MDP algorithms to translate nicely into solving the POMDP, as we will prove later in this chapter that the optimal Q-values of the POMDP $Q^*_{\mathbf{P}}$ and PB-MDP $Q^*_{\mathbf{M_P}}$ are close with high probability.

6.3 Sparse Sampling-ω

In order to show that the optimal Q-values of the POMDP $Q^*_{\mathbf{P}}$ and PB-MDP $Q^*_{\mathbf{M_P}}$ are approximately equivalent, we first introduce an algorithm called Sparse Sampling-ω (sparse sampling with weights), which will serve as a theoretical bridge between POMDP and PB-MDP. Sparse Sampling-ω is a sparse sampling solver that uses particle belief states with particle likelihood weighting to deal with observation uncertainty. As is evident from the name, Sparse Sampling-ω takes inspiration from sparse sampling [92] for continuous state MDPs, using particle belief states. This algorithm can also be seen as a slight modification of POWSS [109], the first known algorithm to enjoy convergence guarantees to the optimal policy for continuous observation POMDP problems with performance guarantees and algorithm logic that do not directly depend on size of the state and observation spaces. This duality effectively lets us bridge POMDPs and PB-MDPs. Note that Sparse Sampling-ω is purely a theoretical intermediary tool to bridge POMDPs and PB-MDPs, and fully expanding the state and action nodes is extremely computationally inefficient. Rather, the existence of such theoretically well-behaved algorithm is what lets us effectively bridge the gap between the optimal Q-values of the POMDP $Q^*_{\mathbf{P}}$ and PB-MDP $Q^*_{\mathbf{M_P}}$.

6.3.1 Algorithm Definition

First, we define the auxiliary functions for Sparse Sampling-ω in Algorithm 6.1, which are also applicable to any sampling-based particle belief MDP planner. GenPF is the helper function to generate the next-step particle belief set, where the particles are evolved through the transition density $\mathcal{T}$ and the weights are updated through the observation density $\mathcal{Z}$. In GenPF, the sampled states s'_i are inserted into each next-step particle belief set $\overline{bao_j}$ with the new weights $w'_i = w_i \cdot \mathcal{Z}(o_j \mid a, s'_i)$, which are the adjusted probability of hypothetically sampling observation o_j from state s'_i. Furthermore, the reward returned by GenPF is the particle likelihood weighted reward $\rho = \sum_i w_i r_i / \sum_i w_i$ of the current particle belief state, which is a constant output for a fixed pair of $\bar{b}, a$. The Plan function is the entry point

Algorithm 6.1 Auxiliary Procedures for Sparse Sampling-ω and other PB-MDP algorithms

```
Algorithm: GenPF($\bar{b}, a$)
Input: particle belief set $\bar{b} = \{(s_i, w_i)\}$, action $a$.
Output: New updated particle belief set $\bar{b}' = \{(s'_i, w'_i)\}$, mean reward $\rho$.
1: $s_o \leftarrow$ sample $s_i$ from $\bar{b}$ w.p. $\frac{w_i}{\sum_i w_i}$
2: $o \leftarrow G(s_o, a)$
3: for $i = 1, \ldots, C$ do
4:     $s'_i, r_i \leftarrow G(s_i, a)$
5:     $w'_i \leftarrow w_i \cdot \mathcal{Z}(o|a, s'_i)$
6: $\bar{b}' \leftarrow \{(s'_i, w'_i)\}_{i=1}^{C}$
7: $\rho \leftarrow \sum_i w_i r_i / \sum_i w_i$
8: return $\bar{b}', \rho$
```

```
Algorithm: Plan($b$)
Input: Belief $b$.
Output: An action $a$.
1: for $i = 1, \ldots, C$ do
2:     $s \leftarrow$ sample from $b$
3:     $\bar{b} \leftarrow \bar{b} \cup \{(s, 1/C)\}$
4: for $i = 1, \ldots, n$ do
5:     Simulate($\bar{b}, 0$)
6: return $a \leftarrow \arg\max_{a \in C(b)} Q(b, a)$
```

Algorithm 6.2 Sparse Sampling-ω Algorithm

```
Global Variables: $\gamma, G, C, D$.
Algorithm: EstimateV($\bar{b}, d$)
Input: particle belief set $\bar{b} = \{(s_i, w_i)\}$, depth $d$.
Output: A scalar $\hat{V}^*_d(\bar{b})$ that is an estimate of $V^*(\bar{b})$.
1: if $d \geq D$ then
2:     return 0
3: for $a \in A$ do
4:     $\hat{Q}^*_d(\bar{b}, a) \leftarrow$ EstimateQ($\bar{b}, a, d$)
5: return $\hat{V}^*_d(\bar{b}) \leftarrow \max_{a \in A} \hat{Q}^*_d(\bar{b}, a)$
```

```
Algorithm: EstimateQ($\bar{b}, a, d$)
Input: particle belief set $\bar{b} = \{(s_i, w_i)\}$, action $a$, depth $d$.
Output: A scalar $\hat{Q}^*_d(\bar{b}, a)$ that is an estimate of $Q^*_d(b, a)$.
1: for $i = 1, \ldots, C$ do
2:     $\bar{b}'_i, \rho \leftarrow$ GenPF($\bar{b}, a$)
3:     $\hat{V}^*_{d+1}(\bar{b}'_i) \leftarrow$ EstimateV($\bar{b}'_i, d + 1$)
4: return $\hat{Q}^*_d(\bar{b}, a) \leftarrow \rho + \frac{1}{C} \sum_{i=1}^{C} \gamma \cdot \hat{V}^*_{d+1}(\bar{b}'_i)$
```

to the Sparse-PFT algorithm, which initializes the particle belief state, subsequently calls Simulate to build the tree and returns the best action with the highest Q-value.

We define the main planning functions in Sparse Sampling-ω, EstimateV and EstimateQ functions, in Algorithm 6.2. The global variables are the discount factor γ, the generative model G, the observation width C, and the planning depth D. We use particle belief set $\bar{b}$ at every step d, which contain pairs (s_i, w_i) that correspond to the generated sample and its corresponding weight. EstimateV is a subroutine that returns the value function V, for an estimated state or belief, by calling EstimateQ for each action and returning the maximum. Similarly, EstimateQ performs sampling and recursive calls to EstimateV to estimate the Q-function at a given step with a weighted average. In EstimateQ, Sparse Sampling-ω samples the next particle belief state using GenPF.

Consequently, the Sparse Sampling-ω policy action can be obtained by calling the value estimation function $\textsc{EstimateV}(\bar{b}_0, 0)$ at the root node and taking an action that maximizes the Q-value. The particle belief set is initialized by drawing samples from b_0 and setting weights to $1/C$, as the samples were drawn directly from b_0. Sparse Sampling-ω is not computationally efficient as it fully expands the sparsely sampled tree with full particle belief states, and serves only to demonstrates theoretical convergence and is only practically applicable to very small toy POMDP problems.

Sparse Sampling-ω is identical to the sparse sampling algorithm [92] planning on a particle belief MDP. It also is a slight modification of the previously-published POWSS algorithm [109]. Specifically, whereas POWSS generates exactly one observation and corresponding new belief for each particle in a belief, Sparse Sampling-ω randomly selects a state to generate the observation each time $\textsc{GenPF}$ is called in Line 2 of Algorithm 6.2. This means that Sparse Sampling-ω performs a Monte Carlo sampling estimate of the next step value, while POWSS performs an importance weighted summation over the estimates.

Most importantly, this duality of being a modification of POWSS algorithm maintaining similar convergence guarantees for POMDPs while simultaneously being an adaptation of the sparse sampling algorithm for particle belief MDP makes it the ideal candidate to bridge POMDPs and PB-MDPs together. As an added benefit, the definition of Sparse Sampling-ω is much simpler than the original POWSS algorithm, while still allowing us to use similar analysis techniques used in both POWSS and sparse sampling.

6.3.2 Theoretical Analysis

In this section, we will prove that Sparse Sampling-ω algorithm can be made to approximate both optimal Q-values of the POMDP $Q^*_{\mathbf{P}}$ and PB-MDP $Q^*_{\mathbf{M_P}}$ arbitrarily close by increasing the observation width C. Theorem 6.2 proves that the Sparse Sampling-ω algorithm approximates these Q-values with high probability by combining results from self-normalized importance sampling estimators and POWSS optimality proofs [109] to prove the optimality in $Q^*_{\mathbf{P}}$, and sparse sampling proof [92] to prove the optimality in $Q^*_{\mathbf{M_P}}$.

6.3.2.1 Importance Sampling

We begin the theoretical portion of this work by stating an important property about self-normalized importance sampling estimators (SN estimators). We have previously published this property [109] but present it again here because of its importance to our analysis. One goal of importance sampling is to estimate an expected value of a function $f(x)$ where x is drawn from a distribution $\mathcal{P}$ while the estimator only has access to another distribution $\mathcal{Q}$ along with the importance weights $w_{\mathcal{P}/\mathcal{Q}}(x) \propto \mathcal{P}(x)/\mathcal{Q}(x)$. This technique is crucial for Sparse Sampling-ω because we wish to estimate the value for beliefs conditioned on observation sequences while only being able to sample from the marginal distribution of states for a given action sequence.

We define the following quantities:

$$\tilde{w}_{\mathcal{P}/\mathcal{Q}}(x) \equiv \frac{w_{\mathcal{P}/\mathcal{Q}}(x)}{\sum_{i=1}^{N} w_{\mathcal{P}/\mathcal{Q}}(x_i)} \qquad \text{(SN Importance Weight)}$$

$$d_\alpha(\mathcal{P}||\mathcal{Q}) \equiv \mathbb{E}_{x\sim\mathcal{Q}}[w_{\mathcal{P}/\mathcal{Q}}(x)^\alpha] \qquad \text{(Rényi Divergence)}$$

$$\tilde{\mu}_{\mathcal{P}/\mathcal{Q}} \equiv \sum_{i=1}^{N} \tilde{w}_{\mathcal{P}/\mathcal{Q}}(x_i) f(x_i). \qquad \text{(SN Estimator)}$$

Of particular importance is the infinite Rényi Divergence, d_∞, which can be rewritten as an almost sure bound on the ratio of $\mathcal{P}$ and $\mathcal{Q}$:

$$d_\infty(\mathcal{P}||\mathcal{Q}) = \operatorname*{ess\,sup}_{x\sim\mathcal{Q}} w_{\mathcal{P}/\mathcal{Q}}(x). \tag{6.9}$$

Assuming $d_\infty(\mathcal{P}||\mathcal{Q})$ is finite, we prove an estimator concentration bound in the following theorem.

Theorem 6.1 (SN d_∞-Concentration Bound)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be two probability measures on the measurable space $(\mathcal{X}, \mathcal{F})$ with $\mathcal{P}$ absolutely continuous w.r.t. $\mathcal{Q}$ and $d_\infty(\mathcal{P}||\mathcal{Q}) < +\infty$. Let $x_1, \dots, x_N$ be independent identically distributed random variables (i.i.d.r.v.) with distribution $\mathcal{Q}$, and $f : \mathcal{X} \to \mathbb{R}$ be a bounded function ($\|f\|_\infty < +\infty$). Then, for any $\lambda > 0$ and N large enough such that $\lambda > \|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})/\sqrt{N}$, the following bound holds with probability at least $1 - 3\exp(-N \cdot t^2(\lambda, N))$:*

$$|\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}}| \leq \lambda, \tag{6.10}$$

$$t(\lambda, N) \equiv \frac{\lambda}{\|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})} - \frac{1}{\sqrt{N}}. \tag{6.11}$$

Theorem 6.1 builds upon the derivation in Proposition D.3 of Metelli *et al.* (2018), which provides a polynomially decaying bound by assuming d_2 is bounded. Here, we compromise by further assuming that d_∞ exists and is bounded to get an exponentially decaying bound. The proof of Theorem 6.1 is given in Appendix A.1, as the theorem remains exactly the same as that of Theorem 5.1, and the intuitive explanation of the d_∞ assumption in the POMDP planning context is given in Section 6.3.2.2.

This exponential decay is important for the proofs in this section. We need to ensure that all nodes of the Sparse Sampling-ω tree at all depths d reach convergence. The branching of the tree induces a factor proportional to N^D. A probabilistic bound that decays exponentially with N will not only help offset the N^D factor even with increasing depths, but also be consistent with Hoeffding-type bound exponential error rate that we also use to bound intermediate estimator errors.

6.3.2.2 Assumptions for Analyzing Sparse Sampling-ω.

The following assumptions are needed for the Sparse Sampling-ω coupled convergence proof:

(i) S and O are continuous spaces, and the action space has a finite number of elements, $|A| < +\infty$.

(ii) For any observation sequence $\{o_{n,j}\}_{n=1}^{d}$, the densities $\mathcal{Z}, \mathcal{T}, b_0$ are chosen such that the Rényi divergence of the target $\mathcal{P}^d$ and sampling distribution $\mathcal{Q}^d$ (Eqs. (6.17) and (6.18)) is bounded above by $d_\infty^{\max}$ for all $d = 0, \ldots, D-1$:

$$d_\infty(\mathcal{P}^d || \mathcal{Q}^d) = \text{ess sup}_{x\sim\mathcal{Q}^d}\, w_{\mathcal{P}^d/\mathcal{Q}^d}(x) \le d_\infty^{\max} \tag{6.12}$$

(iii) The reward function R is bounded by a finite constant $R_{\max}$, and hence the value function is bounded by $V_{\max} \equiv \frac{R_{\max}}{1-\gamma}$.

(iv) We can sample from the generating function G and evaluate the observation density $\mathcal{Z}$.

(v) The POMDP terminates after no more than $D < \infty$ steps.

(vi) We restrict our analysis to all the beliefs $b \in B$ that are realizable from the initial belief b_0 through Bayesian updates with action sequences $\{a_n\}$ and observation sequences $\{o_n\}$.

Intuitively, condition (ii) means that the ratio of the conditional observation probability to the marginal observation probability cannot be too high. Additionally, the results still hold even when either of S or O are discrete, so long as it does not violate condition (ii), by appropriately switching the integrals to sums.

Although our analysis is restricted to the case when $\gamma < 1$ and the problem has a finite horizon, we believe that similar results can be derived for either when $\gamma = 1$ for a finite horizon or for infinite horizon problems when $\gamma < 1$ by using the common argument that eventually future discounted rewards will be small [92, 159]. Furthermore, while the results from this section repeat steps taken in proving POWSS [109], we significantly modify the details for Sparse Sampling-ω.

6.3.2.3 Particle Likelihood Weighting Accuracy.

As a precursor to Theorem 6.2, we establish a general result about function estimation using state particles with likelihood weights. This is useful because the inductive proof for showing Sparse Sampling-ω convergence in Lemma 6.2 relies heavily upon an SN estimator concentration inequality as well as a Hoeffding-type inequality.

Lemma 6.1 (Particle Likelihood SN Estimator Convergence)**.** *Suppose a function f is bounded by a finite constant $\|f\|_\infty \le f_{\max}$, and a particle belief state $\bar{b}_d = \{(s_{d,i}, w_{d,i})\}_{i=1}^{C}$*

at depth d represents b_d with particle likelihood weighting that is recursively updated as $w_{d,i} = w_{d-1,i} \cdot \mathcal{Z}(o_d \mid a, s_d)$. Then, for all $d = 0, \ldots, D-1$, the following weighted average is the SN estimator of f under the belief b_d corresponding to the actions $\{a_n\}_{n=0}^{d-1}$ and observations $\{o_n\}_{n=1}^{d}$, for all beliefs $b_d \in B$ that are realizable given the initial belief b_0:

$$\tilde{\mu}_{\bar{b}_d}[f] = \frac{\sum_{i=1}^{C} w_{d,i} f(s_{d,i})}{\sum_{i=1}^{C} w_{d,i}}, \tag{6.13}$$

and the following concentration bound holds with probability at least $1-3\exp(-C \cdot t_{\max}^2(\lambda, C))$,

$$|\mathbb{E}_{s\sim b_d}[f(s)] - \tilde{\mu}_{\bar{b}_d}[f]| \leq \lambda, \tag{6.14}$$

$$t_{\max}(\lambda, C) \equiv \frac{\lambda}{f_{\max} d_{\infty}^{\max}} - \frac{1}{\sqrt{C}}. \tag{6.15}$$

Proof. We only outline the important steps here, and defer the detailed proof of this lemma to Appendix B.1. The key of this proof lies in the fact that the state particles trajectories $\{s_{n,1}\}, \ldots, \{s_{n,C}\}$ are i.i.d.r.v. sequences of depth d, as GenPF independently generates each state sequence i according to the transition density $\mathcal{T}$. While GenPF generates highly correlated observation sequences and histories $\{o_{n,j}\}_{n=1}^{d}$, the dependence on observation sequence for a given particle belief state only comes through in the particle likelihood weights.

We abbreviate some terms of interest with the following notation:

$$\mathcal{T}_{1:d}^{i} \equiv \prod_{n=1}^{d} \mathcal{T}(s_{n,i} \mid s_{n-1,i}, a_n); \quad \mathcal{Z}_{1:d}^{i,j} \equiv \prod_{n=1}^{d} \mathcal{Z}(o_{n,j} \mid a_n, s_{n,i}), \tag{6.16}$$

where d is the depth, i is the index of the state sample, and j is the index of the observation sample. Absence of indices i, j means that $\{s_n\}$ and/or $\{o_n\}$ appear as regular variables. Intuitively, $\mathcal{T}_{1:d}^{i}$ is the transition density of state sequence i from the root node to depth d, and $\mathcal{Z}_{1:d}^{i,j}$ is the conditional density of observation sequence j given state sequence i from the root node to depth d. Additionally, b_d^i denotes $b_d(s_{d,i})$ and $w_{d,i}$ the weight of $s_{d,i}$.

Then, we apply importance sampling to our system for all depths $d = 0, \ldots, D-1$. Here, $\mathcal{P}^d$ is the normalized measure incorporating the probability of observation sequence $\{o_{n,j}\}_{n=1}^{d}$ on top of the state sequence $\{s_{n,i}\}_{n=1}^{d}$ and action sequence $\{a_n\}_{n=0}^{d-1}$ until the node at depth d, and $\mathcal{Q}^d$ is the measure of the state sequence. We can think of $\mathcal{P}^d$ corresponding to the observation sequence $\{o_{n,j}\}$. For simplicity, we also denote $\mathcal{Z}_{1:d}$ as the product of observation likelihoods $\prod_{n=1}^{d} \mathcal{Z}(o_n \mid a_{n-1}, s_n)$ and $\mathcal{T}_{1:d}$ as the product of transition densities $\prod_{n=1}^{d} \mathcal{T}(s_n \mid s_{n-1}, a_{n-1})$. Then, for an arbitrary action sequence $\{a_n\}$, the following describes the densities necessary to define importance weighting:

$$\mathcal{P}^d = \mathcal{P}^d_{\{a_n, o_{n,j}\}}(\{s_{n,i}\}) = \frac{(\mathcal{Z}_{1:d}^{i,j})(\mathcal{T}_{1:d}^{i}) b_0^i}{\int_{S^{d+1}} (\mathcal{Z}_{1:d}^{j})(\mathcal{T}_{1:d}) b_0 \, ds_{0:d}} \tag{6.17}$$

$$\mathcal{Q}^d = \mathcal{Q}^d_{\{a_n\}}(\{s_{n,i}\}) = (\mathcal{T}_{1:d}^{i}) b_0^i \tag{6.18}$$

$$w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\}) = \frac{(\mathcal{Z}_{1:d}^{i,j})}{\int_{S^{d+1}} (\mathcal{Z}_{1:d}^{j})(\mathcal{T}_{1:d}) b_0 \, ds_{0:d}}. \tag{6.19}$$

Here, the integral to calculate the normalizing constant is taken over S^{d+1}, the Cartesian product of the state space S over $d+1$ steps. Now, we claim that the recursive likelihood updating scheme produces valid likelihood weights up to a normalization. With our recursive definition of the empirical weights, we obtain the full weight of each state sequence for a fixed observation sequence:

$$w_{d,i} = w_{d-1,i} \cdot \mathcal{Z}(o_{d,j} \mid a_d, s_{d,i}) = \ldots = \mathcal{Z}^{i,j}_{1:d} \propto w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\}). \tag{6.20}$$

Consequently, we conclude that the weighted average with particle likelihood weights indeed corresponds to the proper SN estimator:

$$\tilde{\mu}_{\bar{b}_d}[f] = \frac{\sum_{i=1}^{C} w_{d,i} \cdot f(s_{d,i})}{\sum_{i=1}^{C} w_{d,i}} = \frac{\sum_{i=1}^{C} w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\}) \cdot f(s_{d,i})}{\sum_{i=1}^{C} w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\})}. \tag{6.21}$$

We can apply the SN concentration inequality in Theorem 6.1 to obtain the concentration bound. □

Note that proving this lemma allows us to apply the particle likelihood weighting SN inequality whenever we encounter weighted averages with particle likelihood weights for a realizable particle belief. Also, this result does not depend on any specific choice of observation sequence $\{o_n\}$.

6.3.2.4 Coupled Convergence of Sparse Sampling-ω.

The theorem below describes Sparse Sampling-ω's coupled convergence to both optimal Q-values of the POMDP $Q^*_{\mathbf{P}}$ and PB-MDP $Q^*_{\mathbf{M_P}}$, as C is increased.

Theorem 6.2 (Sparse Sampling-ω Coupled Optimality)**.** *Suppose conditions (i)-(vi) are satisfied. Then, for any desired policy optimality $\epsilon > 0$, choosing constants C, λ, δ that satisfy:*

$$\lambda = \epsilon(1-\gamma)^2/8, \tag{6.22}$$

$$\delta = \lambda/(V_{\max} D(1-\gamma)^2), \tag{6.23}$$

$$C = \max\left\{ \left(\frac{4V_{\max} d_{\infty}^{\max}}{\lambda}\right)^2, \frac{64V_{\max}^2}{\lambda^2}\left(D\log\frac{24|A|^{\frac{D+1}{D}} V_{\max}^2 D}{\lambda^2} + \log\frac{1}{\delta}\right)\right\}, \tag{6.24}$$

the Q-function estimates $\hat{Q}^*_{\omega,d}(\bar{b}_d, a)$ obtained for all depths $d = 0, \ldots, D-1$, realized beliefs or histories b_d encountered in the Sparse Sampling-ω tree, and actions a are jointly near-optimal with respect to $Q^*_{\mathbf{P},d}$ and $Q^*_{\mathbf{M_P},d}$ with probability at least $1-\delta$:

$$|Q^*_{\mathbf{P},d}(b_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \leq \frac{\lambda}{1-\gamma}, \tag{6.25}$$

$$|Q^*_{\mathbf{M_P},d}(\bar{b}_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \leq \frac{\lambda}{1-\gamma}. \tag{6.26}$$

To prove Theorem 6.2, we follow a similar proof strategy from our previous proof for POWSS [109] to show that Eq. (6.25) holds, and a similar strategy of the original sparse sampling proof [92] to show that Eq. (6.26) holds. In essence, this Sparse Sampling-ω convergence guarantee builds on POWSS and sparse sampling convergence guarantees, providing coupled convergence results to optimal Q-values of the POMDP $Q^*_{\mathbf{P}}$ and PB-MDP $Q^*_{\mathbf{M_P}}$.

First, we use induction in Lemma 6.2 to prove a concentration inequality for the value function at all nodes in the tree, starting at the leaves and proceeding up to the root. Consequently, proving Lemma 6.2 allows us to prove Theorem 6.2, with some justifications of how the parameter C can actually be explicitly chosen with the choice of ϵ. The detailed proof for Theorem 6.2 is in Appendix B.3.

Lemma 6.2 (Sparse Sampling-ω Estimator Q-Value Coupled Convergence)**.** *For all $d = 0, \dots, D-1$ and a, the following bounds hold with probability at least $1 - 6|A|(4|A|C)^D \exp(-C \cdot \tilde{t}^2)$:*

$$|Q^*_{\mathbf{P},d}(b_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \leq \alpha_d,\ \ \alpha_d = \lambda + \gamma\alpha_{d+1},\ \ \alpha_{D-1} = \lambda, \tag{6.27}$$

$$|Q^*_{\mathbf{M_P},d}(\bar{b}_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \leq \beta_d,\ \ \beta_d = \gamma(\lambda + \beta_{d+1}),\ \ \beta_{D-1} = 0, \tag{6.28}$$

$$t_{\max}(\lambda, C) = \frac{\lambda}{4V_{\max} d^{\max}_{\infty}} - \frac{1}{\sqrt{C}},\ \ \tilde{t} = \min\{t_{\max}, \lambda/4\sqrt{2}V_{\max}\} \tag{6.29}$$

Proof. We outline how we use the particle likelihood SN estimator inequality and Hoeffding inequality to bound the Q-values, and defer the detailed proof to Appendix B.2.

The optimal d-step Q-values for the POMDP $Q^*_{\mathbf{P}}$ and the corresponding PB-MDP $Q^*_{\mathbf{M_P}}$ are

$$Q^*_{\mathbf{P},d}(b_d, a) = \mathbb{E}_{\mathbf{P}}[R(s_d, a) + \gamma V^*_{\mathbf{P},d+1}(b_d a o) \mid b_d] \tag{6.30}$$

$$= \int_S R(s_d, a) b_d \cdot ds_d + \gamma \int_S \int_S \int_O V^*_{\mathbf{P},d+1}(b_d a o)(\mathcal{Z}_{d+1})(\mathcal{T}_{d,d+1}) b_d \cdot ds_{d:d+1} do, \tag{6.31}$$

$$Q^*_{\mathbf{M_P},d}(\bar{b}_d, a) = \rho(\bar{b}, a) + \gamma\, \mathbb{E}_{\mathbf{M_P}}[V^*_{\mathbf{M_P},d+1}(\bar{b}_{d+1}) \mid \bar{b}_d, a] \tag{6.32}$$

$$= \frac{\sum_i w_{d,i} \cdot R(s_{d,i}, a)}{\sum_{i=1}^C w_{d,i}} + \gamma \int_\Sigma V^*_{\mathbf{M_P},d+1}(\bar{b}_{d+1}) \tau(\bar{b}_{d+1} \mid \bar{b}_d, a) d\bar{b}_{d+1}. \tag{6.33}$$

The Sparse Sampling-ω value estimates are mathematically equal to

$$\hat{V}^*_{\omega,d}(\bar{b}_d) = \max_{a \in A} \hat{Q}^*_{\omega,d}\left(\bar{b}_d, a\right) \tag{6.34}$$

$$\hat{Q}^*_{\omega,d}(\bar{b}_d, a) = \frac{\sum_{i=1}^C w_{d,i} r_{d,i}}{\sum_{i=1}^C w_{d,i}} + \frac{1}{C} \sum_{i=1}^{C} \gamma \cdot \hat{V}^*_{\omega,d+1}\left(\bar{b}'^{[I_i]}_{d+1}\right), \tag{6.35}$$

where $\{I_i\}$ are C independent identically distributed random variables with finite discrete distribution $p_{w,d}$ with probability mass $p_{w,d}(I = j) = (w_{d,j} / \sum_k w_{d,k})$, and particle belief state

$\bar{b}'^{[I_i]}_{d+1}$ is updated by an observation generated from s_{d,I_i}. This reflects the fact that GenPF randomly selects a state particle s_o with probability $w_{d,o}/\sum_k w_{d,k}$ C times independently to generate a new observation for the particle belief state after next step.

POMDP Value Convergence: First, we show that Eq. (6.27) is satisfied, which is an adapted and substantially modified proof of POWSS convergence [109]. Using the triangle inequality for a given step d of the inductive proof, we split the difference into two terms, the reward estimation error (A) and the next-step value estimation error (B):

$$|Q^*_{\mathbf{P},d}(b_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \le \underbrace{\left| \mathbb{E}_{\mathbf{P}}[R(s_d, a) \mid b_d] - \frac{\sum_{i=1}^C w_{d,i} r_{d,i}}{\sum_{i=1}^C w_{d,i}} \right|}_{\text{(A)}} \tag{6.36}$$
$$+ \gamma \underbrace{\left| \mathbb{E}_{\mathbf{P}}[V^*_{\mathbf{P},d+1}(b_d a o) \mid b_d] - \frac{1}{C}\sum_{i=1}^C \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1}) \right|}_{\text{(B)}}$$

The reward estimation error (A) is exactly the particle likelihood importance sampling error for estimating the reward function $R(\cdot, a)$, which can be bounded by applying Lemma 6.1. This also proves the base case.

To bound the next-step value estimation error (B), we introduce particle likelihood SN estimators and Monte Carlo average estimators to bridge the following quantities (detailed definitions and bounds of each terms are in Appendix B.3):

$$\underbrace{\left| \mathbb{E}_{\mathbf{P}}[V^*_{\mathbf{P},d+1}(b_d a o) \mid b_d] - \frac{1}{C}\sum_{i=1}^C \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1}) \right|}_{\text{(B)}} \le \tag{6.37}$$
$$\underbrace{\left| \mathbb{E}_{\mathbf{P}}[V^*_{\mathbf{P},d+1}(b_d a o) \mid b_d] - \frac{\sum_{i=1}^C w_{d,i} \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[i]}}{\sum_{i=1}^C w_{d,i}} \right|}_{\text{(1) Importance sampling error}} + \underbrace{\left| \frac{\sum_{i=1}^C w_{d,i} \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[i]}}{\sum_{i=1}^C w_{d,i}} - \frac{1}{C}\sum_{i=1}^C \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[I_i]} \right|}_{\text{(2) MC weighted sum approximation error}}$$
$$+ \underbrace{\left| \frac{1}{C}\sum_{i=1}^C \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[I_i]} - \frac{1}{C}\sum_{i=1}^C V^*_{\mathbf{P},d+1}(b_d a o^{[I_i]}) \right|}_{\text{(3) MC next-step integral approximation error}} + \underbrace{\left| \frac{1}{C}\sum_{i=1}^C V^*_{\mathbf{P},d+1}(b_d a o^{[I_i]}) - \frac{1}{C}\sum_{i=1}^C \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1}) \right|}_{\text{(4) Inductive function estimation error}}.$$

PB-MDP Value Convergence: Second, we show that Eq. (6.28) is satisfied, which is an adapted and substantially modified proof of sparse sampling convergence [92]. Once again, we split the difference between the SN estimator and the $Q^*_{\mathbf{M_P}}$ function into two terms, the reward estimation error (A) and the next-step value estimation error (B):

$$|Q^*_{\mathbf{M_P},d}(\bar{b}_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \leq \underbrace{\left|\rho(\bar{b}_d, a) - \rho(\bar{b}_d, a)\right|}_{(\mathrm{A})\,=\,0} \tag{6.38}$$
$$+ \gamma \underbrace{\left|\mathbb{E}_{\mathbf{M_P}}[V^*_{\mathbf{M_P},d+1}(\bar{b}_{d+1}) \mid \bar{b}_d, a] - \frac{1}{C}\sum_{i=1}^{C} \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1})\right|}_{(\mathrm{B})}$$

Since our particle belief MDP induces no reward estimation error, the term (A) is always 0 and proving the base case $d = D - 1$ is trivial as (A) and (B) are both 0. Then, we show that the difference (B) is bounded for all $d = 0, \dots, D - 1$. We use the triangle inequality repeatedly to separate it into two terms; (1) the MC transition approximation error, and (2) the inductive function estimation error (detailed definitions and bounds of each terms are in Appendix B.3):

$$\underbrace{\left|\mathbb{E}_{\mathbf{M_P}}[V^*_{\mathbf{M_P},d+1}(\bar{b}_{d+1}) \mid \bar{b}_d, a] - \frac{1}{C}\sum_{i=1}^{C} \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1})\right|}_{(\mathrm{B})} \tag{6.39}$$
$$\leq \underbrace{\left|\mathbb{E}_{\mathbf{M_P}}[V^*_{\mathbf{M_P},d+1}(\bar{b}_{d+1}) \mid \bar{b}_d, a] - \frac{1}{C}\sum_{i=1}^{C} V^*_{\mathbf{M_P},d+1}(\bar{b}'^{[I_i]}_{d+1})\right|}_{(1)\text{ MC transition approximation error}} + \underbrace{\left|\frac{1}{C}\sum_{i=1}^{C} V^*_{\mathbf{M_P},d+1}(\bar{b}'^{[I_i]}_{d+1}) - \frac{1}{C}\sum_{i=1}^{C} \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1})\right|}_{(2)\text{ Inductive function estimation error}}.$$

Combining the probability bounds used in both of these procedures results in a worst case $\mathcal{O}(\exp(-t \cdot C))$ probability factor, where t is some constant, as both the SN concentration bound and the Hoeffding bound are exponentially decaying. Since this upper bound on the estimation error needs to hold for all steps $d = 0, \dots, D - 1$, we must apply the worst case union bound on the probability to ensure that every node in the tree achieves the desired concentration bound. This results in a worst case probability factor that is $\mathcal{O}(C^D)$. Therefore, we can obtain the Q-value estimator concentration inequality, with convergence rate $\mathcal{O}(C^D \exp(-t \cdot C))$. □

6.4 Particle Belief MDP Approximation Guarantees

In this section, we establish the theoretical guarantees for using any approximately optimal MDP planning algorithm to solve the POMDP problem $\mathbf{P}$ by planning in the particle belief MDP $\mathbf{M_P}$. Theorem 6.3 shows that the Q-values $Q^*_{\mathbf{P}}$ and $Q^*_{\mathbf{M_P}}$ are close to each other with high probability, and Theorem 6.4 shows that using any approximately optimal MDP planning algorithm $\mathcal{A}$ in the particle belief MDP $\mathbf{M_P}$ as a policy yields near-optimal value in the original POMDP with an additional closed-loop exact belief update step.

6.4.1 Particle Belief MDP Q-Value Approximation Optimality

We introduce Theorem 6.3, which probabilistically bridges the POMDP $\mathbf{P}$ and its corresponding particle belief MDP $\mathbf{M_P}$. In essence, this theorem claims that the two optimal Q-values $Q^*_{\mathbf{P}}$ and $Q^*_{\mathbf{M_P}}$ are close with high probability, because creating a very accurate Q-value estimator via Sparse Sampling-ω that is close to both $Q^*_{\mathbf{P}}$ and $Q^*_{\mathbf{M_P}}$ happens with high probability.

Theorem 6.3 (Particle Belief MDP Q-Value Approximation Optimality). *Given a finite horizon POMDP $\mathbf{P}$ and its corresponding particle belief MDP $\mathbf{M_P}$, there exists a number of particles C for which the optimal Q-value of the POMDP problem $Q^*_{\mathbf{P}}(b,a)$ can be approximated by the optimal Q-value of the particle belief MDP problem $Q^*_{\mathbf{M_P}}(\bar{b},a)$ with arbitrary precision. Namely, under the regularity conditions (i)-(vi), the following bound holds for a given realizable belief b, corresponding sampled particle belief $\bar{b}$, and all available actions a with probability at least $1-\delta_{\mathbf{M_P}}$ for a desired accuracy $\epsilon_{\mathbf{M_P}}$:*

$$|Q^*_{\mathbf{P}}(b,a) - Q^*_{\mathbf{M_P}}(\bar{b},a)| \leq \epsilon_{\mathbf{M_P}}. \tag{6.40}$$

Proof. The main idea of the proof is that we bridge the two Q-values, $Q^*_{\mathbf{P}}$ and $Q^*_{\mathbf{M_P}}$, via approximation through Sparse Sampling-ω with C particles. From Theorem 6.2, we have established that there exists an algorithm, Sparse Sampling-ω, which is jointly optimal in both senses of POMDP $\mathbf{P}$ and its corresponding particle belief MDP $\mathbf{M_P}$. Then, if we were to hypothetically perform Sparse Sampling-ω of depth D, the sum of the errors between the three types of Q-values at the root node, $Q^*_{\mathbf{P}}$, $Q^*_{\mathbf{M_P}}$ and $\hat{Q}^*_{\omega}$, are jointly bounded with probability at least $1-\delta_{\mathbf{M_P}}$ through Theorem 6.2, where $\delta_{\mathbf{M_P}}$ follows the same definition as δ. We use the fact that $Q^*_{\mathbf{P}}$ and $Q^*_{\mathbf{M_P}}$ are the optimal Q-values at $d=0$ for the POMDP and PB-MDP, respectively:

$$\begin{aligned} |Q^*_{\mathbf{P}}(b,a) - Q^*_{\mathbf{M_P}}(\bar{b},a)| &\leq |Q^*_{\mathbf{P}}(b,a) - \hat{Q}^*_{\omega}(\bar{b},a)| + |\hat{Q}^*_{\omega}(\bar{b},a) - Q^*_{\mathbf{M_P}}(\bar{b},a)| && (6.41)\\ &\equiv |Q^*_{\mathbf{P},0}(b_0,a) - \hat{Q}^*_{\omega,0}(\bar{b}_0,a)| + |Q^*_{\mathbf{M_P},0}(\bar{b}_0,a) - \hat{Q}^*_{\omega,0}(\bar{b}_0,a)| && (6.42)\\ &\leq \frac{2\lambda}{1-\gamma} \quad (\equiv \epsilon_{\mathbf{M_P}}). && (6.43)\end{aligned}$$

Here, instead of using the λ definition in Theorem 6.2, we can directly choose λ such that the above quantity is equal to $\epsilon_{\mathbf{M_P}}$, since λ only depends on a desired approximation error ϵ and γ. However, we emphasize the intermediate step of the Q-value difference bounded by $\frac{2\lambda}{1-\gamma}$ to invoke this fact later on in the policy convergence proof of Theorem 6.4. Since this bound holds with high probability for creating any hypothetical Sparse Sampling-ω tree, this must mean that $|Q^*_{\mathbf{P}}(b,a) - Q^*_{\mathbf{M_P}}(\bar{b},a)| \leq \epsilon_{\mathbf{M_P}}$ in general with high probability.

The convergence rate $\delta_{\mathbf{M_P}}$ is $\mathcal{O}(C^D \exp(-\tilde{t}\cdot C))$. This means that as we increase the number of particles, we can expect better performance by approximately solving a POMDP via particle belief MDP planning approximation. □

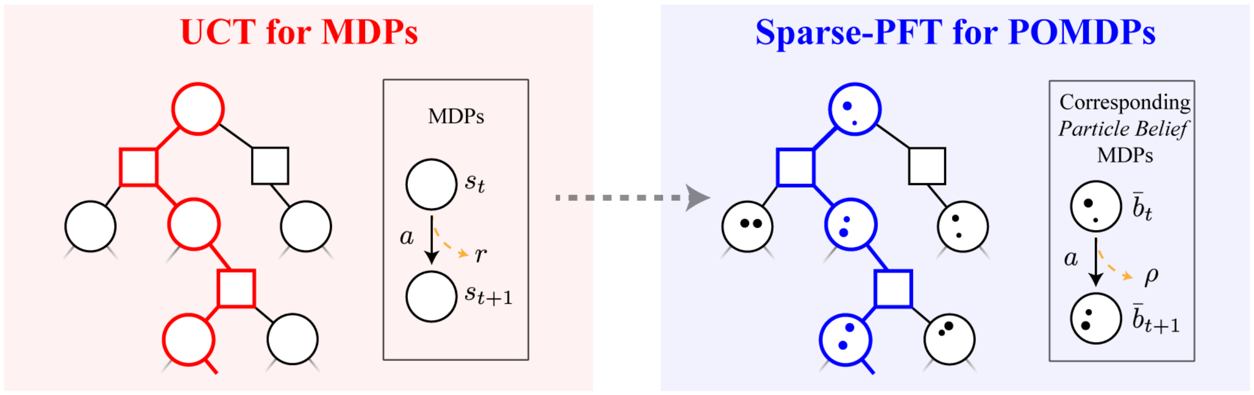


Figure 6.2: **Illustration of promoting an MDP algorithm (UCT) into a POMDP algorithm (Sparse-PFT).** This practically only involves changing the transition generative model into a particle filtering-based transition generative model that deals with particle belief states.

6.4.2 Particle Belief MDP Planning Optimality

Corollary 6.1 (Particle Belief MDP Planning Optimality)**.** *Under regularity conditions necessary for both the particle belief MDP and an MDP planning algorithm $\mathcal{A}$, if the optimal planner can approximate Q-values with arbitrary precision $\epsilon_{\mathcal{A}}$ with probability at least $1-\delta_{\mathcal{A}}$ in the corresponding particle belief MDP of a given POMDP, then the planning algorithm can approximate the POMDP Q-values within $\epsilon_{\mathbf{M_P}}+\epsilon_{\mathcal{A}}$ with probability at least $1-\delta_{\mathbf{M_P}}-\delta_{\mathcal{A}}$:*

$$|Q^*_{\mathbf{P}}(b,a) - \hat{Q}^{\mathcal{A}}_{\mathbf{M_P}}(\bar{b},a)| \leq \epsilon_{\mathbf{M_P}} + \epsilon_{\mathcal{A}}. \tag{6.44}$$

Proof. This is a straightforward application of triangle inequality for the Q-value estimation accuracy and worst case union bound for the probability. □

Note that it would also be possible to devise an expected value version of the bounds by converting the probability statement into an expected value statement.

Essentially, Corollary 6.1 means that we can use any approximately optimal MDP planning algorithm to solve the POMDP problem by planning in the particle belief MDP instead, and still retain similar optimality guarantees. In most practical cases, this method would usually incur an additional $\mathcal{O}(C)$ compute time factor for a transition sampling step as single particle belief state generation now needs to propagate C particles forward instead of a single particle/state.

This remarkable result does not directly depend on the size of the state space nor the observation space. However, the dependence may indirectly come through the observation density and thus the Rényi divergence factor, and in practice, the generative model sampling complexity that often depends on the dimensionality of the state space. Moreover, even though this approach is insensitive to the state and observation space size, the guarantees and practical algorithms are highly sensitive to the planning horizon D.

Algorithm 6.3 Sparse-PFT Algorithm

```
Global Variables: $\gamma, n, c_{\text{UCB}}, G, D$.
Algorithm: SIMULATE($\bar{b}, d$)
Input: particle belief set $\bar{b} = \{(s_i, w_i)\}$, depth $d$.
Output: A scalar $q$ that is the total discounted reward of one simulated trajectory sample.
 1: if $d = D$ then
 2:     return 0
 3: $a \leftarrow \arg\max_{a \in C(\bar{b})} Q(\bar{b}, a) + c_{\text{UCB}} \sqrt{\frac{\log N(\bar{b})}{N(\bar{b}, a)}}$
 4: if $|C(\bar{b}, a)| = C$ then
 5:     $\bar{b}', \rho \leftarrow$ sample from $C(\bar{b}, a)$
 6: else
 7:     $\bar{b}', \rho \leftarrow$ GENPF($\bar{b}, a$)
 8:     $C(\bar{b}, a) \leftarrow C(\bar{b}, a) \cup \{(\bar{b}', \rho)\}$
 9: if $N(\bar{b}) = 0$ then
10:     $q \leftarrow \rho + \gamma \cdot$ ROLLOUT($\bar{b}', d - 1$)
11: else
12:     $q \leftarrow \rho + \gamma \cdot$ SIMULATE($\bar{b}', d - 1$)
13: $N(\bar{b}) \leftarrow N(\bar{b}) + 1$
14: $N(\bar{b}, a) \leftarrow N(\bar{b}, a) + 1$
15: $Q(\bar{b}, a) \leftarrow Q(\bar{b}, a) + \frac{q - Q(\bar{b}, a)}{N(b, a)}$
16: return $q$
```

Consequently, proving Corollary 6.1 allows us to prove Theorem 6.4 with additional results from Kearns *et al.* (2002) and Singh and Yee (1994). Through the near-optimality of the Q-functions, we conclude that the value obtained by employing a near-optimal MDP policy in the PB-MDP is also near-optimal in the original POMDP with further assumptions on the closed-loop POMDP system. The detailed proof for Theorem 6.4 is in Appendix B.4.

Theorem 6.4 (Particle Belief MDP Approximate Policy Convergence)**.** *In addition to regularity conditions for particle belief MDP and the MDP planning algorithm $\mathcal{A}$, assume that the closed-loop POMDP Bayesian belief update step is exact. Then, for any $\epsilon > 0$, we can choose a C such that the value obtained by planning with $\mathcal{A}$ in the PB-MDP is within ϵ of the optimal POMDP value function at b_0:*

$$V^*_{\mathbf{P}}(b_0) - V^{\mathcal{A}}_{\mathbf{M_P}}(b_0) \leq \epsilon. \tag{6.45}$$

6.4.3 Sparse Particle Filter Tree (Sparse-PFT)

By utilizing the results in Theorem 6.3 and Theorem 6.4, we can promote a sampling-based MDP planning algorithm Upper Confidence Tree (UCT) into Sparse Particle Filter Tree (Sparse-PFT) and retain similar convergence guarantees for the POMDP [21, 98, 157]. This results in an algorithm that is simple to implement, and enjoys both theoretical guarantees and high performance in practice.

The core of Sparse-PFT is the SIMULATE function defined in Algorithm 6.3 that is called repeatedly to construct the tree. The set of global variables for Sparse-PFT includes the same global variables used for Sparse Sampling-ω with the addition of n, the number of tree search queries, and c_{UCB}, the Upper Confidence Bound critical factor that determines the amount of exploration. The SIMULATE function is analogous to the function of the same name from UCT [21, 98], with the only difference being Sparse-PFT manages particle belief sets through GENPF rather than states directly.

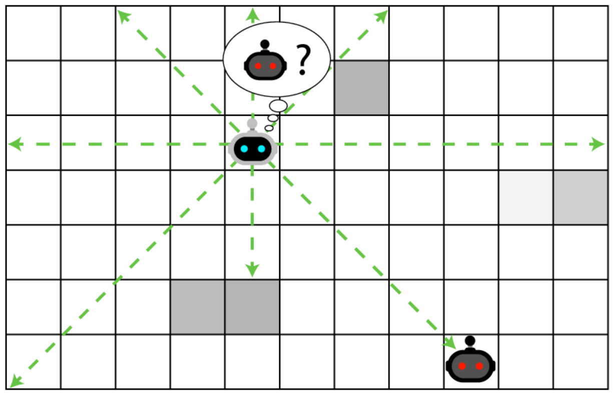
In the above algorithm definition, $C(\cdot)$ represents the list of children nodes, $N(\cdot)$ the number of visits to the node, $Q(\cdot)$ the estimated Q-value at the node, and c_{UCB} the Upper Confidence Bound exploration parameter. These lists are all implicitly initialized to 0 or $\emptyset$. The ROLLOUT procedure is an optional heuristic that runs a simulation with a heuristic rollout policy for d steps to estimate the value, while avoiding building a large computation tree at each step of simulation.

With the introduction of Sparse-PFT, we can view the recent POMDP algorithms as practical extensions of Sparse-PFT. For instance, PFT-DPW [171] is a simple modification of Sparse-PFT by utilizing the double progressive widening (DPW) technique to additionally handle continuous action spaces, and POMCPOW [171] is a further extension that plans based on particle trajectory that allows for flexible particle number representations of a given belief node. However, further theoretical analyses of these algorithms would most likely require more sophisticated techniques and further assumptions.

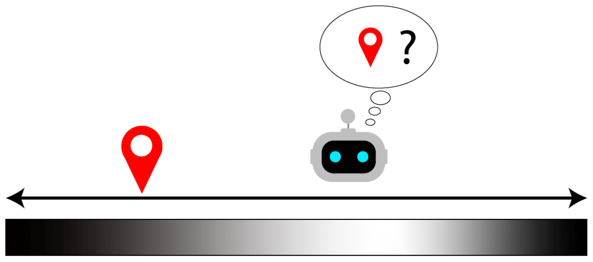
6.5 Numerical Experiments

Numerical simulation experiments were conducted in order to evaluate and compare the performances of our new simple algorithm, Sparse-PFT, along with other solvers. In particular, we also ran experiments for Adaptive online packing-guided search (AdaOPS) [187], a recent solver with practical performance and partial theoretical guarantees. We also show performances of other hallmark algorithms like QMDP and POMCP along with random policy to demonstrate the need for continuous observation POMDP solvers that can handle more general assumptions.

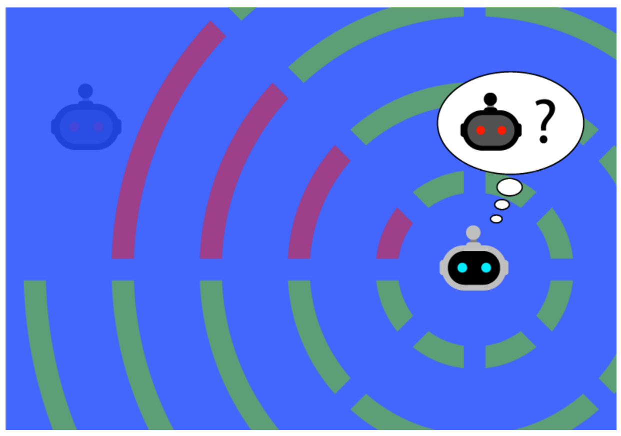
The following sections contain descriptions of the evaluation problems along with discussion of solver performances. In all five of the numerical experiments shown in Fig. 6.3, the POMDP solvers were limited to at most 1 second of planning time per step. For the closed loop planning, the belief updates were accomplished with a particle filter independent of the planner, and no part of the planning tree was saved for re-use on subsequent steps. A total of 5000 simulation experiments were conducted for each configuration combination of solver and environment in order to obtain the Monte Carlo mean and standard error estimates for the Laser Tag and VDP tag environments, and 1000 simulation experiments for the Light Dark and Sub Hunt problems since planners typically yielded more consistent performances for these problems. The tabular summary of all results is given in Table 6.1, and corresponding figure summary of all results for different planning time allotments is given in Fig. 6.4.



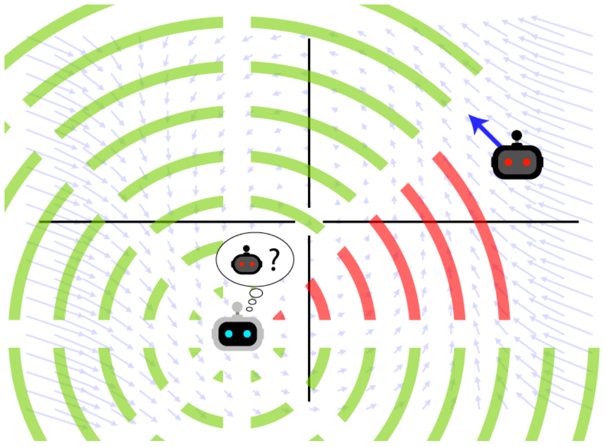
(a) Laser Tag POMDP



(b) Light Dark POMDP



(c) Sub Hunt POMDP



(d) VDP Tag POMDP

Figure 6.3: **Illustration of the four environments we test our continuous observation POMDP algorithms on**: (a) Laser Tag, (b) Light Dark, (c) Sub Hunt, and (d) VDP Tag (discrete and continuous versions).

We also vary belief particle count C with SIMULATE calls held constant to demonstrate the effect of particle belief approximation resolution on the quality of the resulting policy in Fig. 6.10 by using the optimized hyperparameters from Table B.1 with 1000 simulation experiments for Laser Tag, Light Dark and Sub Hunt, and 100 for VDP Tag and Discrete VDP Tag. The open source code for the experiments is built on the POMDPs.jl framework [53], and is available at: github.com/WhiffleFish/PFTExperiments. The hyperparameter values used for the experiments are shown in Appendix B.5.

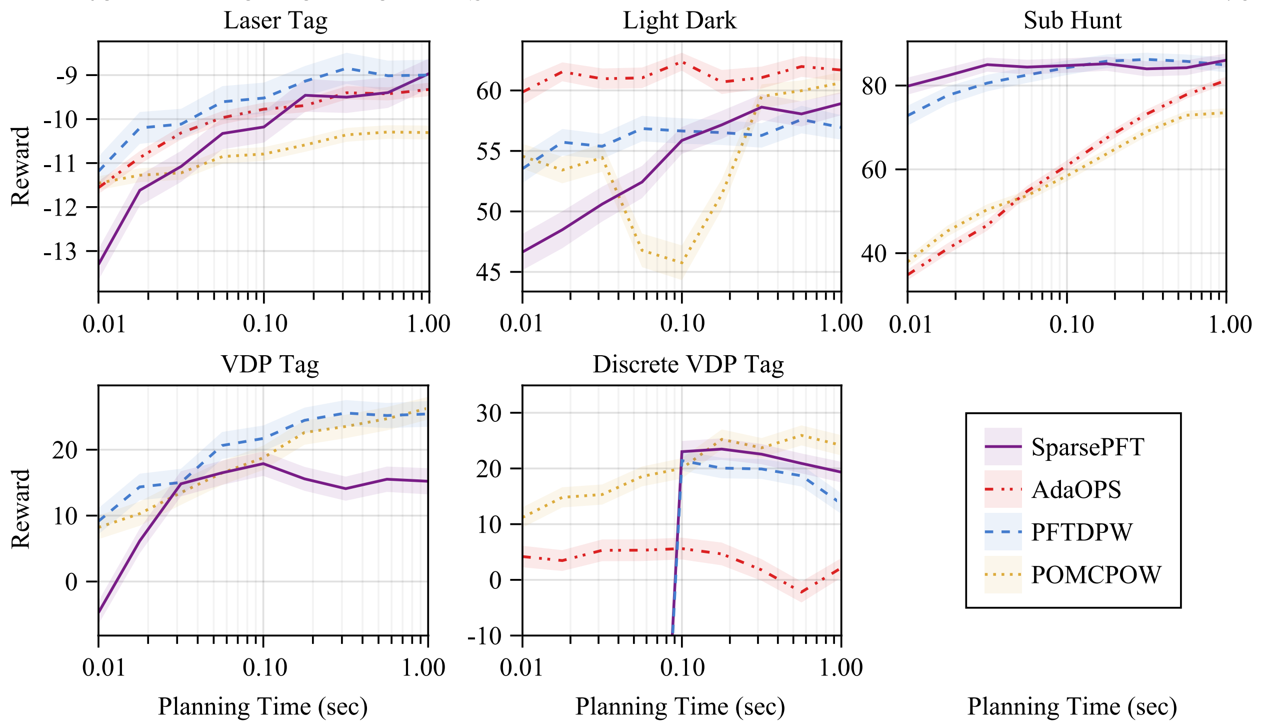


Figure 6.4: **Comparative benchmark results summary of continuous observation POMDP solvers over different planning time allotments.** Ribbons surrounding the Monte Carlo benchmark mean estimates indicate two standard error confidence bands. Planning times are given in log scale.

	Laser Tag (D, D, D)		Light Dark (D, D, C)		Sub Hunt (D, D, C)	
Sparse-PFT	**-8.96 ± 0.17**	🟩	58.9 ± 0.5	🟩	**86.0 ± 0.8**	🟩
PFT-DPW	**-8.99 ± 0.07**	🟩	56.9 ± 0.5	🟩	84.9 ± 0.9	🟩
POMCPOW	-10.3 ± 0.08	🟩	60.6 ± 0.4	🟩	73.5 ± 0.7	🟩
AdaOPS	-9.33 ± 0.08	🟩	**61.7 ± 0.5**	🟩	81.3 ± 0.6	🟩
QMDP	-10.4 ± 0.08	🟩	3.28 ± 0.5	🟥	28.0 ± 0.6	🟥
POMCP	-16.0 ± 0.09	🟥	-14.86 ± 2.3	🟥	30.0 ± 0.8	🟥
Random Policy	-51.0 ± 0.18		-85.0 ± 0.72		4.20 ± 0.27	

	VDP Tag (C, C, C)		VDP Tag^D (C, D, C)	
Sparse-PFT	15.2 ± 1.0	🟩	19.3 ± 0.9	🟩
PFT-DPW	**25.4 ± 1.0**	🟩	13.7 ± 0.9	🟩
POMCPOW	**26.3 ± 0.9**	🟩	**24.2 ± 0.9**	🟩
AdaOPS			2.0 ± 0.4	🟥
Random Policy	-66.8 ± 0.24		-66.6 ± 0.25	

Table 6.1: **Tabular comparative benchmark summary of continuous observation POMDP solvers.** All results are given as (mean $\pm$ stderr) for a 1 second planning time allotment. The algorithm(s) with the best average performance within one standard error is shown in boldface for each experiment. The three letters after each problem name indicate whether the state, action, and observation spaces are continuous or discrete, respectively.

6.5.1 Laser Tag

The Laser Tag POMDP is taken from the DESPOT benchmarks [191] wherein a robot is required to use laser sensors to localize with the ultimate goal of catching an evading robot. The agent's laser sensors extend radially in 8 evenly spaced directions and each return a rounded sensed distance sampled from a normal distribution given by $\mathcal{N}(d, 2.5)$ where d is the true distance to the nearest obstacle. Although the observation space is not continuous, it is sufficiently large (on the order of 10^6) that most online solvers would have to treat this as close to continuous.

From the results, we find that the PFT methods outperform both POMCPOW and AdaOPS: PFT-DPW consistently outperforms both planners across different planning times, and Sparse-PFT outperforms all other planners with increased planning time. This suggests that for Laser Tag, having a full particle belief approximation rather than dynamically varying particle size is helpful for keeping track of likely particle hypotheses. Furthermore, this also suggests that the double progressive widening has diminishing returns when the action space has a fixed small size.

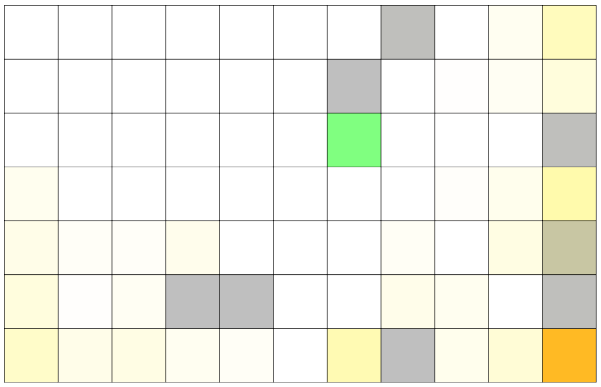


Figure 6.5: **Example Sparse-PFT operating in Laser Tag**: the agent (green) has roughly located the evading robot at bottom right corner (orange).

We note that POMCP particularly struggles on this problem compared to all other algorithms. The large observation space forces the trees constructed by POMCP to become extremely shallow due to each unique sampled observation resulting in a new leaf node. This hinders POMCP's ability to develop a non-myopic multi-step plan and yield accurate action values, empirically showing the importance of particle weighting. On the other hand, QMDP performs similarly well as the modern solvers. While the agent does not initially know its own location, it has sufficient information to localize using the laser sensor observations after some steps, and the evading robot behavior is not stochastic. Thus, the crude QMDP approximation performs well, seemingly due to the problem requiring less active information gathering compared to other problems.

6.5.2 Light Dark

The 1-dimensional Light Dark POMDP revolves around the requirement of active information gathering. The state is an integer representing the position of the agent and the action space is $\mathcal{A} = \{-10, -1, 0, 1, 10\}$. Deterministic transitions are given by $s' = s + a$. The reward,

$$R(s,a) = \begin{cases} +100 & \text{if } s = 0, a = 0 \\ -100 & \text{if } s \neq 0, a = 0 \\ -1 & \text{otherwise} \end{cases}, \tag{6.46}$$

dictates that the optimal policy drive the state to the origin as quickly as possible. Because the state is not immediately known, inferences over the true state must be made over noisy observations that grow in variance proportional to the agent's distance from the light location at $s = 10$. The observation distribution is $\mathcal{N}(s, |s - 10| + \epsilon)$, where ϵ is some small constant included to prevent observation weights from reaching $+\infty$ due to a collapse to a Dirac distribution when the agent arrives at the light location.

The planners that yield the highest expected reward in the Light Dark domain roughly follow a 2-step plan: first localizing at the light location, then traveling down to the goal location. Essentially, the light location becomes a necessary subgoal. We can demonstrate this by creating a heuristic policy that initially steers towards the light region via certainty-equivalent control, and then takes action $a = -10$ down to the goal. This heuristic policy yields an expected reward of 62.0 ± 0.19 which is as good or better than any planner shown in Table 6.1.

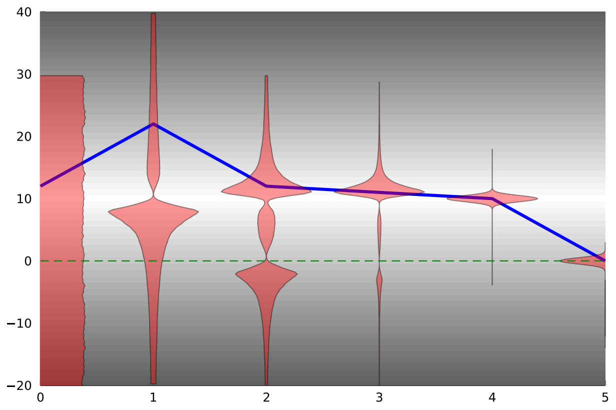


Figure 6.6: **Example Sparse-PFT trajectory for Light Dark**, successfully localizing in the light region to reach the goal in the dark region.

Surprisingly, higher planning times do not necessarily correspond to increasing expected rewards in the Light Dark domain. For AdaOPS, the solver converges to its peak expected reward with a planning time as low as 0.01 seconds leading to marginal improvement with further increases in planning time. Within a planning time interval of $[0.03, 0.1]$ seconds, the performance of POMCPOW decreases, indicating that the planner becomes increasingly confident in a suboptimal plan. One possible source of this overconfidence is beliefs represented by a single particle. This same behavior becomes evident in PFT planners when the PFT planner is supplied with a single rollout value estimation. However, by increasing the number of sampled particles that are chosen as the true state in belief-based rollouts, the belief value estimate is granted lower variance and greater accuracy, effectively reducing the time spent exploring suboptimal branches of the constructed tree.

Because of this necessity to decrease belief entropy before committing to the goal, QMDP performs suboptimally due to no emphasis being placed on costly information gathering. Therefore, regardless of belief distribution entropy, QMDP myopically steers directly towards the goal location but rarely commits within the simulation horizon due to high state uncertainty.

6.5.3 Sub Hunt

In the Sub Hunt POMDP, from the POMCPOW benchmark [171], the agent controls a submarine with the goal of finding and destroying an opposing submarine. The state space consists of the grid locations of both the agent and enemy submarines, a Boolean determining whether or not the enemy is aware of the agent's presence, and the enemy's goal direction ($\{1, \ldots, 20\}^4 \times \{\text{aware, unaware}\} \times \{N, S, E, W\}$). The agent is given the option to move three steps in any of the four cardinal directions, attack the enemy, or ping the enemy with active sonar while the enemy randomly chooses between taking two steps forward or one step diagonally forward.

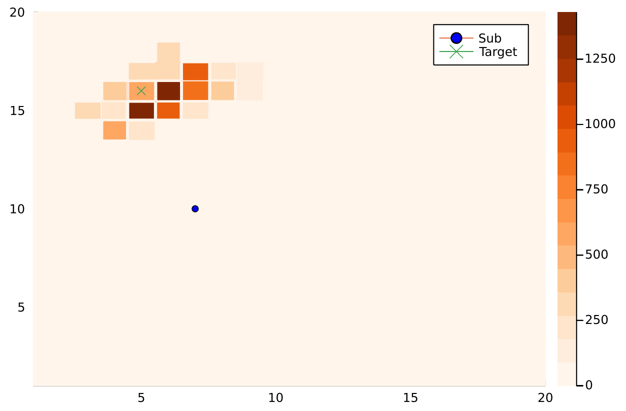


Figure 6.7: **Example Sparse-PFT policy behavior for Sub Hunt**, where the agent's submarine pursues the target with active information gathering.

In the Sub Hunt domain, PFT methods dominate all other planners over all planning times, with Sparse-PFT having a slight edge over all other planners. Because the state space is discrete, value iteration can be used to calculate Q-values for the fully observable MDP, and QMDP [113] can be used for the rollout policy. With this strong belief-based rollout policy, both PFT-DPW and Sparse-PFT are able to construct nearly-optimal policies with planning times as low as 0.01 seconds, leading to no noticeable further improvement over longer planning times. Conversely, POMCPOW and AdaOPS have gradually increasing planning curves in Fig. 6.4, with AdaOPS nearly reaching the performance of PFT planners at 1 second and POMCPOW reaching an earlier inflection point, resulting in a final performance lower than the other three planners.

6.5.4 VDP Tag

The Van Der Pol Tag (VDP Tag) POMDP formulation tasks the agent with moving through a two-dimensional space to catch an opponent whose dynamics are governed by the Van Der Pol differential equations,

$$\dot{x} = \mu\left(x - \frac{x^3}{3} - y\right), \quad \dot{y} = \frac{x}{\mu}, \tag{6.47}$$

for which we use scaling constant $\mu = 2$. Because this problem has a continuous state space ($\mathcal{S} = \mathbb{R}^4$), a continuous action space ($\mathcal{A} = [0, 2\pi) \times \{0, 1\}$) and a continuous observation space

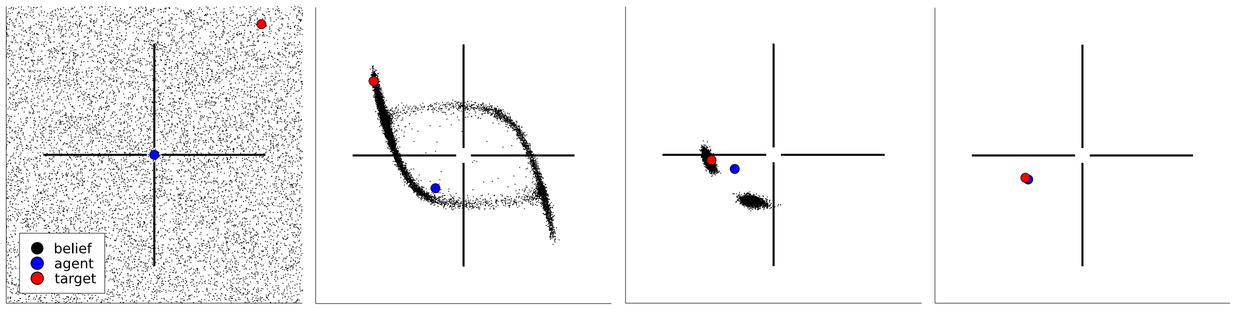


Figure 6.8: **Example Sparse-PFT policy behavior for VDP Tag**, which shows a successful localization of the target following the Van Der Pol dynamics, then a successful capture.

($\mathcal{O} = \mathbb{R}^8$), discrete value iteration is no longer admissible as input for a value estimation policy. AdaOPS is unable to handle continuous action spaces thus it is omitted from this benchmark. For the action-discretized VDP Tag, the available movement directions are 20 evenly spaced angles from 0° to 360°.

The continuous VDP Tag domain is the first in which there exists a noticeable performance gap between Sparse-PFT and PFT-DPW, indicating that action progressive widening offers some utility over fixed widening in continuous action space problems. Furthermore, PFT-DPW and POMCPOW have similar performances across different planning times, suggesting that the main challenge of VDP Tag is being able to handle continuity of state, action, and observation spaces, while the particle belief approximation resolution does not affect the performance as much.

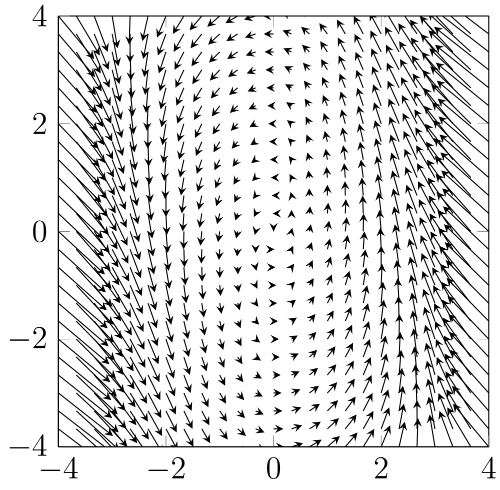


Figure 6.9: **An example Van Der Pol vector field** ($\mu = 0.5$) **which defines the target dynamics.** The target is not blocked by the barriers, unlike the agent.

For discrete VDP Tag, we come across two new peculiarities: AdaOPS performance decreases with increased planning time, and PFT methods perform orders of magnitude worse than other planners at very low allotted planning times. This is likely attributable to a large action space ($|\mathcal{A}| = 20$) and a relatively expensive simulation function (RK4 integration). Because PFT methods propagate a collection of particles upon tree expansion, belief value estimates tend to be more accurate at the cost of added computation scaling linearly with the number of particles representing each belief. Thus, expanding all possible actions while using an computationally expensive simulator on all particles takes a long time, leading to very shallow trees.

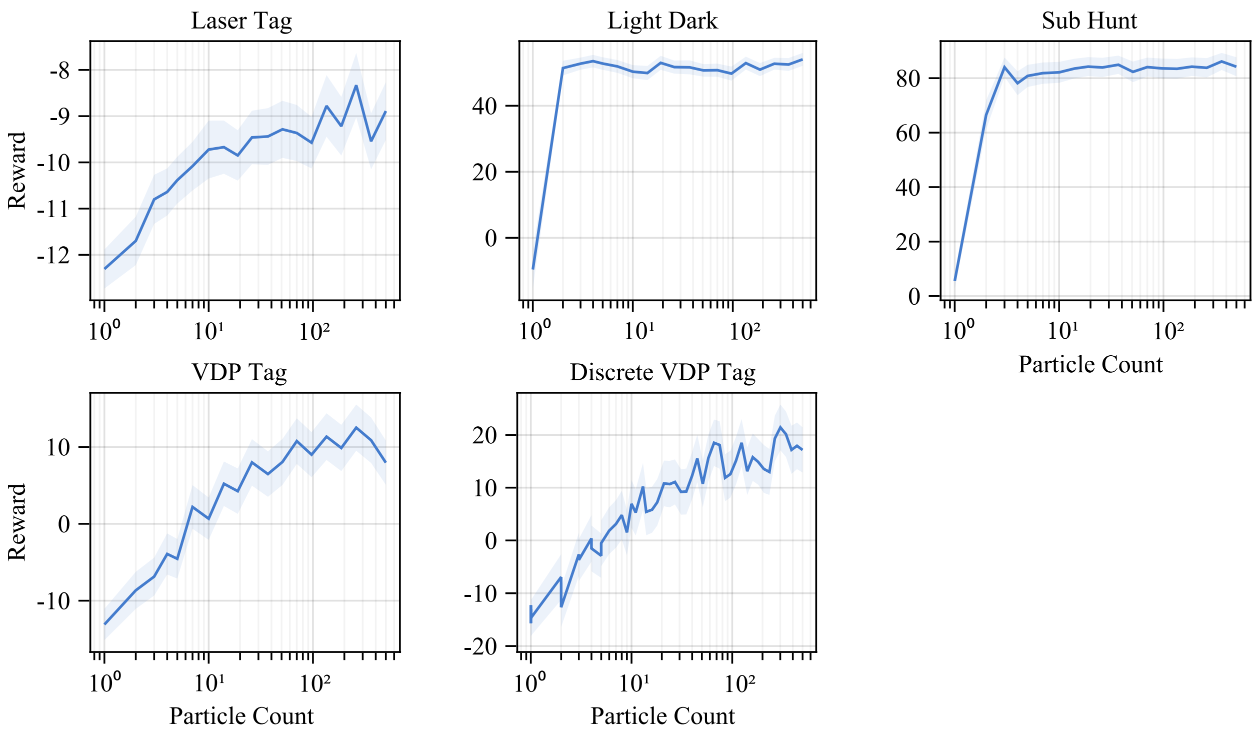


Figure 6.10: **Empirical validation of the effect of increasing particle counts to better planning performance.** Sparse-PFT policy mean rewards with two standard error confidence band, varying belief particle count while holding number of tree search SIMULATE queries constant.

6.5.5 Experimental Validation of Particle Belief Approximation Convergence

In order to test the effect of particle belief approximation resolution, or the number of belief particles C, on planner performance, we vary C while fixing the number of SIMULATE calls and using optimal hyperparameters found in Appendix B.5 for Sparse-PFT planner. By increasing the the number of belief particles, the particle belief becomes a more accurate representation of the actual belief function, and should lead to a better optimal Q-value estimation.

Across all five problem domains, increasing the number of particles C results in a roughly monotonic non-decreasing performance gain as shown in Fig. 6.10. In particular, we see a gradual performance increase for Laser Tag, VDP Tag, and Discrete VDP Tag as we increase the number of particles, while Light Dark and Sub Hunt problems reach their performance capacity rather quickly at less than 10 particles. However, when applying this principle to promote MDP algorithms into PB-MDP algorithms, the particle filtering transition generative model takes an extra $\mathcal{O}(C)$ multiplicative factor for generating transitions, so it is important to balance compute time and value estimation accuracy when deploying these algorithms in practice.

These results offer two valuable insights. First, the number of particles C offers increased performance since the increased resolution of particle belief approximation corresponds to increased accuracy in Q-value estimates as we have established in Section 6.4. This suggests that the resolution of belief approximation is one indicator of a problem difficulty. Second, not all problems benefit the same way from increasing the number of particles. Light Dark and Sub Hunt have rather simple state spaces, and adding more particles did not significantly improve the policy performance. In contrast, the other three problems continually benefited from having increased resolution of belief approximation. Consequently, this suggests that the belief approximation resolution is not the only factor contributing to the problem difficulty, but also other factors like action space cardinality and existence of optimal rollout policies will contribute to the problem difficulty.

6.6 Conclusion

In this work, we formally show that optimality guarantees in a finite sample particle belief MDP (PB-MDP) approximation of a POMDP/belief MDP yields optimality guarantees in the original POMDP as well, which allows for simple yet powerful adaptations of MDP algorithms to solve POMDPs. By proving that the Sparse Sampling-ω Q-value estimates are close to both optimal Q-values of the POMDP and PB-MDP with high probability, we conclude that the optimal Q-values of the POMDP and PB-MDP themselves are close with high probability. This fundamental bridge between PB-MDPs and POMDPs allows us to adapt any sampling-based MDP algorithm of choice to a POMDP by solving the corresponding particle belief MDP approximation and preserve the convergence guarantees in the POMDP. Such operation only increases the computational complexity of transition generation by a factor of $\mathcal{O}(C)$, by using particle filtering-based generative models. Our convergence result is not directly dependent on the size of the state space nor the observation space, but rather dependent on the Rényi divergence that links the probabilities concerning state and observation trajectories. This motivates usage of particle belief-based MDP algorithms such as Sparse Particle Filter Tree (Sparse-PFT), which enjoys algorithmic simplicity, theoretical guarantees and practicality, as it is a variant of upper confidence trees (UCT) [21, 157] for PB-MDPs.

There are many interesting avenues for future research. First, the broader theoretical justification of more complex algorithms, such as POMCPOW and DESPOT-α, still do not exist. Showing theoretical validity of these algorithms would help close the gap between theory and practice even further. In addition, as seen in our numerical experiments, the best performing algorithm varies across different types of benchmarks. Further theoretical and empirical characterization of which algorithms are most effective for which problems could greatly aid practitioners. Also, the particle number sweep suggests a method to characterize the difficulty of a POMDP problem. While there have been many empirical studies of different benchmarks, there are currently no known analytic quantification of the problem difficulty, which may be of interest for both practitioners and researchers alike. Lastly, while the algorithms presented here perform well in low dimensional continuous observation spaces, tree search for more difficult POMDPs, such as those with high dimensional observations [48] and continuous/hybrid action spaces [110, 124, 153] is more difficult, and further analytical and empirical research is warranted.

Chapter 7

Voronoi Progressive Widening

Chapter Outline

This chapter is based on the paper "Voronoi Progressive Widening: Efficient Online Solvers for Continuous State, Action, and Observation POMDPs" [110], written in collaboration with Claire Tomlin and Zachary Sunberg.

7.1 Introduction

The partially observable Markov decision process (POMDP) is a flexible mathematical framework for expressing stochastic sequential decision problems. POMDPs can represent a wide range of real world problems such as autonomous driving [15, 172], cancer screening [13], spoken dialog systems [192], and aircraft collision avoidance [77]. A POMDP is an optimization problem in which we aim to find a policy that maps states to actions which will control the state to maximize the expected sum of rewards. Finding an optimal POMDP policy is computationally demanding because of the uncertainty introduced by imperfect observations [139]. One of the most popular approaches to deal with this computational challenge is to use *online* algorithms that look for local approximate policies as the agent interacts with the environment rather than computing a global policy that maps every possible outcome to an action. Many online POMDP algorithms are variants of Monte Carlo tree search (MCTS) [28, 159, 171] or other tree search variants [101, 191].

The research presented here concerns *continuous space POMDPs*, defined as POMDPs with continuous state, action and observation spaces. If tree search is naively applied to continuous space POMDPs, an immediate problem is that the branching factor of the tree will be infinite, preventing evaluation of future consequences of actions deep in the tree. This problem has been thoroughly studied, and the most popular solutions are sparse sampling [61, 92, 109] and progressive widening (PW) [43, 171], which limit the branching factor by only considering a randomly-selected subset of the states, actions and observations. For continuous states and observations, this random sampling is used to approximate the expectation

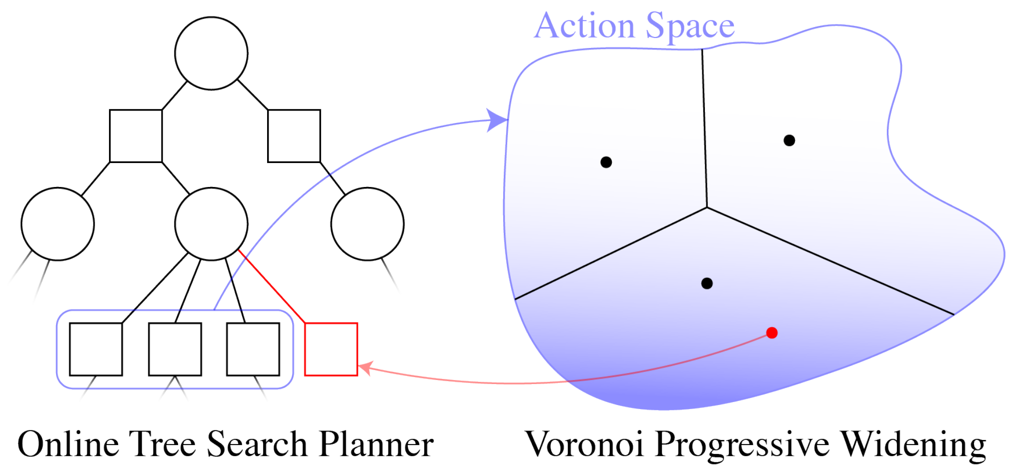


Figure 7.1: **Voronoi progressive widening (VPW) guides the selection of actions to widen the search tree.** Circles denote state/belief nodes, squares denote action nodes.

of the next-step value in Bellman's equation [61, 92, 109]. Since the Monte Carlo integration needed to approximate these expectations has straightforward and robust convergence guarantees, sparse sampling and PW are sufficient [43, 92, 109].

Continuous action spaces are, however, much more difficult to accommodate. Instead of simply estimating expectations, planning in a problem with a continuous action space requires solving a nonconvex optimization problem at each node in the tree. PW has been applied to these problems [43, 171], but, since PW considers a randomly sampled set of actions, it wastes a large amount of computation on suboptimal parts of the action space and has only been demonstrated on small action spaces such as one-dimensional intervals of real numbers. Voronoi optimistic optimization (VOO) [94] is an approach for sequential decision problems with deterministic and fully-observable dynamics that attempts to focus computation on more promising parts of the action space by partitioning it into Voronoi cells and sampling from cells corresponding to previously successful actions.

In this work, we propose Voronoi Progressive Widening (VPW), a versatile technique to modify tree search algorithms to effectively handle continuous or hybrid action spaces. VPW generalizes VOO and PW to problems with both transition and observation uncertainties. Like PW, it balances exploring previously proposed actions and searching for new action candidates. However, VPW searches for new candidates in a much more efficient way via VOO, and balances local and global searching without relying on any additional prior information or expert knowledge. Furthermore, VPW does not require significant additional computation time and can handle both continuous and hybrid action spaces with relative ease.

There are two main contributions in this work. First, we prove theoretical guarantees about certain parts of the VPW approach via Voronoi Optimistic Weighted Sparse Sampling (VOWSS). VOWSS is special case of VPW applied to Partially Observable Weighted Sparse

Solver	Problem	Cont.	Opt.	Fast	Brief Description
VOOT [94]	MDP	S, A	✓	✓	VOO for deterministic MDPs.
POWSS [109]	POMDP	S, O	✓	✗	Sparse sampling with observation weights.
VOWSS	POMDP	S, A, O	✓	✗	Extends POWSS with VPW.
POMCPOW [171]	POMDP	S, A, O	✗	✓	Combines POMCP and DPW.
VOMCPOW	POMDP	S, A, O	✗	✓	Extends POMCPOW with VPW.
BOMCP [124]	POMDP	S, A, O	✗	✓	Extends POMCPOW with Gaussian processes.
VOSS	MDP	S, A	✓	✗	Extends sparse sampling with VPW.

Table 7.1: **Summary of continuous space MDP and POMDP solvers studied and newly proposed in this chapter.** The newly proposed algorithms in this chapter are marked in bold. "Problem" column denotes which problems the solver can handle, "Cont." column denotes which spaces are continuous, "Opt." denotes if the solver has optimality guarantees, and "Fast" denotes if the solver is fast enough to be practical.

Sampling (POWSS) [109] with fixed number of action samples, which integrates the two convergence guarantees from VOO and POWSS into a single guarantee. Specifically, this is the first known solver to have global convergence guarantees for continuous space POMDPs. This shows that VOO and action PW were extended in a way that VPW not only can handle transition and observation uncertainties, but also combines the two procedures in a theoretically sound way.

Second, we propose an efficient VPW-based algorithm: Voronoi Optimistic Monte Carlo Planning with Observation Weighting (VOMCPOW). VOMCPOW can also be thought of as a practical extension of VOWSS. The experiments over different domains show the practical effectiveness of VPW for handling continuous and hybrid action space problems. VOMCPOW consistently outperforms state-of-the-art continuous space POMDP solvers based on PW, such as Partially Observable Monte Carlo Planning with Observation Widening (POMCPOW) [171] and Bayesian Optimized Monte Carlo Planning (BOMCP) [124] in several simulation experiments. Table 7.1 summarizes the solvers that are studied in this chapter.

The remainder of this chapter proceeds as follows: First, Section 7.2 introduces the VPW algorithm, and describes its strengths as well as how to efficiently implement it in practice. Section 7.3 presents the VOWSS algorithm and theoretical analysis justifying VPW. Finally, Section 7.4 empirically shows the optimality of VOWSS action selection and the efficiency and robustness of VOMCPOW over three different simulation experiments.

7.2 Voronoi Progressive Widening

In this section, we first introduce VOO and PW to motivate the formulation of VPW and VOMCPOW.

Algorithm 7.1 VOO Algorithms [94]

```
Global Variables: $A, D(\cdot,\cdot), \omega, \Sigma$.
Algorithm: VOO($\mathbf{a}, \mathbf{Q}$)
Input: Array of Voronoi centers $\mathbf{a} = [a_i]$ and their function values $\mathbf{Q} = [Q(a_i)]$.
Output: A sample $a$.
1: $u \leftarrow \text{Unif}[0,1]$
2: if $u \leq \omega$ or $|\mathbf{Q}| = 0$ then
3:     $a \leftarrow \text{Unif}(A)$
4: else
5:     $a \leftarrow$ BestVoronoiCell($\mathbf{a}, \mathbf{Q}$)
6: return $a$
```

```
Algorithm: BestVoronoiCell($\mathbf{a}, \mathbf{Q}$)
Input: Array of Voronoi centers $\mathbf{a} = [a_i]$ and their function values $\mathbf{Q} = [Q(a_i)]$.
Output: A sample $a$.
1: $a^* \leftarrow \arg\max_{\mathbf{a}} \mathbf{Q}$
2: while $a$ is not closest do
3:     $a \leftarrow N(a^*, \Sigma)$
4:     Check if $D(a, a^*) \leq D(a, a_i)$ $\forall a_i \in \mathbf{a}$, $a_i \neq a^*$
5: return $a$
```

7.2.1 Voronoi Optimistic Optimization

Voronoi optimistic optimization (VOO) [94] is a continuous multi-armed bandit algorithm that adaptively explores the sampling space by partitioning the space into Voronoi cells. We define a Voronoi cell by the set of points closest to the corresponding Voronoi center compared to the other centers, and the best Voronoi cell is the Voronoi cell with the center that achieves the highest function value estimate. In our setting, the sampling space is the action space, the Voronoi centers are the sampled actions, and the function values are the Q-value estimates at the actions. Since Voronoi cells are solely defined by distances to Voronoi centers, VOO additionally takes in a distance metric $D(\cdot,\cdot)$ as an input, and scales well to higher dimensions unlike HOO.

In Algorithm 7.1, we define the adapted functions VOO and BestVoronoiCell. VOO searches the action space either uniformly with probability ω or from the best Voronoi cell with probability $1-\omega$. BestVoronoiCell samples an action from the best Voronoi cell via rejection sampling. In practice, VOO sampling is best implemented by using Gaussian rejection sampling centered around the current best action [94]. While we reflected this in our algorithm definition, we note that the theoretical guarantees only hold for uniform rejection sampling. Furthermore, to improve computation time for the experiments in Section 7.4, we limit the maximum number of rejected samples via methods described in Appendix B. VOO has previously been extended to Voronoi Optimistic Optimization Tree (VOOT) [94], but only for deterministic MDPs.

7.2.2 Progressive Widening

Progressive widening (PW) [43] tackles continuous state and action spaces by gradually expanding the state and action sample sets. PW limits the number of sampled children to $k \cdot N^{\alpha}$, where N corresponds to the number of visits to the parent node in the search tree, and k, α are the widening parameters. Algorithm 7.2 describes action progressive widening.

Algorithm 7.2 Action Progressive Widening [43]

```
Global Variables: $k_a, \alpha_a, A, c$.
Algorithm: PW($h$)
Input: Belief/history node $h$ in the MCTS tree.
Output: An action $a$.
1: if $|C(h)| = 0$ or $|C(h)| \leq k_a N(h)^{\alpha_a}$ then
2:     $a \leftarrow \text{Unif}(A)$
3:     $C(h) \leftarrow C(h) \cup \{a\}$
4: else
5:     $a \leftarrow \arg\max_{a \in C(h)} Q(h,a) + c\sqrt{\frac{\log N(h)}{N(h,a)}}$
6: return $a$
```

Algorithm 7.3 Voronoi Progressive Widening

```
Global Variables: $k_a, \alpha_a, \omega, A, c$.
Algorithm: VPW($h$)
Input: Belief/history node $h$ in the MCTS tree.
Output: An action $a$.
1: if $|C(h)| = 0$ or $|C(h)| \leq k_a N(h)^{\alpha_a}$ then
2:     $a \leftarrow \text{VOO}(C(h), [Q(h,\cdot)])$
3:     $C(h) \leftarrow C(h) \cup \{a\}$
4: else
5:     $a \leftarrow \arg\max_{a \in C(h)} Q(h,a) + c\sqrt{\frac{\log N(h)}{N(h,a)}}$
6: return $a$
```

Here, h is the belief/history node in the MCTS tree, $C(h)$ the list of children action nodes, N the number of visits, and c the Upper Confidence Bound exploration parameter. In the action space, PW balances two modes of exploration: (1) exploring branches with previously proposed actions (line 2), and (2) searching for new suitable action candidates (line 5). However, since PW samples new actions uniformly from the action space, it is rather sample inefficient and does not take into account any information gained from previous samples.

7.2.3 Voronoi Progressive Widening

We now introduce Voronoi Progressive Widening (VPW) in Algorithm 7.3, which generalizes VOO and PW. Essentially, VPW progressively widens with a VOO sample instead of a uniform sample. With this formulation, PW is simply VPW with $\omega = 1$, and VOO is VPW with $k_a \cdot N^{\alpha_a} = +\infty$.

Compared to PW, VPW only additionally relies on having a collection of Q-value estimates, which are typically calculated and stored for MCTS solvers. While this means that we require the Q-value estimates to be relatively faithful to the actual Q-values and also requires an informative rollout policy that can guide the solver to more optimal actions, this is not a requirement specific to VPW as a typical MCTS solver should aim to have both of these components. Furthermore, since the VOO step does not require a significant additional time, VPW operates in a similar time scale as PW/DPW while being able to efficiently sample action candidates that are closer to the optimal action. The only other requirement of VPW is the distance metric on the action space, but this is often simple to define, and in fact lets VPW to tackle hybrid action spaces such as the one in Section 7.4.2.

7.2.4 VOMCPOW

The VPW technique is versatile as it can be applied to any MCTS solver for MDPs and POMDPs without requiring any additional expert or domain knowledge. For instance, one could consider Sparse-UCT-VPW algorithm for continuous space MDPs that uses VPW criterion with the Sparse-UCT algorithm [21] to handle continuous action spaces. Specifically, we demonstrate the versatility of VPW through modifying POMCPOW into VOMCPOW (Voronoi Optimistic Monte Carlo Planning with Observation Weighting) by simply swapping out action PW with VPW. VOMCPOW consistently outperforms POMCPOW and BOMCP by a statistically significant margin across all of our experiments.

7.3 Convergence Analysis

For convergence analysis, we introduce and Voronoi Optimistic Weighted Sparse Sampling (VOWSS), a VPW-based continuous space online POMDP solver that guarantees convergence to an optimal policy. VOWSS justifies the usage of VPW-based solvers that build upon sparse sampling [92], particle weighting and VOO, such as VOMCPOW.

7.3.1 VOWSS

We define EstimateV and EstimateQ functions in Algorithm 7.4. Here, C_s, C_a are the state and action widths, respectively. EstimateV is a subroutine that returns the value function, V, for an estimated state or belief, by calling EstimateQ for each action and returning the maximum. Similarly, EstimateQ performs sampling and recursive calls to EstimateV to estimate the Q-function at a given step with a weighted average in the style of POWSS [109]. The sampled states s'_i are inserted into each next-step belief particle set $\overline{bao_j}$ with the new weights $w'_i = w_i \cdot \mathcal{Z}(o_j|a, s'_i)$, which are the adjusted probability of hypothetically sampling observation o_j from state s'_i.

Thus, the VOWSS policy action is obtained by calling EstimateV$(\bar{b}_0, 0)$ at the root node and taking the action that corresponds to the maximizing Q-value. Note that the particle belief set is initialized by drawing samples from b_0 and normalizing weights to $1/C_s$. Like POWSS, VOWSS is not very computationally efficient as it fully expands the sparsely sampled tree. Thus, VOWSS mainly demonstrates theoretical convergence and are only

Algorithm 7.4 Value estimation algorithms for VOWSS

```
Global Variables: $\gamma, G, C_s, C_a, D$.
Algorithm: EstimateV$(\bar{b}, d)$
Input: Belief particle set $\bar{b} = \{(s_i, w_i)\}$, depth $d$.
Output: A scalar $\hat{V}^\star_d(\bar{b})$ that is an estimate of $V^\star_d(\bar{b})$.
1: If $d \geq D$ the max depth, then return 0.
2: For $i = 1, \cdots, C_a$, sequentially run VPW$(\bar{b})$
   to sample actions $a_i$ and estimate the
   corresponding $Q$-values with $\hat{Q}^\star_d(\bar{b}, a_i)$ =
   EstimateQ$(\bar{b}, a, d)$.
3: Return the value function:
   $$\hat{V}^\star_d(\bar{b}) = \max_{i=1,\cdots,C_a} \hat{Q}^\star_d(\bar{b}, a_i).$$

Algorithm: EstimateQ$(\bar{b}, a, d)$
Input: Belief particle set $\bar{b} = \{(s_i, w_i)\}$, action $a$, depth $d$.
Output: A scalar $\hat{Q}^\star_d(\bar{b}, a)$ that is an estimate of $Q^\star_d(b, a)$.
1: For each particle-weight pair $(s_i, w_i)$ in $\bar{b}$,
   generate $s'_i, o_i, r_i$ from $G(s_i, a)$.
2: For each observation $o_j$ from previous step,
   iterate over $i = 1, \cdots, C_s$ to insert $(s'_i, w_i \cdot
   \mathcal{Z}(o_j|a, s'_i))$ to a new belief particle set $\overline{bao_j}$.
3: Return the $Q$-value estimate:
   $$\hat{Q}^\star_d(\bar{b}, a) = \frac{\sum_{i=1}^{C_s} w_i(r_i + \gamma \text{EstimateV}(\overline{bao_i}, d+1))}{\sum_{i=1}^{C_s} w_i}.$$
```

practically applicable to very small problems. The algorithms in this section are written in a style closer to a plain English format to enhance the readability of mathematical machinery used for these algorithms, since VOWSS is more theoretical in nature.

It is worth noting that with our recursive formulation, VOOT [94] can be recast as a Q-function estimation algorithm that simply returns the one sample Q-function estimate. Particularly, this means that EstimateQ for VOOT would look identical to VOWSS's EstimateQ function except with C_s set to 1 as VOOT assumes no transition and observation uncertainties, with no action width decay. Consequently, VOOT is notably faster than VOWSS and thus practical for deterministic MDP problems, since it only needs to sample the deterministic next step state once. Nevertheless, due to the hierarchical structure of VOOT, the actual VOO algorithm as well as the theoretical guarantees of VOOT should remain identical when recast into this recursive form. We also discuss the stochastic MDP case in Appendix C.

7.3.2 Convergence Guarantees

We obtain global convergence to the optimal value functions for continuous space POMDPs by combining VOO that globally optimizes over the entire action space and POWSS that estimates value functions with arbitrary precision. To our knowledge, VOWSS is the first continuous space POMDP algorithm to have global convergence guarantees without relying on any discretization schemes for any of the spaces. In the subsequent sections, we prove that VOWSS policy can be made to perform arbitrarily close to the optimal policy by increasing both the state and action widths. Our results are derived with $k_a \cdot N^{\alpha_a} = +\infty$ (VPW = VOO), and uniform rejection sampling for BestVoronoiCell.

The convergence proof of VOWSS relies on the proofs for the ancestor algorithms: POWSS and VOO. Since the proof of VOWSS requires the union of all the regularity conditions for POWSS and VOO, which deal with local smoothness of rewards and observation density requirements, we will not repeat them here and detail them in Appendix A.2. One of the conditions requires that the reward function R be bounded and measurable, $||R||_\infty \leq R_{\max}$, and consequently the value is bounded by $V_{\max} \equiv R_{\max}/(1-\gamma)$ for $\gamma < 1$.

Theorem 7.1 (VOWSS Inequality)**.** *Suppose we choose the action sampling width C_a and state sampling width C_s such that under the union of regularity conditions specified by [109] and [94], the intermediate POWSS bounds and VOO bounds in Lemma 7.1 are satisfied at every depth of the tree. Then, the following bounds for the VOWSS estimator $\hat{V}_{\text{VOWSS},d}(\bar{b})$ hold for all $d \in [0, D-1]$ in expectation:*

$$|V^\star_d(b) - \hat{V}_{\text{VOWSS},d}(\bar{b})| \leq \eta_{C_a} + \alpha_{C_s}. \tag{7.1}$$

Here, η_{C_a} and α_{C_s} are non-negative bounds that tend to 0 as we increase C_a and C_s, respectively. We show a brief outline of the proof of Theorem 7.1. In order to prove Theorem 7.1, we first need to prove the intermediate bounds.

Lemma 7.1 (VOWSS Intermediate Inequality)**.** *Suppose with our notation, the POWSS estimators at all depths d are within ϵ of their mean values with probability $1-p$, and the VOO agents have regret bounds of $\mathcal{R}_{C_a}$. The following inequalities hold for all $d \in [0, D-1]$ in expectation:*

$$|V^\star_d(b) - \hat{V}^{C_a}_d(b)| \leq \eta_{C_a}(d), \tag{7.2}$$

$$|\hat{V}^{C_a}_d(b) - \hat{V}_{\text{VOWSS},d}(\bar{b})| \leq \alpha_{C_s}(d). \tag{7.3}$$

In Lemma 7.1, we aim to bound the inequality in (7.1) by applying triangle inequality to the two split terms: the VOO-like regret bound with $\eta_{C_a}(d)$ and the POWSS-like concentration bound with $\alpha_{C_s}(d)$. The exact definitions of these intermediate bound terms are defined and explained further in Appendix A.2. We define each value function used in Lemma 7.1 as the following:

$$\begin{aligned}
V^\star_d(b) &\equiv \max_{a\in A}\; R(b,a) + \gamma\,\mathbb{E}[V^\star_{d+1}(bao)|b], \\
\hat{V}^{C_a}_d(b) &\equiv \max_{a\in VOO(A)} R(b,a) + \gamma\,\mathbb{E}[\hat{V}^{C_a}_{d+1}(bao)|b], \\
\hat{V}_{\text{VOWSS},d}(\bar{b}) &\equiv \max_{a\in VOO(A)} \frac{\sum_{i=1}^{C_s} w_i(r_i + \gamma\cdot\hat{V}_{\text{VOWSS},d+1}(\overline{bao_i}))}{\sum_{i=1}^{C_s} w_i}.
\end{aligned} \tag{7.4}$$

Here, $V^\star_d(b)$ is the optimal value function as per the conventional definition, and $\hat{V}_{\text{VOWSS},d}(\bar{b})$ is the VOWSS estimator for the optimal value function. In addition, we introduce the theoretical intermediate term $\hat{V}^{C_a}_d(b)$ that bridges the gap between the regret bound and the concentration bound by only performing the VOO step and not the sparse sampling step. Using this intermediate term, we can further subdivide the intermediate inequalities to obtain the appropriate regret and concentration bounds using results from [94] and [109], respectively. Proving Lemma 7.1 proves Theorem 7.1.

7.4 Experiments

The numerical experiments in this section confirm the theoretical results of Section 7.3.2, as well as showcase the effectiveness of VPW. We demonstrate the performances of our algorithms in three different control experiments: Linear-Quadratic-Gaussian (LQG) control, Van Der Pol Tag, and modified lunar lander problems. When running experiments for POMCPOW, VOMCPOW, and BOMCP, we use the rollout policy heuristic to estimate the value function as well as take the first action to be the rollout policy action at each newly generated belief node. To determine the hyperparameters of the solvers, we used cross-entropy method (CEM) [116] to maximize the mean expected reward of a hyperparameter set when the optimal hyperparameters are not already available from previous experiments. The VOMCPOW and BOMCP hyperparameters were trained by first initializing them with POMCPOW hyperparameters and then using CEM again including all the additional hyperparameters specific to the solvers. The only hyperparameter that was manually chosen is the Gaussian covariance matrix for VOO rejection sampling. The code for the experiments is built on the POMDPs.jl framework [53]. The exact hyperparameters used for each experiment are given in Appendix B.

7.4.1 Linear-Quadratic-Gaussian Control

We first test our algorithms on a simple 2D-action space LQG control system. Here, we choose a relatively simple problem setup such that we can easily understand and visualize the results while allowing VOWSS to still plan within a reasonable time. Here, VOWSS uses VPW like VOO by setting $k_a \cdot N^{\alpha_a} = +\infty$, but with Gaussian rejection sampling. The dynamics and observation models are

$$x_{t+1} = x_t + u_t + v_t; \quad y_t = x_t + w_t; \quad v_t, w_t \overset{i.i.d.}{\sim} N(0, \sigma^2 I). \tag{7.5}$$

The initial state x_0 is distributed as $N([-10, 10], \sigma^2 I)$, and $\sigma = 0.1$ for all x_0, v_t, w_t. We aim to minimize the cost function $J(x_0)$, while planning for two steps ($N = 2$):

$$J(x_0) = \mathbb{E}[x_N^T x_N + \sum_{t=0}^{N-1} (x_i^T x_i + u_i^T u_i)]. \tag{7.6}$$

The analytical answer can be obtained by solving the LQG backup equation, which is identical to the LQR solution,

$$u_0^\star = -K_0 \hat{x}_0 = -0.6 \cdot \hat{x}_0 = [6.0, -6.0]. \tag{7.7}$$

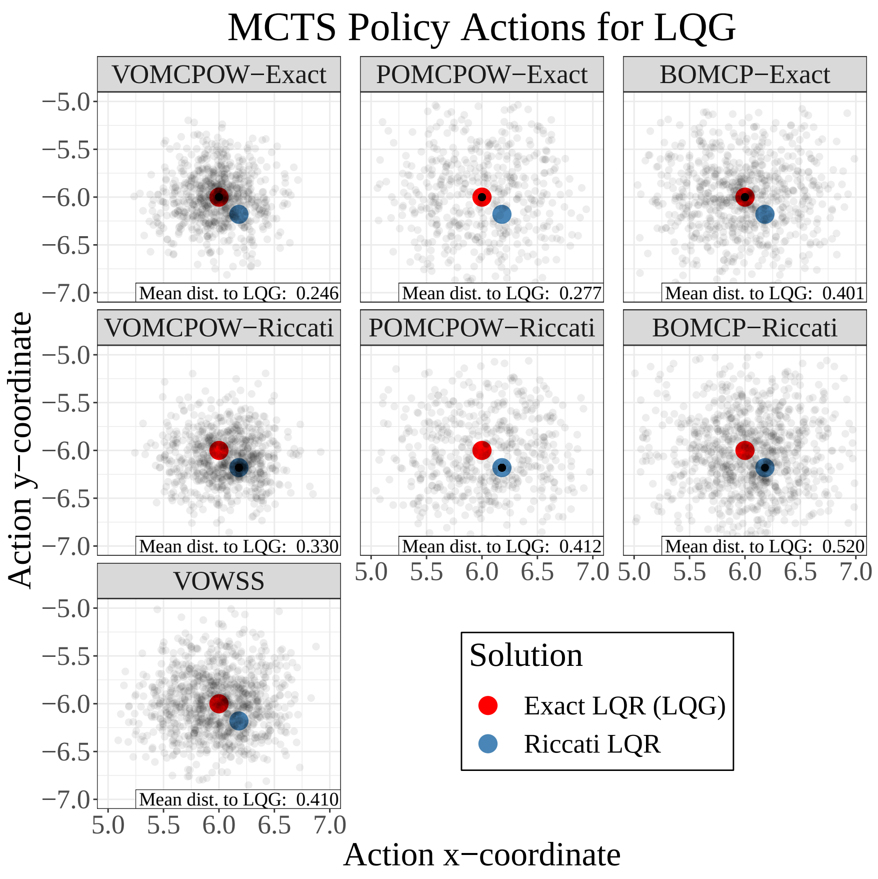
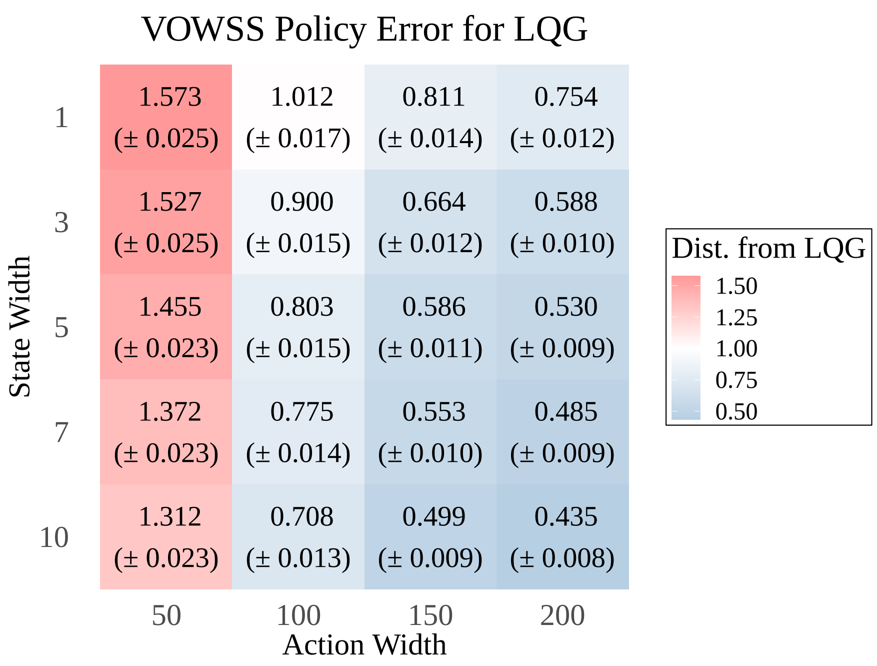


Figure 7.2: **Scatter plots of the first actions chosen by policies from different solvers and rollouts for the two step LQG problem.** The exact LQR solution, which is the same as the LQG solution, is shown as a red dot at $[6.0, -6.0]$, and the Riccati solution as a blue dot at approximately $[6.18, -6.18]$.

Since the problem has an analytical solution, we can directly compare the actions each solver chooses to the analytical solution, for all of VOWSS, POMCPOW, VOMCPOW, and BOMCP. For the solvers that require a rollout policy, we test both the finite horizon LQR solution (referred to as "exact policy") and the steady-state solution to the discrete time algebraic Riccati equation (referred to as "Riccati policy") [37] as the rollout policies to observe effects on each solver. For this particular scenario, the Riccati solution is $u_t^* \approx -0.618 \cdot \hat{x}_t$, and $u_0^* \approx [6.18, -6.18]$, slightly different from the exact policy.

When choosing hyperparameters through CEM, we use the Riccati policy as the rollout policy. The number of queries for POMCPOW and VOMCPOW are set to 1000 and BOMCP to 100, which correspond to approximately 0.1 seconds of planning time. For VOWSS, the state width is set to 10 and action width to 200. Furthermore, we introduce the action width decay term γ_a such that the number of VPW iterations for a given height is γ_a times the VPW iterations for the previous height, similar to the way VOOT is modified for long


VOWSS Policy Error for LQG
State Width
Action Width
Dist. from LQG
1.50
1.25
1.00
0.75
0.50

State Width \ Action Width	50	100	150	200
1	1.573 (± 0.025)	1.012 (± 0.017)	0.811 (± 0.014)	0.754 (± 0.012)
3	1.527 (± 0.025)	0.900 (± 0.015)	0.664 (± 0.012)	0.588 (± 0.010)
5	1.455 (± 0.023)	0.803 (± 0.015)	0.586 (± 0.011)	0.530 (± 0.009)
7	1.372 (± 0.023)	0.775 (± 0.014)	0.553 (± 0.010)	0.485 (± 0.009)
10	1.312 (± 0.023)	0.708 (± 0.013)	0.499 (± 0.009)	0.435 (± 0.008)

Figure 7.3: **Tabular summary of the first actions chosen by VOWSS policy**, where we show the mean Euclidean distance from the LQG solution, and the corresponding standard error in parentheses.

horizons [94]. The action width decay is manually set to $\gamma_a = 0.4$ after a careful inspection to balance performance and run time, which means that the action width for first step is 200, and the action width for second step is 80. While this results in number of evaluations on the order of 10^6, we include the performance of VOWSS as a reference.

We first show the scatter plots of the actions chosen by each solver in Figure 7.2. Each scatter plot shows the result of 1000 simulations. While POMCPOW and BOMCP find solutions biased towards the rollout actions, we see that the effect is much less pronounced in VOMCPOW. Specifically, when the rollout policy is set to Riccati policy, both POMCPOW and BOMCP heavily resort to picking the Riccati solution as shown by the large point mass corresponding to the Riccati solution in the scatter plots. Even though VOMCPOW still produces a sizable mass of points on and surrounding the Riccati solution, it otherwise picks solutions that are close to the LQG solution. VOMCPOW scatter plots show that the bias and the variance of the optimal action estimates are noticeably smaller than those of POMCPOW and BOMCP. Although VOWSS cannot be directly compared to the other solvers since the number of iterations is not on the same order and it doesn't rely on a rollout policy, the overall shape of the scatter plot looks similar to those of VOMCPOW.

In addition, we study the effects of varying the state and action widths for VOWSS. Figure 7.3 shows the mean Euclidean distance to the LQG solution for different combinations of state and action widths. Each cell in the table shows the result of 1000 simulations, and the other hyperparameters are left intact. As we increase either the state or action width, the mean

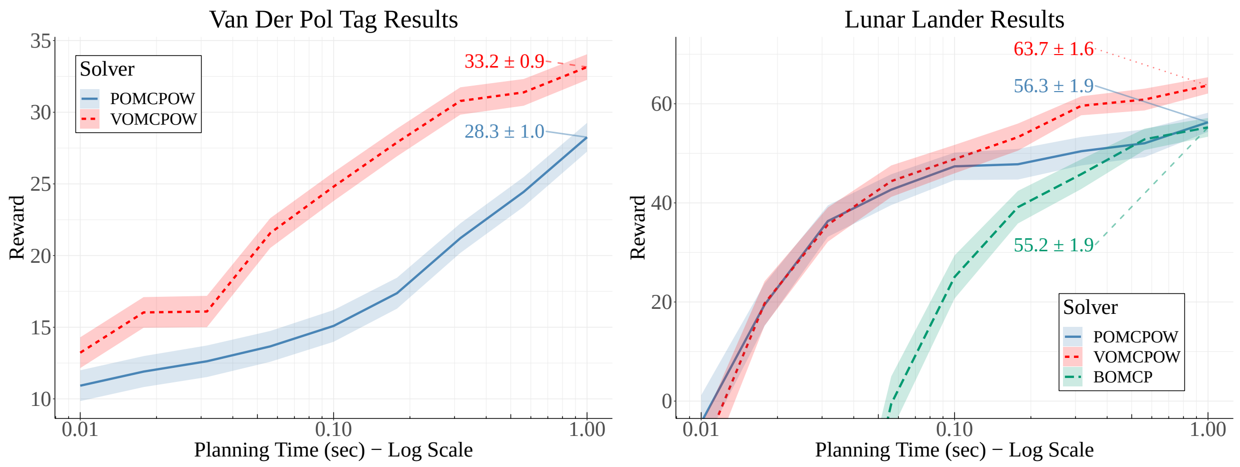


Figure 7.4: **Comparative benchmark results summary of continuous space POMDP solvers over different planning time allotments.** Ribbons surrounding the Monte Carlo benchmark mean estimates indicate one standard error confidence bands. Planning times are given in log scale.

	VDP Tag		Lunar Lander	
VOMCPOW	$\mathbf{33.2 \pm 0.9}$	🟩	$\mathbf{63.7 \pm 1.6}$	🟩
POMCPOW	28.3 ± 1.0	🟩	56.3 ± 1.9	🟩
BOMCP			55.2 ± 1.9	🟩

Table 7.2: **Tabular comparative benchmark summary of continuous space POMDP solvers.** All results are given as (mean ± stderr) for a 1 second planning time allotment. The algorithm(s) with the best average performance within one standard error is shown in boldface for each experiment.

distance and the standard error decrease. This indicates that VOWSS chooses better actions by increasing the state and action widths, which confirms our theoretical results: larger state widths should increase value estimation accuracy and larger action widths should improve action optimization.

7.4.2 Van Der Pol Tag

Next, we test POMCPOW and VOMCPOW on the Van Der Pol Tag (VDP Tag) problem introduced by [171], which is the only problem in their work with continuous state, observation and (hybrid) action spaces. In VDP Tag, an agent navigates through 2D box to tag a target with randomized initial state that follows a dynamics model defined by the Van Der Pol differential equation. The agent moves at a fixed speed, but can choose the direction of travel and can decide to make an accurate observation of where the target is at a higher cost than making the default noisy observation. While the target can freely move within the 2D box, the agent is blocked when it comes into contact with a barrier.

We show the mean reward for 1000 simulations for each solver and each planning time plotted in log scale of seconds in Figure 7.4. We observe that VOMCPOW outperforms POMCPOW at every planning time by a statistically significant margin. It is also worth noting that VOMCPOW takes almost an order of magnitude less planning time to reach the mean reward of 25 compared to POMCPOW. While VDP Tag is a continuous space POMDP, the rewards are still discrete in both state and action spaces, which suggests that even with discrete jumps in the reward function, VPW can still optimize to find better actions. In addition, while it is possible to adapt BOMCP to solve VDP Tag, it requires nontrivial modifications to the action space due to the action space being hybrid ($A = [0, 2\pi) \times \{0, 1\}$) and the angle space being a modular space. This further illustrates the ease of implementing VPW since we only need to supply VPW with a distance metric on the action space, which makes VPW suitable for hybrid action spaces as well.

7.4.3 Lunar Lander

Lastly, we test POMCPOW, VOMCPOW and BOMCP on the modified lunar lander problem proposed in [124], where the main objective is to guide a vehicle to land in a target zone safely. In this version of lunar lander, the vehicle state is defined as $(x, y, \theta, \dot{x}, \dot{y}, \omega)$, and we only obtain noisy observations of the angular rate, horizontal speed, and above-ground level. The action space is defined by the tuple (T, F_x, δ) where T is the main vertical thrust, F_x the corrective horizontal thrust, and δ the offset. The default rollout policy is proportional control based on the observation.

Once again, we show the mean reward for 1000 simulations for each solver and each planning time plotted in log scale of seconds in Figure 7.4. VOMCPOW outperforms both POMCPOW and BOMCP by a statistically significant margin once the planning time exceeds 0.1 seconds. Below 0.1 seconds, the performances of POMCPOW and VOMCPOW are similar due to the problem having a long horizon, in which the effects of VPW will not be as apparent since best Voronoi cell actions will not be sampled very often.

We note that the performance of POMCPOW here seemingly differs from the results in [124], which might be due to a several factors. The lander problem has a high variance in the returns, since failure to land results in a steep penalty of -1000. This failure mode does not happen very often even within 1000 iterations, so the number of failures significantly impacts the mean rewards. Furthermore, we are comparing the algorithms based on planning time rather than number of queries. POMCPOW queries much faster than BOMCP, and average planning time may not sufficiently capture the relationship between total planning time and number of queries for a progressively built tree. It is also possible that authors of [124] did not set POMCPOW first action to be the rollout action.

7.5 Conclusion

In this chapter, we have introduced Voronoi Progressive Widening (VPW), a versatile technique to effectively handle continuous or hybrid action spaces in MDPs and POMDPs. Consequently, we proposed two VPW-based tree search algorithms to demonstrate convergence guarantees and efficiency, justifying the theoretical soundness, versatility and practicality of VPW for many continuous or hybrid action space POMDPs.

This study has yielded a several key insights, which suggest some promising future directions to explore. While each of the VPW-based algorithms either enjoy theoretical guarantees or computational efficiency, the gap between theoretical soundness and practicality still remains. In particular, the state and action selection heuristics as it is utilized in POMCPOW still need to be integrated into the theoretical algorithms like POWSS to bridge the gap between theory and practice. On the other hand, we believe that the VPW technique itself should also have some theoretical guarantees similar to the DPW technique [43].

Additionally, since the convergence guarantees of VOO hold as long as the reward function is locally smooth around at least one global optimum [94], more extensive study could be done for effectiveness of VPW on MDPs and POMDPs with discrete reward structures. Similar to how the Rényi divergence requirement for POWSS [109] could be a difficulty indicator for likelihood-weighted sparse tree solvers, the proof techniques utilized for VOO and DPW could offer insights on what continuous or hybrid action space problems are harder to solve than others.

Part III

Learning and Robotics: Compositional Learning-based Planning with Algorithmic Priors

Chapter 8

Overview

Chapter Outline

In this chapter, we give an overview of all the works discussed in Part III. We introduce two ways of reasoning about real world robotics application scenarios. First work, Visual Tree Search, offers a direct integration of pre-existing particle-based POMDP planning methods with generative neural network model components. Second work, Sequence-Conditioned Transporter Networks, offers an alternative approach to solve visual robotic arm manipulation tasks by introducing sequence conditioning and weighted multi-task learning. Some discussions, results, and relevant figures are borrowed & repeated from the original works listed in each chapter.

8.1 Compositional Learning

This part of the dissertation aims at applying the principles of compositional learning to designing systems and architectures for autonomous planning. Compositional learning attempts to break down a learning task into specialized and compartmentalized neural network components. These learning-enabled components are then integrated into one system to allow learning complex relations and semantics while keeping the training resources and amount of data minimal. Furthermore, compositional learning approaches also benefit from the compartmentalized architecture that are not only often robust and perform better than their naive counterparts but also provide better interpretability in their outputs and policies. Compositional learning also allows *algorithmic priors* to be seamlessly integrated into the system, where specific domain knowledge and problem structures can be factored into the component design or the overall integration of different components.

Thus, continuing from the previous Part II where we are primarily focused on theoretical foundations of model-based POMDP planners, in this part of the dissertation we start considering how to effectively integrate these planners and concepts with learning-enabled components. Learning-enabled components from modern machine learning can effectively reason about complex data structures like images, videos, and text, and learn to recognize complex patterns such as human behavior and game strategies. In particular, this part of the dissertation focuses on solving realistic robotic problems in the domains of navigation and manipulation.

8.2 Visual Tree Search for Vision POMDPs

In Chapter 9, we introduce the Visual Tree Search algorithm for Vision POMDPs [48]. While POMDP is a powerful framework for capturing decision-making problems that involve state and transition uncertainty, most current POMDP planners cannot effectively handle high-dimensional image observations prevalent in real world applications, and often need lengthy online training that requires interaction with the environment.

We propose Visual Tree Search (VTS), a compositional learning and planning procedure that combines generative models learned offline with online model-based POMDP planning. This work is a natural extension from the previous chapters, as we directly reason about how to integrate the algorithmic prior of the POMDP problem structure and particle belief representation with unsupervised generative neural network models. The deep generative observation models generate and evaluate the likelihood of future image observations in a Monte Carlo tree search planner. We show that VTS is robust to different types of image noises that were not present during training and can adapt to different reward structures without the need to re-train. This new approach significantly and stably outperforms several baseline state-of-the-art vision POMDP algorithms while using a fraction of the training time.

8.3 Sequence-Conditioned Transporter Networks for Tabletop Manipulation

In Chapter 10, we introduce Sequence-Conditioned Transporter Networks for vision-based tabletop robotic arm manipulation [111]. Enabling robots to solve multiple manipulation tasks has a wide range of industrial applications. While learning-based approaches enjoy flexibility and generalizability, scaling these approaches to solve such compositional tasks remains a challenge.

We aim to solve multi-task learning through the lens of sequence conditioning and weighted sampling. First, we propose a new suite of benchmark specifically aimed at compositional tasks, MultiRavens, which allows defining custom task combinations through task modules that are inspired by industrial tasks and exemplify the difficulties in vision-based learning and planning methods. Second, we propose a vision-based end-to-end system architecture, Sequence-Conditioned Transporter Networks, which augments Goal-Conditioned Transporter Networks with sequence-conditioning and weighted sampling and can efficiently learn to solve multi-task long horizon problems. Our analysis suggests that not only the new framework significantly improves pick-and-place performance on novel 10 multi-task benchmark problems, but also the multi-task learning with weighted sampling can vastly improve learning and agent performances on individual tasks.

While this approach is not exactly aligned with the traditional POMDP methods discussed earlier in Part II, this work offers an alternative perspective on handling complex observations for visual robotic manipulation. Nevertheless, we apply the same philosophy of model-based planning with algorithmic priors, enabled by the pre-existing Transporter networks [199] that specialize in visual tabletop robotic arm tasks. Specifically, we employ compositional learning once again to devise an architecture that can reason with input sequences of demonstrations and learn and perform multiple tasks at once.

Chapter 9

Compositional Learning-based Planning for Vision POMDPs

Chapter Outline

This chapter is based on the paper "Compositional Learning-based Planning for Vision POMDPs" [48], written in collaboration with Sampada Deglurkar, Johnathan Tucker, Zachary N. Sunberg, Aleksandra Faust, and Claire J. Tomlin.

9.1 Introduction

Many sequential decision making problems, such as autonomous driving [15, 172], cancer screening [13], spoken dialog systems [192], and aircraft collision avoidance [77], involve uncertainty in both sensing and planning. Planning under partial observability is challenging, as the agent must address both localization and uncertainty-aware planning through active information gathering in the environment. By capturing the observation uncertainty through a *belief* distribution over possible states, the agent will be able to fully close the observation-plan-action loop. While there are many methods that can either handle visual localization [83, 87, 90] or planning under uncertainty [176, 179], a naïve combination of these methods, for instance by assuming the certainty equivalence principle or simplifying the observation space, may not yield meaningful closed-loop control policies that enable active information gathering under more general environmental assumptions.

The partially observable Markov decision process (POMDP) formalism is a powerful framework that can capture and systematically solve these sequential decision making under uncertainty problems. However, finding an optimal POMDP policy is computationally demanding, and often intractable, due to the uncertainty introduced by imperfect observations [139]. One popular approach to deal with this challenge is to use *online* algorithms that look for local approximate policies as the agent interacts with the environment rather

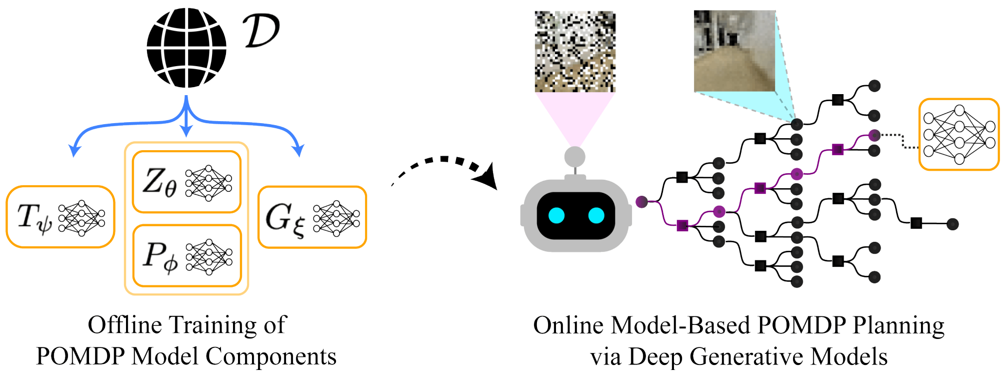

$\mathcal{D}$
Z_θ
T_ψ
P_ϕ
G_ξ
Offline Training of POMDP Model Components
Online Model-Based POMDP Planning via Deep Generative Models

Figure 9.1: **Visual Tree Search** interfaces offline deep generative model training with online model-based POMDP filtering and planning.

than a global policy that maps every possible outcome to an action, such as Monte Carlo tree search (MCTS) and similar variants [28, 101, 159, 171, 191]. Many of these state-of-the-art MCTS algorithms enjoy computational efficiency [110, 124, 171] and finite sample convergence guarantees to the optimal policy [109, 110]. Despite their flexibility and optimality, these methods rely on having access to generative models and observation density models, which limits the class of problems they can solve in practice. In many realistic scenarios with high dimensional observations like RGB images, these POMDP methods cannot be applied without knowing or learning the relevant models or simplifying the environment.

Recently, there has been an increased interest in solving *vision POMDPs*, i.e. POMDPS with image or video observations, using deep learning methods. Model-free vision POMDP algorithms train an end-to-end deep neural network policy to learn both a latent belief representation and a planner [80, 89, 131], which benefit from not having to specify the transition and observation models and can learn complex policies. However, they may lack interpretability, not generalize well to new unseen tasks, and not leverage much prior knowledge about the system, especially in robotics settings. In contrast, model-based vision POMDP algorithms [160, 183] combine classical filtering and planning techniques with deep learning. While such algorithmic structure allows the models to focus on specific tasks, making them sample efficient and robust, these methods often rely on simplified approaches for planning in the belief space and require learning an advantage function that only partially captures uncertainty by coupling observations with rewards.

Thus, we propose Visual Tree Search (VTS), a procedure to solve vision POMDPs by combining deep generative models and online tree search planning, effectively framing the POMDP reinforcement learning problem as a compositional unsupervised learning problem. Our key insight is to utilize compositional learning approaches to bridge the offline training of individual sets of models with online model-based planning that uses learning-enabled components. Introducing the algorithmic prior knowledge of particle filtering and MCTS decreases the computational complexity required by the learning and captures state and transition uncertainty without dependence on reward structure. Our empirical analy-

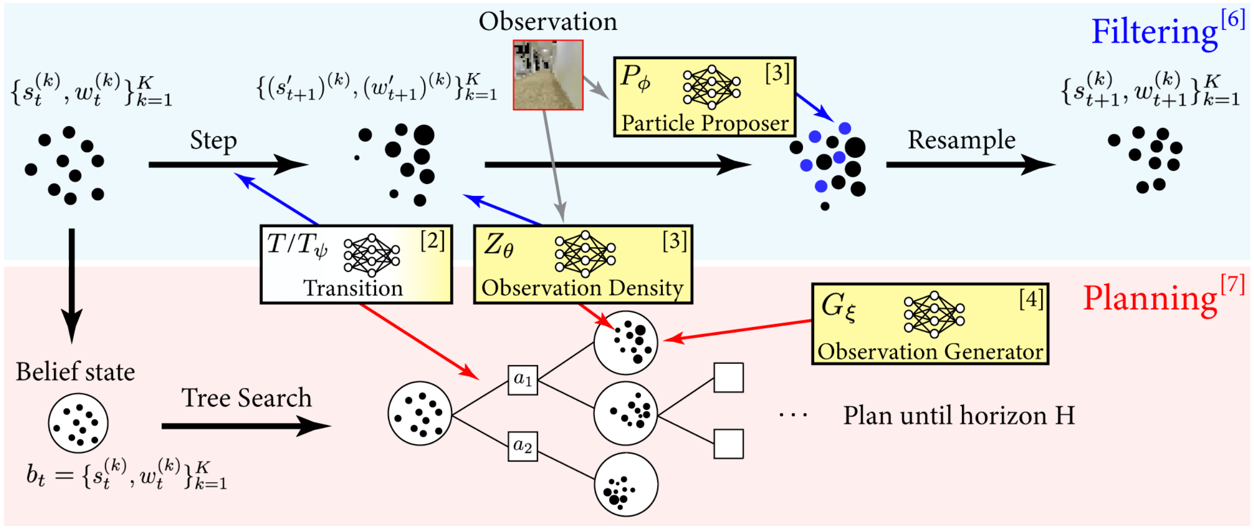

Observation
Filtering[6]
$\{s_t^{(k)}, w_t^{(k)}\}_{k=1}^K$
$\{(s'_{t+1})^{(k)}, (w'_{t+1})^{(k)}\}_{k=1}^K$
P_ϕ [3] Particle Proposer
$\{s_{t+1}^{(k)}, w_{t+1}^{(k)}\}_{k=1}^K$
Step
Resample
T/T_ψ [2] Transition
Z_θ [3] Observation Density
G_ξ [4] Observation Generator
Planning[7]
Belief state
$b_t = \{s_t^{(k)}, w_t^{(k)}\}_{k=1}^K$
Tree Search
a_1
a_2
Plan until horizon H

Figure 9.2: **Overview of Visual Tree Search (VTS).** The models learned with neural networks are shown in yellow – the transition model is shown in gradient, since it can be learned or pre-defined. The contributions of the models for filtering are shown in blue arrows, and planning in red arrows, and numbers in brackets show corresponding steps in Algorithm 9.1.

ses demonstrate that tackling uncertainty via VTS enhances performance, robustness, and interpretability of neural network components.

9.2 Visual Tree Search

Solving a POMDP can be split into solving the filtering and the planning problems separately. In the Visual Tree Search (VTS) algorithm, we integrate the learned POMDP model components with classical filtering and planning techniques. This results in a POMDP solver that can learn model components that are more sample efficient and interpretable than end-to-end approaches. It also benefits from having a robust particle filter and planner built upon techniques with theoretical guarantees, and can adapt to different task rewards. To integrate planning techniques that use online tree search, we must have access to a conditional generative model that can generate image observations o from a given state s according to the likelihood density $Z(o|s)$. In this section, we outline the filtering and planning algorithms and models, and the compositional training procedure.

9.2.1 Differentiable Particle Filtering

For particle filtering, we leverage a family of architectures called Differentiable Particle Filters (DPF) [83], which combine classical particle filtering algorithms with convolutional neural networks that can handle complex observations. We learn two neural network-based models:

Algorithm 9.1 Visual Tree Search Algorithm.

Input: Hyperparameters for neural networks (ζ), DPF (χ), MCTS (ρ), maximum time step $T_{\max}$.
Output: Action a_t at each step t.

```
1: Collect data $\mathcal{D}$ of tuples $(s_t, a_t, o_t, s_{t+1})$ through random sampling or exploration.
2: [Optional] Train the transition model $T_\psi$ with data tuples $(s_t, a_t, s_{t+1})$.
3: [Training] Jointly train the observation density model $Z_\theta$ and particle proposer model $P_\phi$ with
   data tuples $(s_t, o_t, \{\hat{s}_{t,i}\})$, where $\{\hat{s}_{t,i}\}$ are particle estimates of $s_t$.
4: [Training] Train the observation conditional generator model $G_\xi$ with data tuples $(s_t, o_t)$.
5: for $t = 1$ to $T_{\max}$ do
6:     [Filtering] If $t = 1$, initialize the belief state $b_t$. Otherwise, after receiving $o_t$, update the
       belief state $b_t$ with DPF$(T_\psi, Z_\theta, P_\phi, \chi)$.
7:     [Planning] Run MCTS$(T_\psi, Z_\theta, G_\xi, \rho)$ on the belief state $b_t$ to obtain and perform action $a_t$.
```

(1) Observation density Z_θ that gives the likelihood weights $w_t = Z_\theta(o_t|s_t)$, (2) Particle proposer P_ϕ that allows us to sample states $s_t \sim P_\phi(o_t)$. The Greek letters denote the parameters of these neural network models. Specifically for our work, we adapt the DPF architecture introduced in DualSMC [183], in which the observation and proposer networks are trained with an adversarial optimization objective. In principle, we can also train DPF with entropy regularization [41] with optimality guarantees. We assume that the transition model T is known, which is not a limiting assumption for many POMDP problems (e.g. POMDPs with physical dynamics), but in principle it can be learned and modeled with a neural network T_ψ as well.

The filtering procedure is described below; the belief is represented by $b_t \approx \{s_t^{(k)}, w_t^{(k)}\}_{k=1}^K$, with K as the number of particles. First, the agent takes a step with an action chosen by the planner. Then, the agent updates the predicted states $(s'_{t+1})^{(k)}$ with the transition model T. The observation o_t obtained from the environment is fed into the observation density Z_θ, which provides the likelihood of an observation given the state. This is used to update the likelihood weights $(w'_{t+1})^{(k)}$ of each particle. In order to ensure robustness in the particle representation of the belief state, the particle proposer deep generative model P_ϕ proposes plausible state particles for a given observation, replacing some fraction of the particles. This fraction is made to decay exponentially over time.

9.2.2 Tree Search Planner

Monte Carlo Tree Search. For the online planner, we use the Particle Filter Trees-Double Progressive Widening (PFT-DPW) algorithm [171]. PFT-DPW is a particle belief-based MCTS planner that is relatively easy to implement and efficiently vectorizes particle filtering. Additionally, a simplified version of PFT-DPW has optimality guarantees [109]. However, any continuous POMDP tree search planner can be used instead.

For our navigation problems, we discretize the action space such that the robotic agent moves in the 8 cardinal and diagonal directions with full thrust. While this means we work with a limited action space, we can ensure that we travel with full thrust to get to the goal faster and reduce the complexity of both planning and generating observations. However, we could also work with a continuous action space in principle. Furthermore, we provide PFT-DPW with a naïve rollout policy of actuating straight towards the goal and calculating the expected reward, which PFT-DPW can use as a reference and vastly improve upon.

Observation conditional generative model. Deep conditional generative models enable online model-based POMDP planning with images. To plan with a tree search planner, we need to be able to generate the next step states and observations, and evaluate the likelihood of the observations. With models from DPF, we can generate the next step state with transition model T and calculate the likelihood weight with observation density Z_θ, which are also used in the filtering procedure. Thus, we only additionally need to learn a deep generative model G_ξ that generates an observation o_t given a state s_t: $o_t \sim G_\xi(s_t)$. We use a Conditional Variational Autoencoder (CVAE) [163] to model the observation conditional generator G_ξ, where the state s_t is the conditional variable, since it had the most consistent training and performance in our experiments.

9.2.3 Compositional Training of Visual Tree Search

We train each neural network model with pre-collected data $\mathcal{D}$, containing tuples of (s_t, a_t, o_t, s_{t+1}), as shown in Lines 1-4 of Algorithm 9.1. We provide the Z_θ and P_ϕ models with randomly sampled batches of state, observation, and synthetic belief particle sets $(s_t, o_t, \{\hat{s}_{t,i}\})$. While the belief particle sets are not a part of the pre-collected data, we can easily create them by sampling states from a normal distribution centered at s_t. The Z_θ and P_ϕ are trained with the adversarial objective in DualSMC [183]: Z_θ serves as a discriminator that gives higher likelihoods to states that are more likely for a given observation, and P_ϕ serves as a conditional generator that proposes plausible state particles for a given observation. The G_ξ model is trained with random batches of samples of state and observation pairs: (s_t, o_t). These random batches provide a good data prior for all types of states and observations an agent may encounter during planning, unlike methods such as DualSMC which only encounter data collected from the locally optimal policy. For our neural network training procedures, we make the ground truth state information available to the planner during training, but keep it unavailable during testing.

Compared to other on-policy RL algorithms and architectures, the VTS training procedure shifts the POMDP problem from a reinforcement learning problem to a compositional learning problem, in which training each model in the POMDP is framed as an unsupervised learning problem. This drastically decreases the problem complexity, as we have explicit control over the learning objective of each model and the schedule of the training. It also allows us to better approximate the data distribution via sampling or exploration, as opposed to an evolving on-policy planner distribution that often starts off poorly and provides heavily biased data.

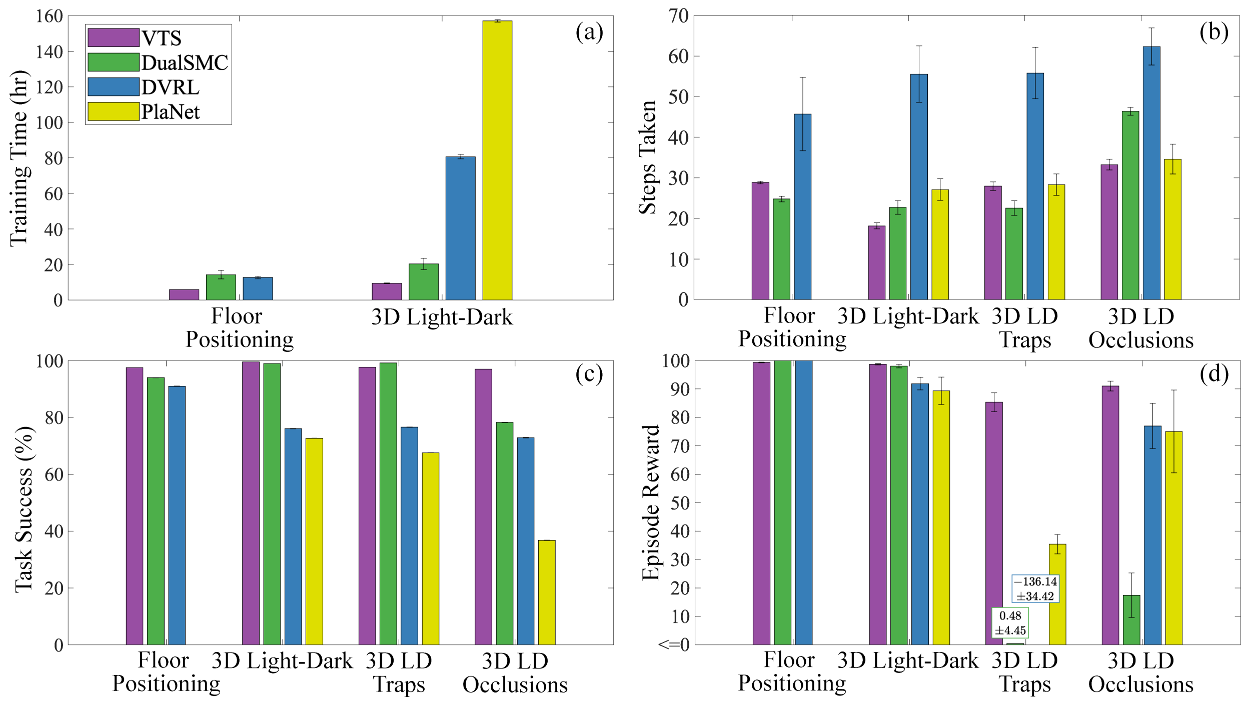


Figure 9.3: **The results of 4 different planners, VTS, DualSMC, DVRL, and PlaNet, on Floor Positioning and 3D Light-Dark.** Sub figures denote: (a) training time, (b) steps taken, (c) task success, and (d) episode reward.

9.3 Experiments

We compare VTS to other state-of-the-art vision POMDP algorithms, DualSMC [183], DVRL [80] and PlaNet [69], on two benchmark vision POMDP problems. First, we tested our algorithm on the 2D Floor Positioning problem [183] to demonstrate that using the VTS learning and planning procedure can significantly decrease the training time to learn a solver that is agnostic to the task or reward structure of the problem. Then, we prepared our own version of the 3D Light-Dark problem in the Stanford Large-Scale 3D Indoor Spaces dataset [10] to set up a more challenging navigation task that requires the agent to plan with realistic indoor building RGB images. We also performed ablation tests with the 3D Light-Dark experiment in which we varied the reward structure using spurious traps and the observation space using random visual occlusions. In each section, we calculate the results of the planner performances for 1000 testing episodes for Floor Positioning and 500 for 3D Light-Dark. For online planning speed, VTS takes around 0.26 seconds to plan for Floor Positioning and 0.80 seconds for 3D Light-Dark on average, while other planners require less than 0.05 seconds for both problems. The experimental summary figures are given in Fig. 9.3 and the full tabular summary in Table 9.1. In addition, VTS training data details are given in Appendix D.2, and the hyperparameters and computation details in

Planner & Problem (Successful Trials)	**Task Success (%)**	**Episode Reward**	**Steps Taken**	**Training Time (h)**	**Planning Time (s)**	**Particle Distance**
Floor Positioning						
VTS (10/10)	**97.6** ±0.002	99.36 ± 0.12	28.85 ±0.28	**5.88** ±0.02	0.26 ±0.0022	**0.07** ±0.002
DualSMC (10/10)	94.0 ±0.049	**100.0** ±0.0	**24.78** ±0.70	14.27 ±2.39	0.04 ±0.0003	0.08 ±0.003
DVRL (10/10)	91.0 ±0.084	**100.0** ±0.0	45.71 ±9.01	12.74 ±0.72	0.0063 ±6.4e-5	–
3D Light-Dark						
VTS (10/10)	**99.6** ±0.002	**98.71** ± 0.22	**18.18** ± 0.76	**9.42** ±0.21	0.76 ± 0.005	**0.48** ± 0.07
DualSMC (**6**/10)	99.0 ±0.009	98.05 ± 0.62	22.71 ± 1.70	20.31 ±3.13	0.06 ± 0.003	0.50 ± 0.02
DVRL (10/10)	76.1 ±0.081	91.88 ± 2.2	55.33 ± 6.94	80.65 ±1.18	0.0076 ±3.0e-4	–
PlaNet (4/4)	72.7 ±0.029	89.35 ± 4.84	27.1 ± 2.67	157.1 ±0.66	0.061 ±4.1e-4	–
3D LD + Traps						
VTS	97.7 ±0.003	**85.34** ±3.33	27.96 ±1.06	–	0.80 ±0.0039	**0.42** ±0.06
DualSMC	**99.2** ±0.007	0.48 ±4.45	**22.55** ±1.80	–	0.06 ±0.0006	0.49 ±0.02
DVRL	76.6 ±0.078	-136.14 ±34.42	55.79 ±6.34	–	0.0076 ±2.9e-4	–
PlaNet	67.6 ±0.016	35.41 ±3.38	28.31 ±2.63	–	0.061 ±7.1e-4	–
3D LD + Occlusions						
VTS	**97.0** ±0.007	**91.05** ±1.73	**33.25** ±1.33	–	0.77 ±0.0051	**0.82** ±0.08
DualSMC	78.3 ±0.052	17.45 ±7.82	46.38 ±0.97	–	0.06 ±0.0005	0.91 ±0.03
DVRL	72.9 ±0.081	77.0 ±7.97	62.32 ±4.56	–	0.0075 ±3.1e-4	–
PlaNet	36.8 ±0.078	75.07 ±14.55	34.62 ±3.69	–	0.059 ±5.4e-4	–

Table 9.1: **Summary of planner performances on the vision POMDP problems: Floor Positioning problem and the 3D Light-Dark problem variants.** Each column reports the mean of means estimator over different seeds, and ± indicates one standard error of the estimator. Mean particle distance is defined as the mean of the distance between the mean belief state and the true state. All statistics are for when the agent is successful.

9.3.1 Floor Positioning Problem

In this problem, a robotic agent is randomly placed around the center of either the top or the bottom floor, and it must infer its position by relying on a radar-like observation in all four cardinal directions, which bounces off the nearest wall. The top and bottom floors are indistinguishable within the "corridor states" of the hallways, but the robotic agent can take advantage of the "wall states" by traveling closer to the top or bottom walls of each floor, where it can receive different observations due to the different wall placements in each floor. The agent must reach the goal and avoid the trap, where the goal is the left end of the hallway in the top floor and right end in the bottom floor, and the trap is at the opposite side of the goal in each floor.

Planner comparison. Overall, VTS is the most successful among the three planners with reasonable online planning time and steps taken, while requiring less than half the training time compared to the other planners. In Fig. 9.4, we show an example trajectory of the VTS

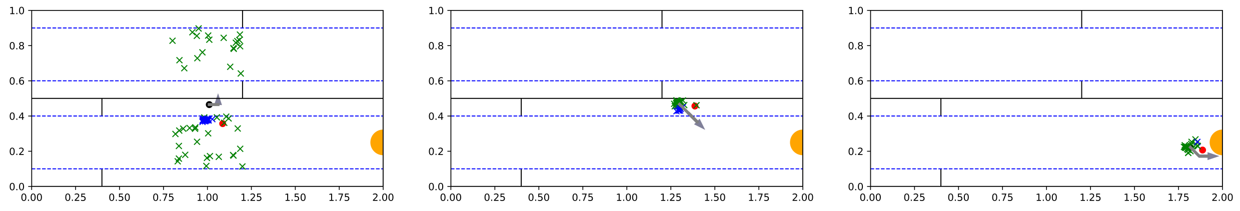

(a) Initial Belief (b) Localization (c) Reaching Goal

Figure 9.4: **Example VTS behavior for Floor Positioning**, where the planner starts the problem with the initial belief (a), localizes in the wall states (b), and successfully reaches the goal (c).

agent, in which the agent quickly localizes in the wall states and then reaches the goal.

The VTS system gains significant performance optimality and efficiency by balancing offline training and online planning. Specifically, VTS training is quick since it does not rely on planner performance and only needs to be supplied with relevant state and observation batches. However, other online training algorithms require the policy to interact with the environment, and as such, much of the time is spent earlier in the training episodes when the planner can neither localize nor reach the goal. Since tree search methods learn the policy online and are more difficult to parallelize, they typically trade off the offline training time with the online learning and planning time. Despite the longer planning, VTS plans reasonably fast while maintaining the efficiency and flexibility of an online planner. We also tested DualSMC with a known T model to ensure VTS is not given an advantage by knowing T for model-based planning, but observed no statistically significant difference in performance.

9.3.2 3D Light-Dark Problem

The Light-Dark problem is a family of problems in which an agent starting in a "dark" region can localize by taking a detour into a "light" region before reaching the goal. Observations are noisier in the dark region than in the light region. The 3D Light-Dark problem extends this problem to have 3D image observations with salt-and-pepper pixel-wise noise.

The observations consist of $32 \times 32 \times 3$ RGB images from the agent's perspective. Our environment is more challenging to reason with than the one in [183], since the RGB images are rendered from a realistic hallway scene dataset rather than a synthetic environment. The agent receives a reward for reaching the goal and a penalty if it enters a "trap".

Planner comparison. In the Light-Dark problem and its variants, VTS performs the best among four planners not only by taking full advantage of the image features, but also by being able to adapt to different reward structures and distribution shifts in the noisy observations during test time. Also, DualSMC sometimes fails to have successful training

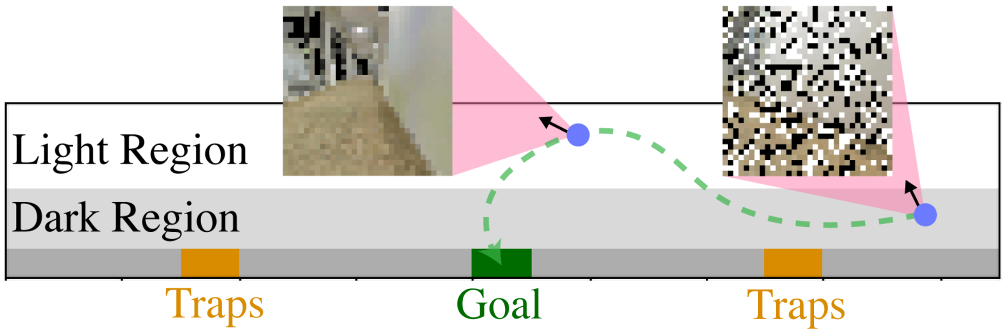


Figure 9.5: **Example schema for 3D Light-Dark**, localizing in the light region to reach the goal in the dark region.

seeds, and DVRL and PlaNet have lower task success rates despite all successful training seeds.

Among the successful training seeds for DualSMC, VTS and DualSMC have comparable success rates, but the VTS planner takes fewer steps to reach the goal. While this is only 4.5 steps difference on average, further inspection of the planner trajectories in Fig. 9.6 reveals an interesting insight. Unlike the traditional light-dark problems, the VTS results for 3D Light-Dark suggest that the planner in fact is able to localize solely with the noisy observations in the dark region. This shows that while the salt-and-pepper noise observations are hard to interpret, the corrupted RGB images actually contain sufficient information to localize given enough observations.

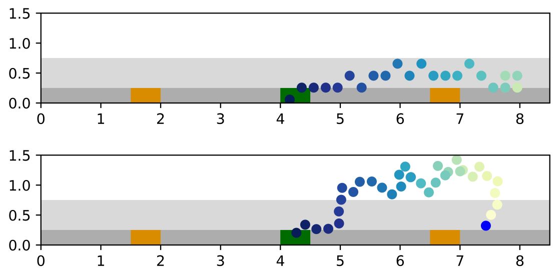


Figure 9.6: **VTS can localize without light observations** (top), unlike DualSMC (bottom).

Test-time changes: Spurious traps. VTS has additional advantages in its robustness and adaptability, which we showcase through experiments on modified Light-Dark environments. First, to test the adaptability of different planners on test-time reward structure difference, we perform an ablation with spurious traps that appear during test time. These regions could represent the presence of unforeseen obstacles or hazards. Over 500 testing episodes, we randomly generate the locations of two 0.5×0.5 square

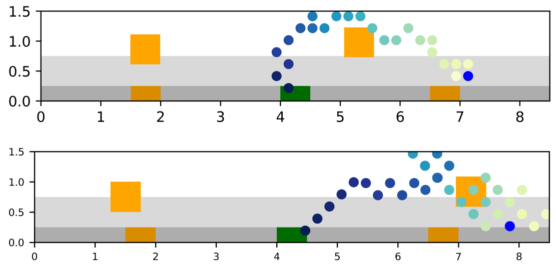


Figure 9.7: **VTS can avoid new test time traps** (top), while other planners like DVRL disregard the traps (bottom).

traps over a particular strip in the environment and compare the performances of the models without additional training or modification.

Since VTS uses an online planner, it can easily adapt to new reward structures at test time without the need to retrain an entire planner. We see in Fig. 9.7 that while the other planners ignore these new trap regions because they are not represented by their models, the VTS planner is able to take into account rewards seen while planning. Thus, while the success rate of all planners remains similar to the vanilla experiment, VTS achieves a much higher reward than all other planners while taking more steps as it actively tries to avoid these traps.

Test-time changes: Image occlusions. Second, we compare planner performances when the form of the noise in the image observations changes during test time. During test time only, images seen in the dark region contain random blacked out 15×15 squares instead of salt-and-pepper noise.

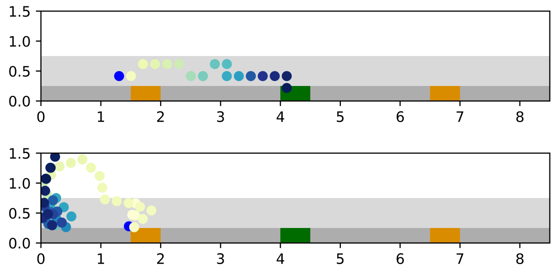


Figure 9.8: **VTS can optimally plan with occlusions** (top), while others like PlaNet fail to reach the goal (bottom).

We also find that VTS is robust to distribution shifts in the image noise at test time. Fig. 9.8 shows VTS maintaining a good success rate and number of steps taken, while other planners are struggling to generalize to this new scenario. The additional robustness of VTS seems to be due to the Z_θ and P_ϕ models being trained with high fidelity data samples through random batch sampling or exploration.

In contrast, other planners that use on-policy training and complex non-separable architectures suffer from such distribution shifts. This is likely due to lack of control over on-policy exploration and the inability to cover many possible scenarios with such limited interaction with the environment. Due to this difference in data variation, we conjecture that the Z_θ and P_ϕ models in VTS learn more precise and robust features.

9.4 Conclusion

The development of Visual Tree Search suggests new ways to think about integrating uncertainty aware learning and control. VTS demonstrates that a more principled integration of control and planning techniques both illuminates the interpretability of each model component and saves training time by cleanly partitioning what each network is responsible for. This benefits researchers and practitioners alike, as a more interpretable and theoretically principled planner that is also quicker to train is beneficial in many practical safety critical scenarios with limited training resources.

Our work also raises the question of what is the most natural, effective, and safety-ensured method of combining pre-existing controllers and planners, which are extensively studied both theoretically and experimentally, with learning-based components, which can model many complex functions and phenomena. VTS is one such way of accomplishing that goal, but there are other ways to achieve principled integration of learning and control.

Chapter 10

Sequence-Conditioned Transporter Networks

Chapter Outline

This chapter is based on the paper "Multi-Task Learning with Sequence-Conditioned Transporter Networks" [111], written in collaboration with Andy Zeng, Brian Ichter, Maryam Bandari, Erwin Coumans, Claire Tomlin, Stefan Schaal, and Aleksandra Faust.

10.1 Introduction

Compositional multi-task settings are found in a number of industrial settings: from placing gears and chaining parts for assembly in a manufacturing plant, to routing cables and stacking servers for datacenter logistics. Such tasks often require robots to perform multiple maneuvers using different manipulation skills *(primitives)*. Humans have the ability to generalize between compositional tasks of varying complexity and diversity [182], but solving compositional tasks remains a major challenge in robotics. Enabling robotic autonomous agents to learn and reason through compositional multi-tasks can further unlock autonomous manufacturing frontiers by diversifying the problems that robots can solve while significantly decreasing production costs [114].

Complex manipulation is compositional by nature – requiring multi-step interactions with the physical world, and can be formulated as sequencing multiple individual tasks over a longer horizon. Learning-based approaches offer the flexibility to learn complex manipulation skills and generalize to unseen configurations [56, 85, 107, 115]. However, scaling these approaches to solve compositional tasks may not always be successful [85, 86, 178]. Specifically, task compositionality often requires exponentially increasing amounts of demonstrations for agents to learn and generalize well, while avoiding compounding long horizon

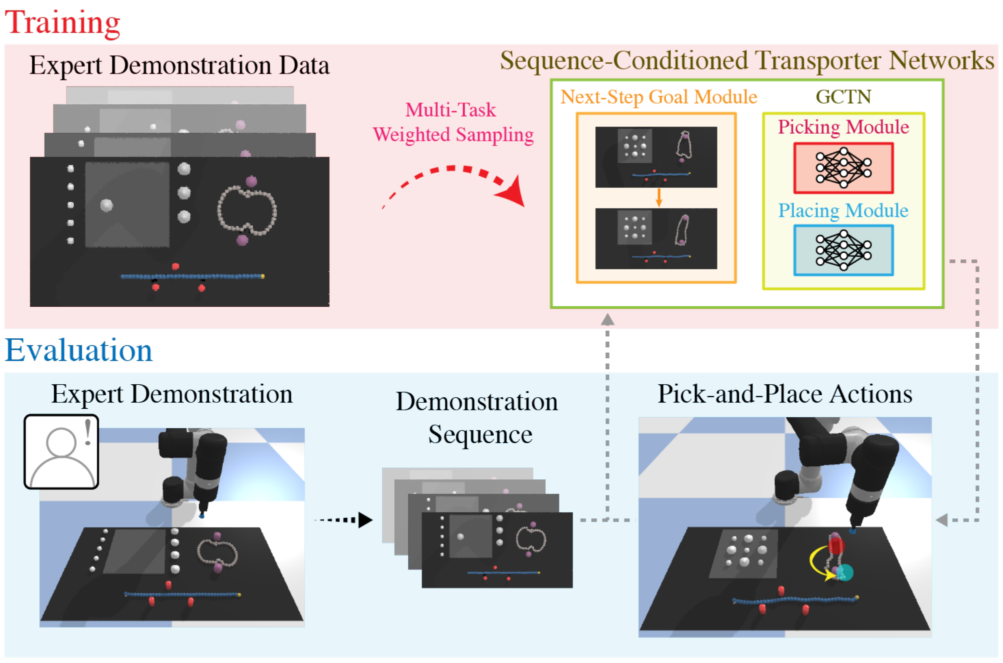


Figure 10.1: **Overview of Sequence-Conditioned Transporter Networks (SCTN).** SCTN is trained through multi-task weighted sampling, and can reason with demonstration sequence images with the trained networks to output pick-and-place actions.

planning errors. In recent works in compositional task learning, it has been shown that learning agents should leverage shared task structures [66, 165], and techniques such as sequence-conditioning [47], balanced sampling and learning curricula [86] can aid this process. Thus, learning systems should effectively reason with task structures to solve compositional tasks.

In this work, we propose a new suite of benchmark problems inspired by industrial compositional tasks, *MultiRavens*, which allows customizing compositional task problems with task modules. To effectively solve these compositional tasks, we then propose a system architecture that scales up to solve long horizon compositional manipulation tasks. Our vision-based end-to-end system architecture, *Sequence-Conditioned Transporter Networks*, uses sequence-conditioning and multi-task learning with weighted sampling to solve multi-task long horizon problems, leveraging the task structure and compositionality with little data.

Our results show that the new framework significantly outperforms original Transporter networks [154, 199] on 10 multi-task benchmark problems in *MultiRavens*, where each problem exemplifies the difficulties in vision-based learning and planning tasks. Furthermore, our results suggest that multi-task learning with weighted sampling can improve learning and agent performances on individual tasks. Beyond compositional tasks, our results provide insights for effective intermediate representations and learning shared structures.

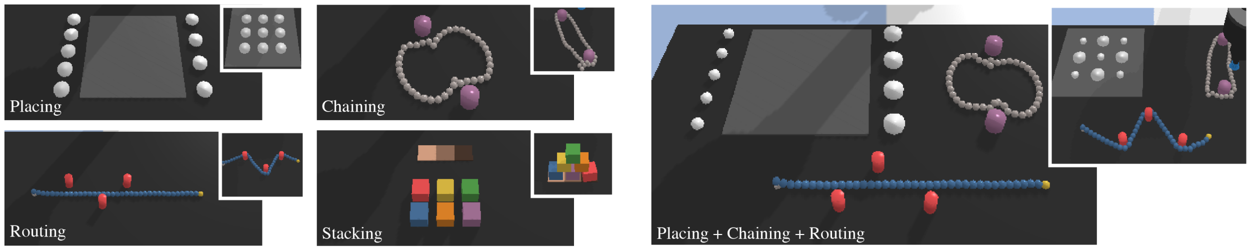


Figure 10.2: **MultiRavens.** Task modules introduced in MultiRavens: *placing, chaining, routing* and *stacking*, and their goal configurations (left). The task modules effectively capture the compositionality of industrial manipulation problems. An example multi-task problem *placing + chaining + routing* and its goal configuration (right).

10.2 Transporter Networks for Rearranging Objects

10.2.1 Problem Formulation

Consider the problem of rearranging objects as learning a policy $\pi : \mathcal{O} \to \mathcal{A}$ that maps the robot visual observations $\mathbf{o}_t \in \mathcal{O}$ to corresponding pick-and-place actions $\mathbf{a}_t \in \mathcal{A}$:

$$\mathbf{o}_t \mapsto \pi(\mathbf{o}_t) = \mathbf{a}_t = (\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}}) \in \mathcal{A}. \tag{10.1}$$

Both $\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}} \in \text{SE}(2)$ describe the geometric pose of the end effector when grasping and releasing the object, where SE(2) is the Special Euclidean group that covers all translations and rotations in the 2D xy-plane. For a given pick-and-place action pair, the end effector approaches $\mathcal{T}_{\text{pick}}$ until contact is detected, attempts to grasp the object, lifts up to a fixed z-height, approaches $\mathcal{T}_{\text{place}}$ to lower the object until contact is detected, and releases the grasp. This framework can be generalized further to many tasks solvable with two-pose motion primitives [199].

It is crucial that the policy π learns from a training procedure with closed-loop visual feedback, as many compositional tasks require sequential planning over long horizons while avoiding compounding error. The visual observations of the expert demonstrations are given from the top-down view of the planar surface, from which the picking and placing poses are sampled. We assume access to a small dataset $\mathcal{D} = \{\zeta_1, \zeta_2, \cdots, \zeta_N\}$ of N stochastic expert demonstrations, used to train π. Each episode $\zeta_i = \{(\mathbf{o}_1, \mathbf{a}_1), \cdots, (\mathbf{o}_{T_i}, \mathbf{a}_{T_i})\}$ of length T_i is a sequence of observation-action pairs $(\mathbf{o}_t, \mathbf{a}_t)$.

10.2.2 Transporter Frameworks

Transporter Networks [199] and *Goal-Conditioned Transporter Networks* (GCTN) [154] are model architectures that learn pick-and-place actions by using attention and convolution modules. For picking, the attention module predicts regions of placing interest given the

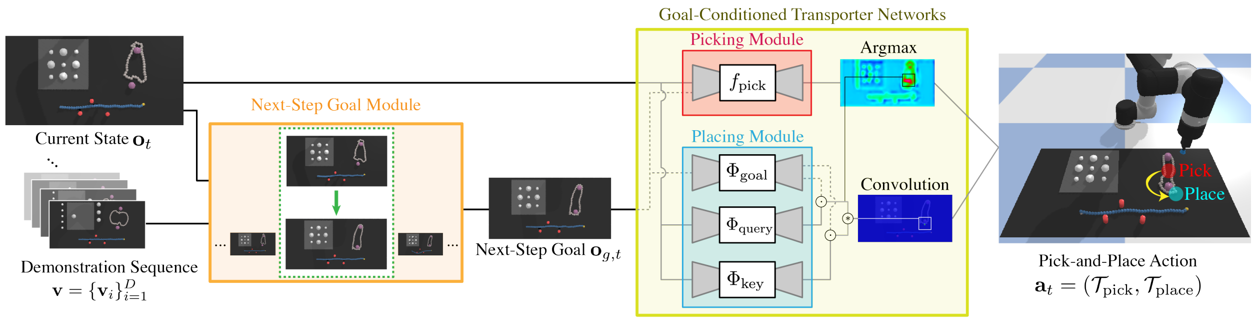

Goal-Conditioned Transporter Networks
Picking Module
f_{pick}
Argmax
Next-Step Goal Module
Placing Module
Φ_{goal}
Φ_{query}
Φ_{key}
Convolution
Current State $\mathbf{o}_t$
Demonstration Sequence $\mathbf{v} = \{\mathbf{v}_i\}_{i=1}^{D}$
Next-Step Goal $\mathbf{o}_{g,t}$
Pick
Place
Pick-and-Place Action $\mathbf{a}_t = (\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}})$

Figure 10.3: **Sequence-Conditioned Transporter Networks (SCTN).** Roll out of SCTN policy on *placing + chaining + routing*. SCTN takes in as input the current state $\mathbf{o}_t$ and the demonstration sequence $\mathbf{v}$. The next-step goal module determines the next-step goal $\mathbf{o}_{g,t}$ from $\mathbf{o}_t$ and $\mathbf{v}$, which is then fed into the GCTN module with $\mathbf{o}_t$ to obtain the pick-and-place action $\mathbf{a}_t$.

current observation. Then for placing, the convolution module uses the picking region to determine the spatial displacement via cross-correlating the image features over the observation and goal (for GCTN).

Both Transporter frameworks utilize the spatial equivariance of 2D visual observations, where the top-down observations are invariant under 2D translations and rotations [40, 99, 195]. Thus, the observations $\mathbf{o}_t$ are top-down orthographic images of the workspace, where each pixel represents a narrow vertical column of 3D space. The top-down images simplify reasoning with pick-and-place operations, as coordinates on the workspace can be directly mapped to the observation pixel space, and improving training data augmentation process [144], as translating and rotating the observation can quickly generate different configurations of the object. Thus, both Transporter frameworks have shown significant success in pick-and-place tasks, including stacking, sweeping, and rearranging deformable objects [154, 199].

GCTN consists of 4 Fully Convolutional Networks (FCNs): one FCN is used for picking module and three FCNs for placing module. All FCNs are hourglass encoder-decoder residual networks (ResNets) [73] with 43-layers and 9.9M parameters, made up of 12 residual blocks and 8-strides.

10.2.2.1 Picking Module

The picking module consists of one FCN, f_{pick}. It takes in as input the stacked image of the observation $\mathbf{o}_t$ and goal $\mathbf{o}_g$ images, and outputs dense pixel-wise prediction of action-values Q_{pick} that correlate with inferred picking success. This gives us the goal-conditioned picking action $\mathcal{T}_{\text{pick}} = \arg\max_{(u,v)}\{Q_{\text{pick}}((u,v)|\mathbf{o}_t, \mathbf{o}_g)\}$, with (u,v) as the pixel corresponding to the picking action determined by the camera-to-robot vision mapping.

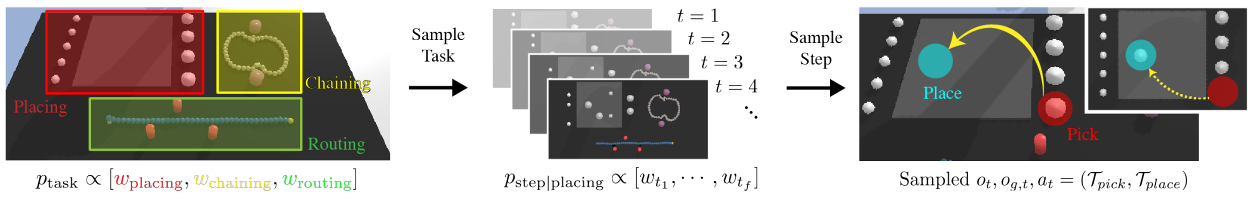

Placing
Chaining
Routing
Sample Task
$t=1$
$t=2$
$t=3$
$t=4$
Sample Step
Place
Pick
$p_{\text{task}} \propto [w_{\text{placing}}, w_{\text{chaining}}, w_{\text{routing}}]$
$p_{\text{step|placing}} \propto [w_{t_1}, \cdots, w_{t_f}]$
Sampled $o_t, o_{g,t}, a_t = (\mathcal{T}_{pick}, \mathcal{T}_{place})$

Figure 10.4: **Multi-Task Weighted Sampling Training.** For a selected demo, we first choose which task to sample from with probability p_{task}, then choose which step to sample from within the chosen task with probability p_{step}, getting $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{g,t})$.

10.2.2.2 Placing Module

The placing module consists of three FCNs, $\Phi_{\text{key}}, \Phi_{\text{query}}, \Phi_{\text{goal}}$. First, Φ_{key} and Φ_{query} take in the observation $\mathbf{o}_t$ to output dense pixel-wise feature maps of the inputs. Then, Φ_{goal} takes in the goal image $\mathbf{o}_g$ to also output a pixel-wise feature map, which is then multiplied to each of the outputs of Φ_{key} and Φ_{query} using the Hadamard product to produce pixel-wise correlated features ψ:

$$\psi_{\text{query}}(\mathbf{o}_t, \mathbf{o}_g) = \Phi_{\text{query}}(\mathbf{o}_t) \odot \Phi_{\text{goal}}(\mathbf{o}_g), \tag{10.2}$$
$$\psi_{\text{key}}(\mathbf{o}_t, \mathbf{o}_g) = \Phi_{\text{key}}(\mathbf{o}_t) \odot \Phi_{\text{goal}}(\mathbf{o}_g). \tag{10.3}$$

Finally, the query feature is cropped around the pick action $\psi_{\text{query}}(\mathbf{o}_t, \mathbf{o}_g)[\mathcal{T}_{\text{pick}}]$, then cross-correlated with the key feature to produce dense pixel-wise prediction of action-values Q_{place} that correlate with inferred placing success:

$$Q_{\text{place}}(\Delta\tau|\mathbf{o}_t, \mathbf{o}_g, \mathcal{T}_{\text{pick}}) = \psi_{\text{query}}(\mathbf{o}_t, \mathbf{o}_g)[\mathcal{T}_{\text{pick}}] \ * \ \psi_{\text{key}}(\mathbf{o}_t, \mathbf{o}_g)[\Delta\tau]. \tag{10.4}$$

The convolution covers the space of all possible placing poses through $\Delta\tau$, and gives us the goal-conditioned placing action $\mathcal{T}_{\text{place}} = \arg\max_{\Delta\tau}\{Q_{\text{place}}(\Delta\tau|\mathbf{o}_t, \mathbf{o}_g, \mathcal{T}_{\text{pick}})\}$.

10.3 Sequence-Conditioned Transporter Networks

We present our two main architecture contributions: (1) Sequence-Conditioned Transporter Networks (SCTN), which can effectively leverage structure and compositionality of multi-task problems through demonstration image sequences; and (2) multi-task learning with weighted sampling prior that improves learning robustness and efficiency.

10.3.1 Sequence-Conditioned Transporter Networks

We first determine what form of intermediate representation is suitable for *demonstration sequences*. One way to represent sequential information is demonstration sequence images [47, 193]. We collect demonstration sequence images at every time step t we receive a positive delta reward: whenever the expert demonstrator successfully places an object at a desired location or completes a part of a sequential task, we record the resulting observation. Thus, after a successful expert demonstration with D delta reward signals, we will have collected demonstration sequence images $\mathbf{v} \equiv \{\mathbf{v}_i\}_{i=1}^{D}$ that reaches a desired final goal configuration.

Next, we introduce the *next-step goal module*, a general framework for determining the next suitable goal given an observation $\mathbf{o}_t$. The next-step goal module seeks to find the closest delta reward time step in the demonstration image sequence, and infer the next-step goal as the demonstration image after the matched step. There are many possible implementations of this module: it could be a raw image distance-based module that compares the distance between the current observation $\mathbf{o}_t$ against the demonstration sequence images $\mathbf{v}_i$, or a neural network-based module that learns and compares temporal embeddings of $\mathbf{v}_i, \mathbf{o}_t$ [155].

In this work, we set the next-step goal module as an oracle operator that knows which delta reward time step the agent is in and outputs the next-step goal image from $\mathbf{v}$ during evaluation, for ease of implementation and to demonstrate proof of concept. This oracle operator is reasonable since in our real robot system, a human operator checks each step of the robot action and gives signal to continue or stop.

Thus, SCTN takes in as input the observation image $\mathbf{o}_t$ and the demonstration sequence $\mathbf{v}$ of the task being solved into a desired final configuration, determines the next-step goal image $\mathbf{o}_{g,t}$ from $\mathbf{v}$, and passes $(\mathbf{o}_t, \mathbf{o}_{g,t})$ into GCTN to obtain the pick-and-place action $\mathbf{a}_t$ (Fig. 10.3). We note that this is a general system architecture that could potentially work well with any discrete-time goal-conditioned agents.

10.3.2 Multi-Task Training with Weighted Sampling

We introduce next-step goal-conditioning, as well as task-level and step-level weighting that serve as useful sampling priors for stochastic gradient steps in training SCTN.

10.3.2.1 Next-Step Goal-Conditioning

To train the next-step goal-conditioning architecture, we sample $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{g,t})$ from a given demonstration episode, with $\mathbf{o}_{g,t}$ as the next-step goal image within the episode. This approach is based on Hindsight Experience Relay [7], where the goal image $\mathbf{o}_{g,t}$ is the first state with a delta reward signal after $\mathbf{o}_t$.

10.3.2.2 Weighted Sampling for Multi-Task Problems

To train SCTN, a demonstration ζ_i is randomly chosen from a dataset of N demonstrations $\mathcal{D} = \{\zeta_1, \zeta_2, \cdots, \zeta_N\}$, after which a step t of the demo needs to be sampled from the set of all time steps $t \in [1, \cdots, T_i]$ to give us the training sample $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{g,t})$. However, sampling uniformly across time steps can be problematic for multi-task problems that consist of tasks with varying difficulties and horizons.

Module	Precise Placing	Sequential Planning	Occluded Goal	Deformable Objects	Lengthy Task (6+)	Goal Objects n (Max. Steps)
placing	✓	✗	✗	✗	✓	9 $(n+2)$
stacking	✓	✓	✓	✗	✓	6 $(n+2)$
chaining	✗	✗	✗	✓	✗	2 $(n+2)$
routing	✗	✓	✗	✓	✗	2-3 $(n+4)$

Single-Task Problems	Double-Task Problems	Triple-Task Problems
1. *placing*	5. *placing + routing*	9. *placing + chaining + routing*
2. *stacking*	6. *stacking + routing*	10. *placing + stacking + routing*
3. *chaining*	7. *placing + chaining*	
4. *routing*	8. *placing + stacking*	

Table 10.1: **MultiRavens**. Overview of task modules and their attributes (top), and the 10 multi-task problems we test in this work (bottom).

We introduce weighted sampling on both task-level and step-level to leverage task structure information. For task-level weighting, we assign each task i in a multi-task problem a weight w_i. We use two weighting schemes: (i) uniform weighting ("Unif") that gives all tasks equal weights, and (ii) complexity weighting ("Comp") that weighs a given task proportional to its inferred task complexity. The inferred task complexity of a task is how many gradient steps it takes to reach optimal evaluation performance. Then for step-level weighting, we assign each step t of the demonstration corresponding for each task a weight $w_{i,t}$. We assume that the expert demonstration solves each task sequentially, which means there is no mixing of demo steps between tasks.

To determine the inferred task complexity and the step-level weighting for each task, we employ black-box optimization [63] to obtain the values, which are shown in Appendix E.3. With the weights determined, we can now enhance multi-task learning with weighted sampling. We first choose a task i with probability $p_{\text{task}} \propto \{w_i\}, i \in \{\text{tasks}\}$, then choose a step t with probability $p_{\text{step}|i} \propto \{w_{i,t}\}, t \in [t_{i,1}, \cdots, t_{i,f}]$ where $t_{i,1}$ and $t_{i,f}$ denote the first and last steps of the demo episode that pertains to solving task i. Fig. 10.4 depicts the weighted sampling process for training SCTN.

10.4 Simulator and Tasks

We evaluate the system on *MultiRavens*, a novel suite of simulated compositional manipulation tasks built on top of PyBullet [44] with an OpenAI gym [27] interface.

10.4.1 Modularized Tasks in PyBullet

To develop standardized benchmarks for compositional tasks, we propose a new PyBullet simulation platform, *MultiRavens*, based on Ravens [199] and DeformableRavens [154], which allows support for loading multiple tasks at once. PyBullet is a widely-used publicly available simulator for robotics research, and the prior Ravens frameworks support various discrete-time tabletop manipulation tasks. In MultiRavens, we introduce three new task modules: *placing, chaining* and *routing*; as well as modify one of the existing tasks from Ravens: *stacking*. The tasks take inspiration from the National Institute of Standards and Technology (NIST) Assembly Task Boards [54]. MultiRavens fully embraces compositionality of tasks by allowing full customizability of task modules in compositional tasks.

Even though the PyBullet implementations of these tasks are simplified, the four task modules capture the compositionality of industrial tasks while increasing the difficulty of pick-and-place reasoning compared to Ravens and DeformableRavens. The challenges include precisely placing without direct visual cues for long horizon, dealing with deformable objects with other static objects present, and planning around occluded goals without memorizing task sequences. Fig. 10.2 shows the four task modules and an example multi-task problem, and Table 10.1 provides a quick description for each *placing, routing, chaining* and *stacking*.

10.4.2 Benchmark for Multi-Task Pick-and-Place Problems

We benchmark our agents on 10 discrete-time multi-task tabletop manipulation problems from MultiRavens in Table 10.1. Each environment setup consists of a Universal Robot arm UR5 with a suction gripper, a 0.5×1m tabletop workspace where task modules are loaded, and 3 calibrated RGB-D cameras with known intrinsics and extrinsics, diagonally overlooking the workspace. The three RGB-D images are augmented to produce a single top-down observation $\mathbf{o}_t \in \mathbb{R}^{320\times160\times6}$, which has pixel resolution of 320×160 representing a 3.125×3.125mm vertical column of 3D space with 3 RGB channel and 3 channel-wise depth values. In principle, this camera setup can be replaced with a single top-down camera, but the image reconstruction method gives us a clear view of the workspace without the UR5 arm.

We use the same grasping and releasing motion primitives in the Transporter works [154, 199], which locks in the degrees of freedom of the end effector and the object using fixed constraints to pick and lift an object up. Furthermore, we use different motion primitive settings for different tasks, determined by which tasks we have completed up until a given step. For instance, for *routing*, we use a primitive setting that makes the arm move slowly at a low lifting height to reduce unnecessary cable movements and prevent undoing any previous successful routes. For *stacking*, we use a setting that lets the arm move quickly at a higher height to be able to stack blocks on top of each other.

10.5 Experiments

We use stochastic demonstrator policies to obtain $N = 1000$ demonstration data per problem. Each policy we benchmark below are then trained with 1, 10, 100 or 1000 demonstrations. For each problem, we test GCTN and SCTN, both with various weighted sampling schemes to analyze whether the system architecture of sequence-conditioning and weighted sampling helps with the performance. Training details, such as learning rate, optimizer, and data augmentation, remain the same as Transporter works [154, 199].

For single-task problems, we test the following four agents: GCTN, GCTN-Step, SCTN, and SCTN-Step. GCTN-Step and SCTN-Step are Transporter agents that are trained with step-level weighting for a given single-task problem. For multi-task problems, we test the following six agents: GCTN, GCTN-Unif-Step, GCTN-Comp-Step, SCTN, SCTN-Unif-Step, SCTN-Comp-Step. The agents with Unif-Step suffixes are trained with uniform task-level weighting and step-level weighting, and those with Comp-Step are trained with inferred task complexity task-level weighting and step-level weighting. The weighting schemes are given in Table E.1.

Zeng et al. [199] and Seita et al. [154] also test *GT-State MLP* models in their analyses, which use ground truth pose information of each objects in the scene and process the input with multi-layer perception (MLP) networks. However, two of our task modules, *chaining* and *routing*, have variable number of objects in the scene due to randomized chain and cable lengths. This renders the naïve ground truth-based models unapplicable to these problems. Other directly comparable vision-based discrete-time pick-and-place agents tested in Zeng et al. [199], such as *Form2Fit* [194] and *ConvMLP* [107], may potentially be made comparable to GCTN and SCTN. However, these agents are not goal-conditioned.

While it is possible to substantially modify these agents, these models are shown to perform rather poorly compared to the Transporter variants and will require nontrivial modifications to perform reasonably well on these tasks. Since we are mainly interested in analyzing the benefits of the new systems architecture of multi-task learning and sequence-conditioning, we focus our analysis on how different weighting and conditioning schemes affect the performance.

10.6 Results

We summarize results in Tables 10.2 and 10.3. For each agent and number of demo episodes $N \in \{1, 10, 100, 1000\}$, we train the models 5 times with different TensorFlow [1] random seeds, trained for 20K $\times\, m$ iterations with batch size of 1, where m is the number of task modules in a given problem. We evaluate the trained model every tenth of the training iterations, where the trained model snapshot is validated on a roll out of 20 evaluation episodes. This results in $20 \times 5 = 100$ metrics for 10 snapshots of the model. We evaluate agent performances based on completion rate, which aptly captures agents' abilities to reliably complete all tasks in a given episode. For completion rate, we assign the value of 1 for a given episode

	1. *placing*				2. *stacking*			
Method	1	10	100	1000	1	10	100	1000
GCTN	37.0	44.0	39.0	37.0	0.0	0.0	0.0	0.0
GCTN-Step	20.0	35.0	29.0	37.0	0.0	0.0	0.0	0.0
SCTN	**94.0**	**100.0**	**100.0**	**100.0**	0.0	47.0	53.0	**70.0**
SCTN-Step	77.0	100.0	98.0	80.0	**3.0**	**50.0**	**73.0**	67.0
	3. *chaining*				4. *routing*			
Method	1	10	100	1000	1	10	100	1000
GCTN	4.0	17.0	19.0	2.0	**13.0**	**66.0**	69.0	**75.0**
GCTN-Step	4.0	22.0	1.0	8.0	13.0	64.0	**78.0**	68.0
SCTN	**10.0**	26.0	**36.0**	27.0	1.0	10.0	12.0	15.0
SCTN-Step	7.0	**30.0**	28.0	**33.0**	1.0	11.0	10.0	10.0

Table 10.2: **Single-task results.** Completion rate vs. number of demonstration episodes used in training. The completion rate is calculated as mean % over 20×5 test-time episodes in simulation of the best saved snapshot, the # of demos used in training are chosen from $\{1, 10, 100, 1000\}$, and bold indicates the best performance for a given problem and # of demos.

if and only if all the task modules are completed, and otherwise 0. We average the completion rates over each snapshot and report the maximum over all snapshots in Tables 10.2 and 10.3.

10.6.1 Single-Task

For single-task problems, both variants of SCTNs outperform GCTNs in *placing* and *stacking*. SCTN has a remarkably high completion rate of 94% for *placing* even with just 1 demo episode, while GCTNs hover around 30-40%. GCTNs often suffer from two types of errors: inaccurately placing the disks, and stacking a disk on top of another one. In these scenarios, GCTN does not know how to properly correct itself since the top-down view is unable to capture the stacking mistake and the attention module mostly focuses only on the disks that are directly outside the square plate.

SCTN-Step also has a high completion rate of 50% with 10 demos and 73% with 100 demos for *stacking*, while GCTNs completely fail, which is expected as GCTNs cannot reason with blocks that are (partially) occluded from the top-down view. Interestingly, GCTNs successfully imitate the overall behavior of stacking the blocks but completely disregard the goal configuration (Fig. 10.5).

All Transporter agents perform similarly poorly on the *chaining* task, while GCTNs perform far better on *routing* with completion rate of 66% with only 10 demos. SCTNs often struggle at performing the final cable stretching move to match the goal configuration and seem to prematurely consider that it has already completed the task, then perform out-of-distribution moves that lead to failure. This seems to be an artifact of both overfitting and

	5. *placing + routing*				6. *stacking + routing*			
Method	1	10	100	1000	1	10	100	1000
GCTN	1.0 (4.8)	28.0 (29.0)	20.0 (26.9)	25.0 (27.7)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
GCTN-U	3.0 (2.6)	25.0 (22.4)	28.0 (22.6)	29.0 (25.2)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
GCTN-C	1.0 (2.6)	5.0 (22.4)	18.0 (22.6)	13.0 (25.2)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
SCTN	**5.0 (0.9)**	46.0† (10.0)	62.0† (12.0)	50.0† (15.0)	0.0 (0.0)	**35.0† (4.7)**	**54.0† (6.4)**	39.0† (10.5)
SCTN-U	4.0 (0.8)	**50.0† (11.0)**	**81.0† (9.8)**	**86.0† (8.0)**	0.0 (0.0)	25.0† (5.5)	45.0† (7.3)	**65.0† (6.7)**
SCTN-C	2.0 (0.8)	25.0† (11.0)	38.0† (9.8)	44.0† (8.0)	0.0 (0.0)	30.0† (5.5)	49.0† (7.3)	49.0† (6.7)

	7. *placing + chaining*				8. *placing + stacking*			
Method	1	10	100	1000	1	10	100	1000
GCTN	3.0 (1.5)	4.0 (7.5)	9.0 (7.4)	2.0 (0.7)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
GCTN-U	1.0 (0.8)	7.0 (7.7)	6.0 (0.3)	8.0 (3.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
GCTN-C	2.0 (0.8)	2.0 (7.7)	0.0 (0.3)	1.0 (3.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
SCTN	**4.0 (9.4)**	33.0 (26.0)	40.0 (36.0)	44.0† (27.0)	**1.0 (0.0)**	44.0 (47.0)	54.0 (53.0)	56.0 (70.0)
SCTN-U	3.0 (5.4)	**38.0 (30.0)**	**48.0† (27.4)**	**50.0† (26.4)**	1.0 (2.3)	57.0 (50.0)	**68.0 (71.5)**	51.0 (53.6)
SCTN-C	1.0 (5.4)	16.0 (30.0)	46.0† (27.4)	32.0 (26.4)	0.0 (2.3)	**58.0 (50.0)**	64.0 (71.5)	**62.0 (53.6)**

	9. *placing + chaining + routing*				10. *placing + stacking + routing*			
Method	1	10	100	1000	1	10	100	1000
GCTN	1.0 (0.2)	5.0 (4.9)	4.0 (5.1)	9.0 (0.6)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
GCTN-U	0.0 (0.1)	4.0 (4.9)	9.0 (0.2)	8.0 (2.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
GCTN-C	0.0 (0.1)	2.0 (4.9)	1.0 (0.2)	4.0 (2.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
SCTN	1.0 (0.1)	**27.0† (2.6)**	33.0† (4.3)	27.0† (4.0)	0.0 (0.0)	21.0† (4.7)	35.0† (6.4)	41.2† (10.5)
SCTN-U	**2.0 (0.1)**	26.0† (3.3)	**49.0† (2.7)**	**53.8† (2.6)**	0.0 (0.0)	**42.5† (5.5)**	44.0† (7.2)	**48.7† (5.4)**
SCTN-C	0.0 (0.1)	21.0† (3.3)	32.0† (2.7)	39.0† (2.6)	0.0 (0.0)	23.0† (5.5)	**49.0† (7.2)**	29.0† (5.4)

Table 10.3: **Multi-task results.** Completion rate vs. number of demonstration episodes used in training. The completion rate is calculated as mean % over 20×5 test-time episodes in simulation of the best saved snapshot, the # of demos used in training are chosen from $\{1, 10, 100, 1000\}$, and bold indicates the best performance for a given problem and # of demos. The shorthand "-U" denotes the Unif-Step weighting and "-C" denotes the Comp-Step weighting. Gray number in parentheses indicates expected performance of combining single-task Transporter agents, and dagger (†) indicates more than 10% completion rate improvement compared to the expected performance.

task complexity, as efficient routing demonstration steps can overshadow the complexities of dealing with linear deformable objects.

Thus, for single-task problems, SCTNs perform far better and more efficiently than GCTNs on tasks that deal with small individual objects over long horizons, but does not perform better than GCTNs on tasks that deal with deformable objects. On a single-task level, the step-level weighting does not seem to improve the performance significantly.

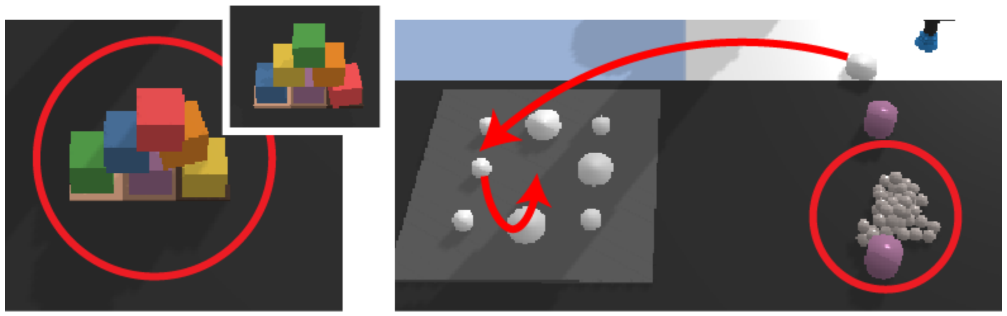


Figure 10.5: **Shortcomings of GCTN:** GCTN can imitate the block stacking behavior but mostly disregards the goal configuration (left), and makes several misplacing mistakes in multi-task problems (right).

10.6.2 Multi-Task

For all multi-task problems, SCTNs vastly outperform GCTNs. We also report the *expected completion rate* in Table 10.3, which is the completion rate we can expect if we use separate Transporter agents of same settings for each task in the multi-task problem, calculated by multiplying the mean completion rates for each task. SCTNs with weighted sampling often performed far better on multi-task problems than the expected completion rate. For *placing + routing*, SCTN-Unif-Step has a completion rate of 81%, while the expected completion rate is only 9.8%, which means SCTN performs better on the *routing* task when jointly trained on *placing* and *routing* than when solely trained on *routing*.

This trend continues to show in triple-task problems, where SCTN-Unif-Step achieves 49.0% completion rate with just 10 demos for *placing + chaining + routing*. Meanwhile, GCTN often makes several mistakes along the way (Fig. 10.5). Thus, for multi-task problems, both sequence-conditioning and weighted sampling significantly improve the performance of the Transporter agents, often improving far beyond the expected performance when trained on multiple tasks.

10.7 Conclusions

We introduce a suite of modularized tasks for benchmarking vision-based manipulation agents on multi-task problems. In addition, we propose an architecture that enables discrete-time goal-conditioned pick-and-place agents to reason with sequence-conditioning and allows for more informed weighted sampling to tackle multi-task problems of long horizons.

Reasoning over multi-task problems is challenging, and prior works have shown mixed success results in multi-task training. Our proposed system demonstrates high completion rates in many multi-task problems of long planning horizon when trained on as few as 10 demonstration episodes, and shows that sequence-conditioning with appropriate sampling schemes can bolster multi-task learning and planning. Despite the promising results, it still remains a question how the new system architecture generalizes to other goal-conditioned agents. Furthermore, while access to robots was limited due the COVID-19 pandemic, we aim to conduct physical experiments with suitable multi-task problems to see how well the system performs in realistic scenarios.

Part IV

Ecology Application: Novel Ecological Applications of Optimal Path Planning

Chapter 11

Overview

Chapter Outline

In this chapter, we give an overview of the work discussed in Part IV. We introduce the ecological species coexistence navigation problem and how to approach it with sequential decision making methods. Some discussions, results, and relevant figures are borrowed & repeated from the original works listed in each chapter.

11.1 Applications of Sequential Decision Making to Ecological Community State Navigation

This part of the dissertation aims at applying the principles from Parts II and III to other promising domain intersections. One such domain to apply and extend sequential decision making techniques to is ecology, specifically community state navigation, which we focus on in this part of the dissertation. Ecological problems often require reasoning about how populations dynamically evolve, how to effectively control and leverage such dynamics, and design experiments and policies that are both effective and the least harmful to the environment. In particular, using theoretically principled and justified algorithms is of practical importance in ecological settings for species coexistence navigation, as practitioners are often interested in algorithms and techniques that are scientifically justified with guarantees (Part II), and can leverage the prior knowledge and modern techniques to aid these planning techniques (Part III). Generating near-optimal plans with guarantees is of utmost importance in ecology, where experiments can be performed and decisions can be made on a time scale of years and decades, such as in the case of annual plant growth, forest management, and wildlife preservation.

Consequently, as the next step of Parts II and III, we consider how to effectively apply these techniques to ecological problems. In particular, this part of the dissertation dives deep into solving the species coexistence navigation problem, where we want to drive a population state from one configuration from another. The state diagram approach [22] introduced in Chapter 12 outlines how we can formulate an ecological problem into a sequential decision making problem, by generating a graph of possible transitions and searching for the optimal path.

11.2 Species Coexistence Navigation

In Chapter 12, we introduce the state diagram approach, which enables navigation between initial and desired community states using shortcuts [22]. Ecological management problems often involve navigating from an initial to a desired community state. We ask whether navigation without brute-force additions and deletions of species is possible via: adding/deleting a small number of individuals of a species, changing the environment, and waiting. Navigation can yield direct paths (single sequence of actions) or shortcut paths (multiple sequences of actions with lower cost than a direct path). We ask (1) when is non-brute-force navigation possible?; (2) do shortcuts exist and what are their properties?; and (3) what heuristics predict shortcut existence? Using a state diagram framework applied to several empirical datasets, we show that (1) non-brute-force navigation is only possible between some state pairs, (2) shortcuts exist between many state pairs; and (3) changes in abundance and richness are the strongest predictors of shortcut existence, independent of dataset and algorithm choices. State diagrams thus unveil hidden strategies for manipulating species coexistence and efficiently navigating between states.

Chapter 12

Navigation Between Initial and Desired Community States Using Shortcuts

Chapter Outline

This chapter is based on the paper "Navigation between initial and desired community states using shortcuts", written in collaboration with Benjamin W. Blonder, Claire Tomlin, and Zachary Sunberg.

12.1 Introduction

Many ecological management problems involve observing a community in an initial state, then taking a sequence of actions to yield a desired state (e.g. promoting gut microbiome health after infection, restoring a degraded rangeland). Management problems are often solved by brute-force navigation, which involves removing all individuals of undesired species and adding many individuals of desired species at great effort (e.g., antibiotics+probiotics, bulldozing+replanting). Such navigation may succeed, but at high cost and impact. Alternatives may exist that are more efficient and have fewer side effects. The challenge is to identify action sequences, i.e. navigation, that yield the desired state at lower cost and effort.

Some prior navigation approaches focused on continuous control of community dynamics. The problem has recently been conceptually explored in models [8, 18, 26, 82]. Applications exist for e.g., fisheries, forestry, agriculture, and other natural resource management challenges where continual intervention is of interest [24, 100, 102, 138], and also in microbial systems where metabolite production or infectious disease are priorities [8, 42, 60]. However, continuous control of multiple species' abundances becomes mathematically prohibitive and biologically unrealistic in high-richness communities.

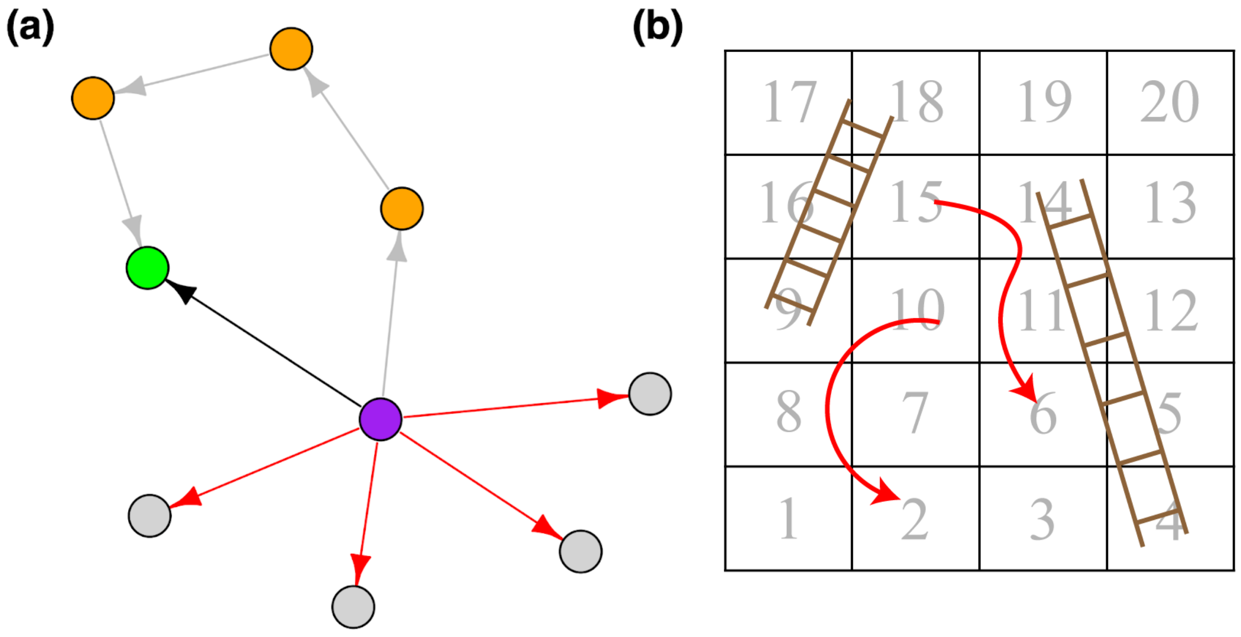


Figure 12.1: **Ecological Navigation Problem.** (a) Navigation is the problem of discovering sequences of actions that shift a community from an initial state (purple circle) to a desired state (green circle) while not unnecessarily visiting other states (gray circles). A direct path (black arrow) involves taking a single low-cost action. A shortcut path (gray arrows connecting orange circles) involves taking several low-cost actions, and represents a 'work-with-nature' solution. A brute-force solution (red arrows) involves deleting all individuals of all undesired species and adding many individuals of all desired species. (b) Navigation is loosely analogous to playing 'Snakes and Ladders'. In this game, players transition between squares (states) through sequential movement (actions) that either progress along the board (direct path) or jump around via snakes or ladders (shortcut paths).

We instead propose a discretized navigation approach that focuses on control of community outcomes. Many management problems can be simplified to coexistence outcomes [23, 39, 120]. Reaching an outcome (desired state) may be more important than the transient dynamics. Formulation as a discrete path planning problem can reduce mathematical complexity and improve biological realism.

Our hypothesis is that the internal dynamics of a community enable taking actions that nudge a community between states, either through direct paths or shortcut paths, both of which are lower-effort than brute-force navigation. Here, we define a direct path as a single set of low-effort actions that yield the desired state, and a shortcut path as a sequence of multiple sets of low-effort actions that yield the desired state; action sets are separated by waiting for the community to reach a feasible and stable fixed point. The term 'shortcut' is used to indicate that path cost is small, not path length (Fig. 12.1 (a)). Conceptually, we nudge a community until it tips into an alternate basin of attraction, then repeat this nudging process until the desired state is reached. Several small nudges may be lower in cost than a single large push into the desired state.

Brute-force navigation is always theoretically possible between states by removing all individuals of undesired species and then adding a sufficient number of desired species, ignoring the internal dynamics of the community. However, brute-force is often impractical, and if the desired state is not feasible and stable, further continuous effort would be needed to maintain the state. We focus therefore on finding direct and shortcut paths between feasible and stable states only. Direct paths may be findable via trial-and-error. However, shortcuts are difficult to find because of the near-infinite numbers of potential action sequences to explore.

The navigation problem is loosely analogous to the game of 'Snakes and Ladders' (known originally as 'gyan caupar') (Fig. 12.1 (b)) [177]. In this game, "the player should complete [the tour of] the board according to its numbering, starting at birth and ending at liberation. Going upward comes about by means of the ladder; going down comes about from the body of the snake. Going up is achieved from good actions; [going down from] the face of the snake is caused by bad actions. Vaikuntha [the heaven of Visnu] is reached by completing the game; otherwise the player must go on climbing" [71]. While Snakes and Ladders is a game of chance, not choice, our hypothesis is approximately equivalent to finding and then using 'snakes' (richness-decreasing shortcuts) and 'ladders' (richness-increasing shortcuts) to navigate between states.

First, we show how to enumerate a state diagram characterizing all of the possible transitions between all possible fixed point states. We consider actions that include adding ϵ individuals of a certain species, deleting ϵ individuals of a certain species, changing the environment, or waiting. The first three actions are assumed to occur instantaneously, shifting the community into a transient state, while the last action takes time, shifting the community to a fixed point. Each type of action i is also assumed to have a different cost C_i. We then show how to identify shortcuts on the state diagram for arbitrary pairs of initial and desired states.

We apply the approach to six empirical parameterizations of the generalized Lotka-Volterra model, varying in taxonomy and species pool richness. We use these data to ask: (1) when is navigation between states possible without using brute-force; (2) are shortcut paths common, and what are their characteristics; and (3) are shortcuts predictable based on dataset or initial/desired community properties?

12.2 The State Diagram Approach

12.2.1 The state diagram

There is a set of n species comprising a regional pool, of which any subset may co-occur locally in the community. The state of the community, $X = \{X_i(t)\} \in \mathbb{R}^n_{\geq 0}$, is defined as the vector of abundances of each species X_i $(1 \geq i \geq n)$ at a time t. There is a set of discrete environments with cardinality m defined by $\{E_j\} \in M$ with $1 \leq j \leq m$. Note that the environment may actually be continuous; here we simply consider some discrete points within the environment to be reachable by actions, e.g., to model cases where an experimentalist could select among 'cold' to 'warm' and 'hot' conditions ($m = 3$). There is a dynamical model that predicts temporal changes in the state as a function of variables, which may include X and E, $\frac{dX(t)}{dt} = f(X(t), E)$.

Based on this dynamical model, there are a set of fixed points with cardinality Ξ, $\{\xi_k\}$ with $1 \leq k \leq \Xi$, defining the points where $\frac{dX(t)}{dt} = 0$. Note that if E changes, so too may Ξ. A fixed point k can have an attribute $f_s(\xi_k)$ indicating that it is feasible (i.e. all species present i occur at non-negative abundances; $\xi_{k,i} \geq 0$) and stable (for every small $\varepsilon > 0$ there exists a $\delta > 0$ such that if $|X(t_0) - \xi_k| < \delta$ then $|X(t) - \xi_k| < \varepsilon$ for all $t \geq t_0$).

We next enumerate $\{\xi_k\}$ over all combinations of species being present or absent in the community (i.e. the empty community, all species occurring alone, all pairs, all triplets, etc.). These fixed points can be identified by exploring every subspace of the state space (all combinations of presences/absences), then re-calculating dynamical model nullclines.

We consider four types of actions, indexed $1 \leq q \leq 4$. Each action type q is assumed to have some cost $C_q \geq 0$ and have a different consequence: ($q = 1$) adding a small number (ϵ) of individuals of a single species i (i.e. $X_i \to X_i + \epsilon$) ; ($q = 2$) deleting a small number (ϵ) of individuals of a single species i (i.e. $X_i \to \max(X_i - \epsilon, 0)$) ; ($q = 3$) changing the state of the environment j to j, $1 \leq j^* \leq m$ ($E_j \to E_{j^*}$), no change to X), and ($q = 4$) waiting for a shift into fixed point $k^*, 1 \leq k^* \leq \Xi$ ($X \to \xi_{k^*}$), no change to E). Each species i can either be added or deleted up to one time until a waiting action has been performed. For all actions except waiting, the consequence is assumed to occur instantaneously; for waiting, the consequence is assumed to occur as $t \to \infty$ and determined by the dynamical model. That is, we assume that states do not reach a fixed point until a waiting action, and that multiple non-waiting actions can be taken in sequence before waiting.

The system can now be discretized into a smaller state space $\{Y\}$ that describes fixed points and transient points. In each environment E, we therefore assume that each state can either be at one of the fixed points ξ_k or, for each fixed point, at one of the 3^n possible transient ϵ-addition or ϵ-deletion states that occur immediately after any number of actions is taken. The overall cardinality of the discretized state space $\{Y\}$ is therefore $(m \times \Xi \times 3)^n$ or approximately $m \times 2^n \times 3^n$ assuming one fixed point per species combination. The action space can also be discretized. There are a total of n ϵ-additions and ϵ-deletions, m environmental changes, and 1 wait action, yielding a cardinality of $2n + m + 1$. Each action, now by definition, yields a transition from a state in $\{Y\}$ to another state in $\{Y\}$.

With this information for fixed points and the outcomes of actions at each fixed point, we can construct a directed graph called the state diagram. Vertices are states in $\{Y\}$ and edges are actions, where the arrow head is the state after the action and the arrow base is the state before the action. Each vertex k has attribute $f_s(\xi_k)$; each edge δ has an attribute C_q. We define an action sequence $\Delta = \{\delta_1, \delta_2, \ldots\}$ as an ordered set of edges (actions) that connects an initial vertex (state) to a desired vertex (state), with associated cost sequence $\omega = \{C_{q,1}, C_{q,2}, \ldots\}$.

Our primary insight is that the navigation problem is now equivalent to a shortest-path (lowest-cost) problem on a directed graph (the state diagram), i.e. finding a Δ that minimizes $\sum \omega$. This general mathematical problem can be solved efficiently [36, 57]. If a path does not exist, the only solution is brute-force; if a path does exist, and has one wait action, it is direct, and if it has more than one wait action, it is a shortcut.

12.2.2 Implementation

We implemented the state diagram approach for the GLV model, which has been widely studied to explore questions of species coexistence [17, 149] and can accommodate cases where the environment influences parameter values [180]. The dynamical model is

$$\frac{dX(t)}{dt} = \text{diag}\left(X(t)\right)\left(r(E) + A(E)X(t)\right), \tag{12.1}$$

where E is assumed constant unless changed by an action. Here, $r\left(E\right)$ is a $n \times 1$ vector that indicates the intrinsic growth rates of each species, and $A\left(E\right)$ is a $n \times n$ matrix whose i, j entry represents the change in species i's per-capita growth rate for a unit change in the density of species j.

If A is non-singular, for each parameter combination, there is one non-trivial fixed point, determinable by nullcline analysis:

$$\xi = -A^{-1}(E)r(E). \tag{12.2}$$

If the fixed point is not feasible, the state will shift to a subspace with some species absent (see below). If A is singular, there can be many fixed points corresponding to the null space of A, corresponding to cases where parameters are either linear combinations or there is partitioning in the interaction network [8]. Stability is defined by the criterion

$$\max_i \left[Re\left(\left\{\lambda_i(E)\right\}\right)\right] < 0, \tag{12.3}$$

where $\left\{\lambda_i(E)\right\}$ are the n eigenvalues of $A\left(E\right)$, and feasibility is determined based on the values of ξ.

To then calculate all the fixed points Ξ, the process can be iterated for all combinations of species. Because all GLV interactions are pairwise, cases where species j is absent can be handled by dropping row j and column j of the A matrix (i.e. obtaining the principal submatrix), and simultaneously dropping entry j of the r vector. Multiple entries can be dropped in cases where multiple species are absent. This is non-trivial, as the eigenvalues of a principal submatrix (which are closely related to the matrix inverse, and thus the location of the fixed point) are not necessarily the same as for the original matrix [81]. That is, combinations of species may behave differently from subsets of those combinations [149], a phenomenon also seen in models with higher-order interactions [119]. If A and all its principal submatrices are non-singular, then there is a single fixed point per iteration, yielding $\Xi = 2^n$

Dataset	**Species** (n)	**Envs** (m)	**Number of edges in state diagram**	**Proportion of feasible & stable**	**Abundance**
Ciliate	5	1	27 ± 2	0.25	2.75 ± 1.48
Ciliate+env3	5	3	2851 ± 1562	0.9	0.07 ± 0.02
Ciliate+env5	5	5	5667 ± 2820	0.74	0.61 ± 1.26
Human gut	12	1	7132 ± 219	0.05	2.08 ± 8.35
Mouse gut	11	1	22065 ± 1590	0.24	6.1 ± 46.83
Protist	11	1	408 ± 39	0.02	8.29 ± 65.82

Table 12.1: **Ecological Navigation Empirical Dataset.** Summary of properties for empirical datasets used in this study. Abundance values are summarized across all assemblages and then across all experimental conditions. The number of edges in the state diagram are summarized across all ϵ values.

fixed points for each value of E. If A is singular, there may be more or fewer fixed points to be considered.

The outcomes of actions are determined based on numerical integration of the dynamical model. First, we enumerate all desired addition, deletion and environment change actions for each fixed point, arriving at intermediate states. Then, when the waiting action is performed, the initial condition of the numerical integration is set to the intermediate state, and the dynamics are run forward with integration time span proportional to the smallest eigenvalue of A to ensure that the system can approach equilibrium. The resulting final abundances are then matched to the corresponding fixed point if the integration is successful and results in a non-trivial fixed point.

We calculate Δ and $\sum \omega$ for pairs of starting and desired states using A* search, which is a best-first search algorithm that expands local paths around the source vertex according to a combination of the cost of the path from the initial vertex to the current vertex plus the cost of a heuristic estimate of the cost from the current vertex to the desired vertex. It is guaranteed to find a solution if one exists [72]. We use an admissible heuristic that optimistically assumes that a single round of adding small numbers of individuals of currently missing species followed by a waiting action is sufficient to reach the desired basin of attraction. All algorithms were implemented in Julia (version 1.6.0). ODEs were solved using Rodas4P with absolute tolerance 10^{-6}, relative tolerance 10^{-6}, and max iterations 10^3.

12.2.3 Empirical parameterization

We studied six cases where parameter estimates for A and r come from fitting generalized Lotka Volterra models to empirical data (Table 12.1, taxon names in Table F.1). These comprise: ('Ciliate') a $n = 5$ protozoan ciliate community [120] based on data for 19 °C growth; ('Ciliate+environment3', often abbreviated as 'Ciliate+env3') the above $n = 5$ community for $m = 3$ environments: 15, 19, and 23 °C growth from [140], ('Ciliate+environment5', often abbreviated as 'Ciliate+env5') as above for $m = 5$ environments also including 17 and 21°C growth; ('Human gut') a $n = 12$ $m = 1$ synthetic human gut microbial community [181]; ('Mouse gut') a $n = 11$ $m = 1$ mouse gut microbial community including the difficult-to-remove pathogen *Clostridium difficile* [169] based on data from [30]; and ('Protist') a $n = 11$ $m = 1$ protist and rotifer community based on A values from [32] and r values from [31] and supplemented by additional r values for two missing taxa via personal communications.

12.2.4 Computational experiments

We performed A* experiments over all multiple action cost combinations and action magnitudes. Addition and deletion actions used ϵ in $\{10^{-1}, 10^{-3}, 10^{-5}\}$. Each type of action q used costs in $\{10^{-1}, 10^{0}, 10^{1}\}$. We also tested whether capping the total number of actions before a wait (a scenario where actions should be simple to implement) influenced navigation. This comprises $3 \times 3^4 \times 2 = 486$ experiments per dataset. Impacts of capping were minimal so main-text results only consider no capping, with capped results provided in output files. For each dataset, we picked 10,000 random pairs of initial and desired states. We determined whether a non-brute-force navigation solution existed for each dataset for 10,000 subsampled state pairs for which both start and end states are feasible and stable. State pairs were sampled without replacement using a fixed random number generator seed within each dataset to enable direct comparison between experimental results with different hyperparameter choices.

12.2.5 Statistical analysis

To address Question 1, for each A* experiment, we determined whether non-brute-force navigation was possible via any path. We also visualized state diagrams for selected cases, and determined whether, across cases, some intermediate states were more commonly visited (i.e. variation in node degree and centrality).

To address Question 2, for each A* experiment where non-brute-force navigation was possible, we determined whether the lowest-cost path was direct or a shortcut. We assessed variation in path length, and also visualized paths for selected cases.

To address Question 3, we built a random forest model that outputs probabilities, where path type (brute-force, direct, shortcut) was the dependent variable. Predictor variables reflected several easily-measured state properties, assuming no knowledge about the state diagram or the GLV dynamics: change in mean abundance, richness, and Jaccard similarity between initial and desired states; $\log_{10} \epsilon$; n; m; all four costs C_q; and dataset name. To reduce computational costs, a subset of 100 (or the maximum available) state pairs were randomly sampled from each of the 2916 A* experiments, after which we balanced the sampling by path type (brute-force, direct, shortcut) to the minimum number of samples available in each type. The final dataset comprised 25,782 cases. We used default parameters in the ranger package (version 0.14.1). We calculated a permutation importance for each predictor, made partial dependence plots for the most important predictors, and calculated

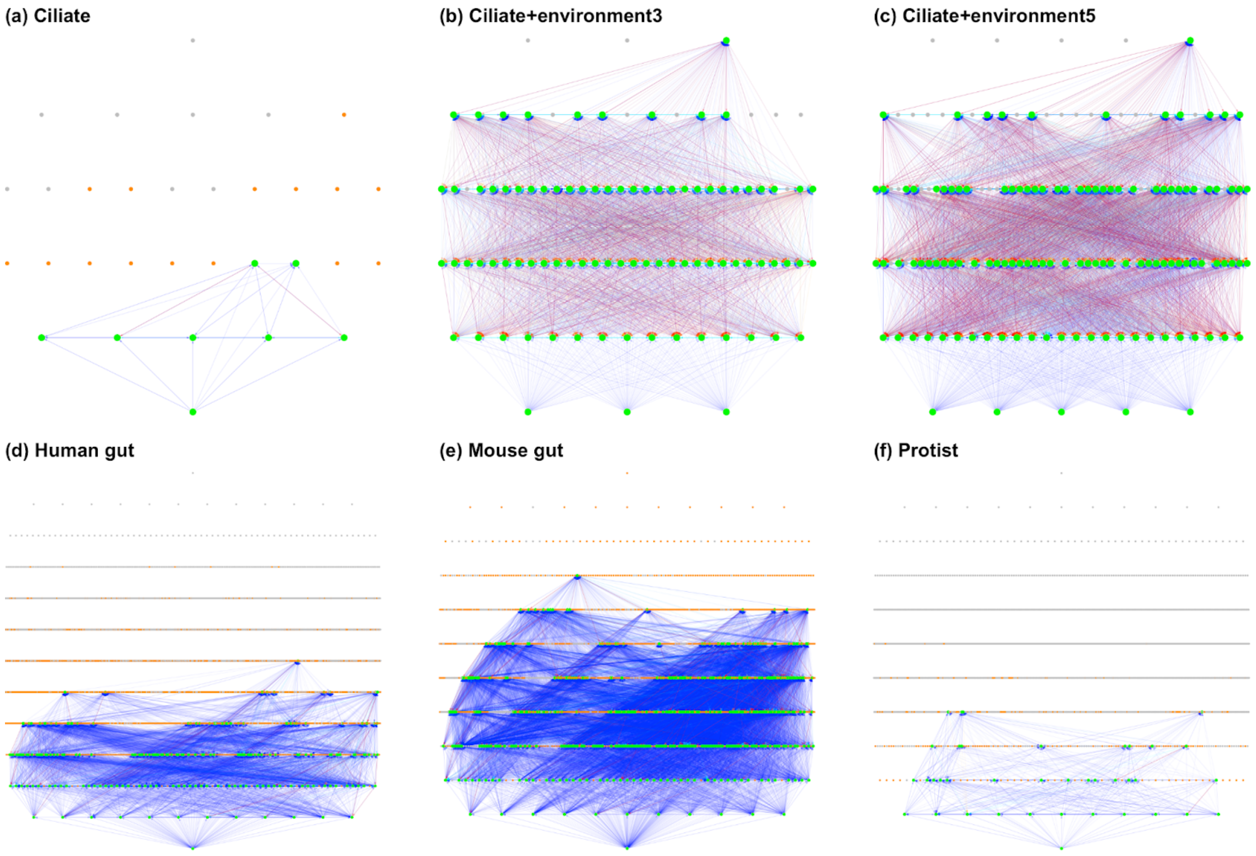


Figure 12.2: **State Diagrams.** Example state diagrams for all datasets. Circles represent fixed point states and are colored green if feasible and stable (i.e. possible navigation target), and gray if not. Orange circles indicate transient states that have higher richness than their pre-action state. States are arranged by richness on the y-axis, with the empty state at bottom and the maximum richness state at top. Arrows indicate actions; redder arrows are primarily deletions, while bluer arrows are primarily additions, and intermediate colors indicate mixtures of both additions and deletions; arrow thickness indicates inverse action cost (thicker = lower cost). Panels (b) and (c) indicate cases where there are multiple environments. For visual presentation, environment-changing actions are not separately colored, and states are not ordered by environment (this is why there is more than one state shown at minimum/maximum richness). Visualizations are for $C_{\epsilon-addition} = 1, C_{\epsilon-deletion} = 1, C_{environment} = 1, C_{wait} = 0.1$, and $\epsilon = 0.1$. See Fig. F.2 for the 'ladder' path subset (the invasion graph) and Fig. F.3 for the 'snake' path subset (the un-invasion graph).

overall accuracy using a 10-fold cross-validation in the caret package (version 6.0-93). All analyses were performed in R (version 4.2.0).

12.3 Results using Experimental Data

12.3.1 Question 1: Navigation

State diagrams had complex topologies that varied widely with dataset (Fig. 12.2). Some datasets only contained a small fraction of feasible and stable states, limiting non-brute-force navigation among low richness states (e.g., protist), while others supported navigation to high richness states (e.g., mouse gut). Higher-richness transient states used for navigation occurred widely in all datasets, indicating that species interactions, here competitive exclusions, played a key role in navigation, but also represented a potential hazard if they would be unsafe to reach (see Discussion). Actions were dominated by additions in some datasets (e.g., human gut, mouse gut) and by deletions in others (e.g., ciliate+environment3, ciliate+environment5), though actions comprising both additions/deletions also occurred (Fig. F.1).

Varying the GLV parameterization influenced the state properties, and thus the possible navigation targets. Varying ϵ changed the topology of the state diagram, with larger ϵ often resulting in more edges, but sometimes loss of edges (Fig. F.4). For a fixed state diagram topology, varying the costs C_q influenced the edge weights and thus the navigation paths.

Navigation probabilities, defined as the number of state pairs connected by a non-brute-force path divided by the number of feasible and stable state pairs, varied widely (Fig. 12.3 (a)). Probabilities were lowest for the human gut and highest for the ciliate+environment5 dataset. Increasing ϵ increased probabilities for all datasets. Some intermediate states were consistently visited (Fig. F.5), showing that there are hubs on the state diagram. However, hubs were not common in the ciliate+environment datasets, suggesting that environmental variation enables more diverse navigation pathways. Hubs were not correlated with in-degree or out-degree on the state diagram (Fig. F.6). In general, there was a tradeoff between in- and out-degree, indicating that states that are easier to reach are harder to leave, and vice versa.

12.3.2 Question 2: Shortcut properties

Shortcut probabilities, defined as the probability a state pair was connected by a shortcut, conditioned on navigation between the states being possible, also varied substantially (Fig. 12.3 (b)). Shortcut probabilities ranged from 14% to 71% across datasets and ϵ values, except for the ciliate dataset at 0%. Increasing ϵ did not consistently increase shortcut probability.

Among shortcut paths, the number of steps varied widely (Fig. F.7). The mouse gut and ciliate+environment datasets consistently had the longest path lengths, some involving as many as eight sequential actions, which is consistent with the greater number of links present in their state diagrams (Fig. 12.2). Other datasets typically involved paths comprising 2-4 actions.

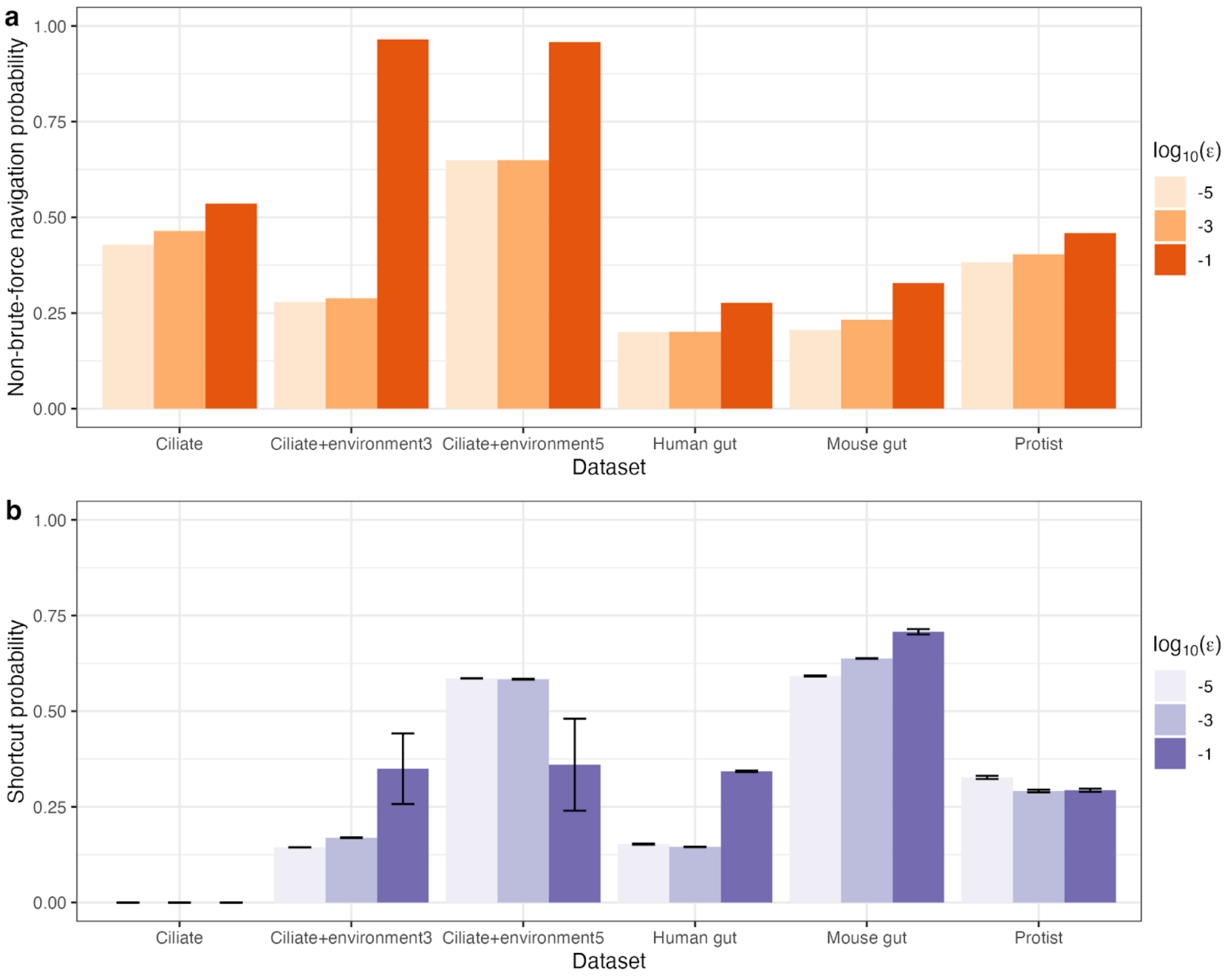


Figure 12.3: **Navigation Statistics.** (a) Probability that non-brute-force navigation is possible between two randomly selected feasible and stable states. (b) Probability that a shortcut path exists between two randomly selected states, given that navigation is possible. Bars indicate different datasets and are colored by ϵ. Error bars in panel (b) indicate standard deviations across assumed costs C_q; no error bars are shown in (a) because costs do not influence estimates.

Visualizing shortcut paths illustrates the complexity of navigation. In the mouse gut, completely removing the pathogen *C. difficile* when it is initially present was often possible. For the experimental conditions described in Fig. 12.2, we found 4,304/10,000 cases with the pathogen present; of these, a complete removal via shortcut was possible in 111 cases. Two examples are shown in Fig. 12.4 (a-b). Similarly, community turnover is often achievable by leveraging environmental change, as in the ciliate+environment5 dataset. Also for the experimental conditions described in Fig. 12.2, we found 716/10,000 cases that had no net change from 15°C growth; of these, reduction in richness via shortcuts leveraging environ-

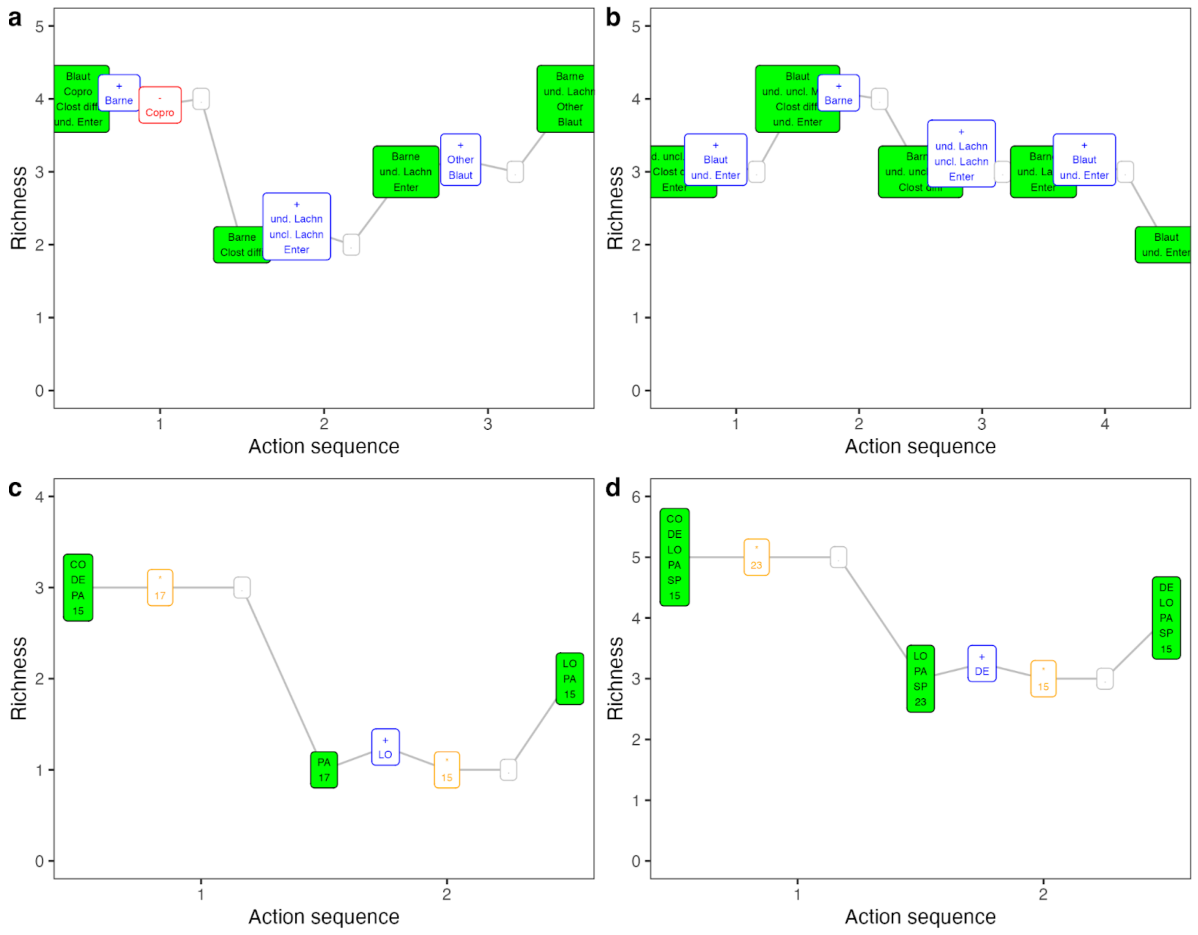


Figure 12.4: **Example Shortcut Paths.** Example shortcut paths for (a-b) completely removing the pathogen *Clostridium difficile* in the mouse gut dataset, and (c-d) reducing species richness via environmental change in the ciliate+environment5 dataset. Each panel indicates states connected by sequential actions on the x-axis, ordered by richness on the y-axis. Green boxes indicate feasible and stable states, with the initial state on the left and the desired state on the right. White boxes indicate actions, with ϵ-additions as blue '+', ϵ-deletions as red '-', environment changes as orange '*', and waits as gray '.'. Visualizations are for $C_{\epsilon-addition} = 1, C_{\epsilon-deletion} = 10, C_{environment} = 1, C_{wait} = 0.1$, and $\epsilon = 0.1$ (i.e. $10\times$ more costly to ϵ-delete than ϵ-add) Taxon names are in Table F.1. (a) 1: One species is introduced at low density and another is given a small negative abundance perturbation, causing the establishment of one species and the competitive exclusion of three others. 2: Three species are introduced at low density, causing the competitive exclusion of *C. difficile*. 3: Two species are introduced at low density, yielding competitive exclusion of one species and coexistence of four species in the desired state. (b) 1: Two species are introduced at low density, causing one competitive exclusion. 2: One species is introduced at low density, causing two competitive exclusions. 3: Three species are introduced at low density, causing the competitive exclusion of *C. difficile* and two other species. 4: Two species are introduced at low density, yielding competitive exclusion of three species and coexistence of two species in the desired state. (c) 1: The environment is warmed, causing competitive exclusion of two species. 2: One species is introduced at low density and the environment is cooled, causing coexistence of two species in the desired state. (d) Similar to (c).

mental change was possible in 206 cases. Two examples are shown in Fig. 12.4 (a-b). In all cases, navigation used timely actions to cause useful competitive exclusions, which allowed jumping between states until the desired state was reached. In other cases (not shown) where $C_{\epsilon-deletion}$ is assumed smaller, ϵ-deletions were more commonly used.

12.3.3 Question 3: Predicting shortcuts

The random forest model of path type (brute-force, direct, shortcut) had a cross-validation accuracy of 77.2%. Permutation importances of predictors varied widely (Fig. F.8). The most important predictors were ΔRichness and ΔAbundance (desired state value minus initial state value) (Fig. 12.5, Fig. F.9). Shortcut paths were most probable when ΔRichness was positive and ΔAbundance was negative, i.e. cases involving introducing species and displacing dominant species. Shortcut paths were also more probable when Jaccard similarity was small, ϵ was large, and m was large; costs had negligible effects (Fig. 12.5).

12.4 Discussion

We showed that navigation between states is an equivalent problem to searching for lowest-cost sequences of actions that comprise direct and shortcut paths. Shortcuts can be obtained by using small sequential abundance perturbations (e.g. low-density introductions) and environment perturbations to nudge communities between states. Shortcuts were most probable when large richness-increasing, abundance-decreasing, similarity-decreasing state shifts were desired, when perturbation size (ϵ) was large, and when environmental change was possible. Thus, our work suggests that brute-force approaches to navigation like antibiotics or clearcutting may have realistic and less impactful alternatives.

12.4.1 Application cases

The approach could be used for navigation problems where there are a finite number of fixed points to be considered, and where the time to reach a fixed point is substantially smaller than the timescale of the overall problem. Realistic application cases may include communities with fast population dynamics, e.g., microbial communities or bioreactors/chemostats, or annual plants. Optimistic application cases could include resolving human health problems that are linked to the microbiome, [167, 168], e.g. *C. difficile* removal, or improvement of crop/soil health via associated microbial communities [134]. Additionally, applications could be possible in annual plant restoration projects [46, 141].

The state diagram approach could also be useful for assembling synthetic communities, e.g. in microbial bioreactor applications [18, 39]. This problem maps onto the navigation problem, because the desired state is a certain feasible and stable community and the initial state is an empty community. Action sequences could be identified to achieve these goals when brute-force assembly of the desired state is not possible or efficient.

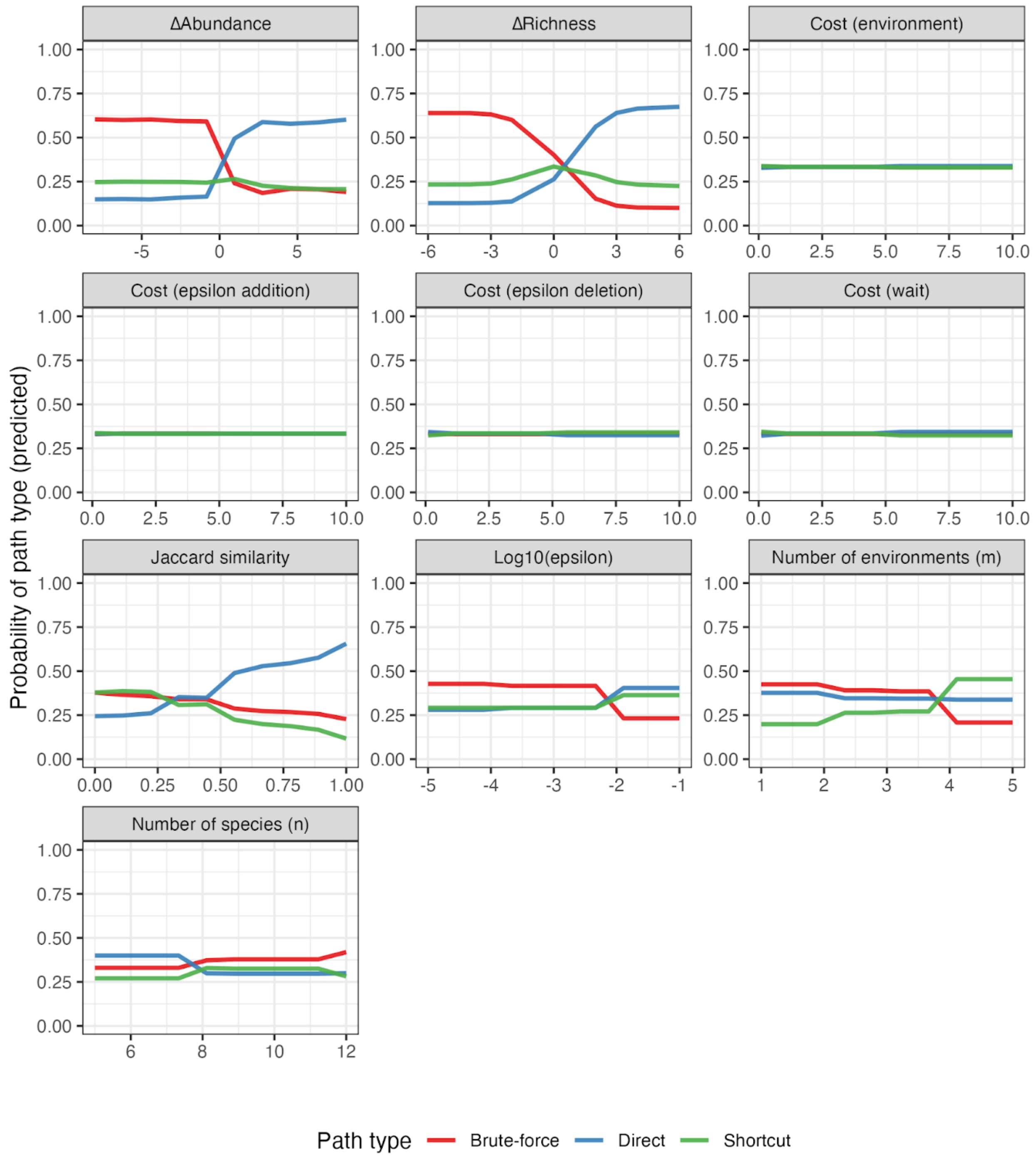


Figure 12.5: **Random Forest Partial Dependence (Full).** Partial dependence plots indicating the effect of each individual predictor on the probability of navigation yielding a brute-force solution, direct path, or shortcut path.

12.4.2 Extensions to the navigation approach

We implicitly assumed that the species pool richness was relatively low, which allowed us to use the A* algorithm. This algorithm does not work well when n or m are large, because the state diagram becomes too large to explore. However, the pathfinding problem does not actually require full exploration of the state diagram if quasi-optimal solutions are acceptable. Such solutions can be found through local search, which only requires enumeration of a smaller set of states that are transiently reached, plus a slightly larger set of states that are explored and discarded. Approximate algorithms such as Monte Carlo Tree Search (MCTS) [28] can be used for larger problems by focusing computation only on promising state and action sequences. Moreover, MCTS can handle stochastic transitions, as well as uncertainty in observations of states when the problem is formulated as a partially observable Markov decision process [91, 110, 112].

We also assumed that navigation problems involve a single desired state. However in realistic use cases more diffuse targets may exist, e.g. any state with high richness, or where a certain species is present, or where mean trait composition is within a certain range. A* cannot handle this scenario, but MCTS can.

In addition, we assumed that the costs of each action are constant by type. However, ϵ-deleting one species might be more costly than for another, either because the time or effort required is high or may depend on whether a third species is also present. Or the costs of different actions may also not be known in advance. Similarly, we assumed that the magnitude of actions ϵ is constant. Based on our computational experiments, variation in action costs seems unlikely to substantially impact navigation, whereas variation in action magnitude does, with larger ϵ enabling more shortcuts. MCTS could also be used to probabilistically identify navigation solutions when costs are unknown or variable [48].

Last, we assumed that there are no feedbacks among the environment and species, e.g. depletion of limiting resources affecting competition [175]. These effects would shift the identity of and relationships among fixed points. Including them is possible if the environment variables can be treated as state variables, which would require some modification of the current implementation.

Trajectories do not necessarily reach fixed points in other models, and could instead reach other attractors like limit cycles. Additionally, multiple fixed points for each combination of species could exist, meaning that the value of ϵ would take a larger role in determining which basin of attraction was reached. Both scenarios would increase the cardinality of the state space and action space. However, if ‘states’ and ‘actions’ can still be defined, then a discretized state diagram can still be constructed.

Safe navigation is also a priority for applications. Navigation should avoid certain states if they are unethical to create, or if their creation would have negative ecological consequences [11, 132]. Notably many paths discovered by our approach transiently put the community into higher-richness states that include novel species (e.g. orange-colored states in Fig. 12.2). This strategy may have substantial risk if those novel species escape due to mechanisms not included in the dynamical model. Adding safety constraints could strongly influence reachability of desired states [16] and require algorithms beyond our current implementation.

12.4.3 Implications for community assembly

State diagrams provide potential linkages to community assembly, under the assumption that the invasion of new species is infrequent relative to the dynamics. The invasion graph [76] is the subgraph of the state diagram comprising only actions that include a single addition and then a wait action (all richness-increasing 'ladders'; Fig. F.2). These actions, and the states they connect, enumerate the most complex communities that can be reached via sequential single invasions. Notably, most states cannot be reached this way; they instead require more complex actions present in the full state diagram (e.g. direct paths involving multiple simultaneous additions and then a wait; or shortcut paths involving multiple wait actions). Conversely, one can also conceptualize an 'un-invasion' graph, which is the subgraph of the state diagram comprising the wait actions linking transient states to fixed point states with no environment change (all richness-decreasing 'snakes'; Fig. F.3). These actions, and the states they connect, enumerate the possible paths by which transiently-reached communities can decay into stable communities. The un-invasion and invasion graphs have non-trivial structures that may be useful for describing community assembly/dis-assembly pathways. We have not yet investigated the general properties of these subgraphs, but see [5, 70, 76].

Second, state diagrams may also help understand priority effects, i.e. order-dependent community assembly [59]. This is because repeatedly taking single actions and then waiting potentially has outcomes that depend on the order of operations; more strongly, taking multiple actions at the same time and then waiting may have different consequences than taking each action in sequence. We did not systematically explore such order dependence, but see [156].

Third, some states may be harder to reach than others, both in community assembly and in navigation. States that have no incident paths are impossible to reach except by brute-force assembly, while those that have very few outgoing paths (especially involving shortcuts) are potentially less likely to reach by chance. These states are related to the 'holes' described by [9]. Initial states with very few outgoing paths are potentially less likely to change state by chance. States that are only reached by 'ladders' may be more easily built up from lower richness states, while states that are only reached by 'snakes' may be more easily broken down from higher richness states. In support of this idea, species combinations most likely to persist under environmental perturbation are more frequent [122]. There may also be 'game changing' species [49] that are disproportionately important for shaping the properties of the state diagram, both in terms of the prevalence and identity of feasible and stable states, as well as the prevalence and identity of shortcuts.

12.5 Conclusion

State diagrams may be useful for solving applied navigation problems and understanding community assembly. Our current work is limited by its focus on numerical simulation for a single dynamical model. Adapting coexistence theory [62, 106, 149] to make general predictions about state diagram topology may be fruitful. Additionally, experimental validation of navigation predictions for community ecology has been absent except in a few microbial [18, 39] and insect [50] cases. Validation is a priority next step for making progress towards real-world applications.

Part V

Conclusion

Chapter 13

Conclusion and Future Works

As both a theoretician and a practitioner of sequential decision making under uncertainty, I aim to design algorithms and procedures that have theoretical guarantees and can be useful in many different contexts. As I dove deeper into some application domains like robotics, ecology, and other natural sciences, it was evident that both the ability to formulate and translate these problems into decision making problems and also the adaptability for the methods to be able to address the realistic problems were important. Consequently, in order for these sequential decision making algorithms to be truly impactful in their application domains, I firmly believe it is crucial that these algorithms not only enjoy more rigorous theoretical guarantees under relaxed assumptions but also take into account the context in which the algorithms are used, especially the various realistic constraints that are in odds with the previous assumptions. Thus, in this section, I briefly outline and contextualize my conclusions, and provide some future directions and studies that could be helpful in this endeavor.

13.1 Conclusion

In Part II, I have developed a number of continuous space POMDP algorithms with provable guarantees, and provided a general convergence result that formally justifies the particle belief approximation technique used in many POMDP algorithms. Previously, there were many efficient continuous observation POMDP algorithms that utilized particle likelihood weighting scheme, but few algorithms were equipped with theoretical guarantees. Furthermore, not many algorithms could additionally handle the continuous/hybrid action space assumption, resulting in suboptimal plans and often needing a discretization scheme.

The introduction of POWSS and consequently Sparse Sampling-ω algorithm enabled rigorous mathematical analyses of a family of online continuous observation POMDP algorithms that utilize the particle likelihood weighting scheme, which notably does not rely on of any discretization schemes nor directly depends on observation space size. The development of POWSS and Sparse Sampling-ω provided the foundations for the particle belief MDP approximate optimality result, where the gap between exactly solving a POMDP problem and its corresponding particle belief MDP approximation was shown to be bounded. This novel result is first-of-its-kind to provide any general guarantee for using particle likelihood weighted belief representations, and will serve not only as the theoretical justification for pre-existing continuous observation POMDP solvers but also as the theoretical stepping stone for more rigorous analyses and tighter inequality bounds. VPW technique adds on to the theoretical rigor by additionally tackling the continuous action assumption. This algorithm is the first known continuous space POMDP algorithm with guarantees, and can have practical extensions that serve as useful baselines of comparison against other continuous space POMDP algorithms due to the ease of implementation.

In Part III, I have devised two approaches to integrate planning algorithms and machine learning model components. Previously for vision POMDPs, many algorithms approached the problem through either end-to-end learning of a global policy or combining model-based planning with deep learning models through differentiable particle filters. However, these algorithms often lacked interpretability, and required long online interactive training. VTS architecture enabled further integration of POMDP model components by learning deep generative models. By explicitly reasoning with likely future observations encountered from the training data, VTS strikes a balance between offline training and online planning times while remaining successful and robust.

For robotic manipulation, transporter networks and their variants provided a powerful method of rearranging objects through computer vision and inverse kinematics control. By introducing sequence conditioning and weighted multi-task learning, Sequence-Conditioned Transporter Networks can further tackle long horizon multiple tasks at once even with minimal training data.

In Part IV, I have proposed a method to translate species coexistence navigation problem into a path planning problem. The species coexistence dynamics information is distilled into a state diagram, which can be used to determine if shortcut paths are possible and desirable. Previous works have focused on explicitly solving the control problem, which often quickly becomes prohibitive with more species present. The state diagram approach is a novel perspective on species coexistence navigation by only looking at small perturbations to the state and then waiting to reach the desired stable and feasible fixed point. The integration of fundamental ideas from species coexistence dynamics modeling in ecology and path planning algorithms in computer science not only provides an efficient and robust approach to control but also motivates further integration and collaboration of the two fields.

13.2 Future Directions

13.2.1 Theoretical Foundations of Sequential Decision Making with More Realistic Assumptions

Planning with model error. One interesting question is: how much of these guarantees can carry over when the models that we are assuming are somewhat wrong? In many realistic scenarios, approximate models are used very often for various reasons: there could be simplifying assumptions to deal with the compute and modeling complexity issues, lack of understanding of the phenomena to capture the entire behavior of the system, lack of data to learn and represent the model components, and inherent uncertainty of the system that causes variations. Here, I aim to incorporate concepts from robust statistics to attempt to prove near-optimality guarantees (or the lack thereof) when the models that we have access to are not exactly accurate. This area could have a huge impact on model-based planning and model-based reinforcement learning, as model errors are inevitable in real scenarios.

Constrained or risk-sensitive planning. Another avenue of research is to incorporate constrained optimization or risk-sensitive optimization to sequential decision making under uncertainty. There has recently been extensive amounts of studies on constrained and risk-sensitive planning in the MDP context with promising empirical applications. Being able to take hard constraints into consideration as well as strike a balance between cautious and efficient planning with risk-sensitive measures will be extremely valuable in safety-critical scenarios. Towards this vision, I aim to develop and prove POMDP methods that are able to reason about constraints and risk-sensitive objective measures.

Further justifications of existing algorithms. On the other hand, many of the existing modern algorithms for POMDPs still do not have complete guarantees. Especially for tree search methods, sequentially building the tree essentially is a non-stationary process which is extremely hard to reason about theoretically. There are still a lot of assumptions that need to be justified, for instance using variable numbers of particles at different depths of the planning tree or selectively exploring state trajectories of different lengths.

13.2.2 Various Approaches to Compositional Learning-based Planning

Learning-based planning with guarantees. The visual tree search approach combined the conventional POMDP closed-loop planning method with learning-based model components. Despite the closed-loop planning components having theoretical guarantees, there still isn't a rigorous guarantee for the entire planning system, especially due to errors from neural network model-based planning. Combined with the theoretical model error quantification, I aim to design a more theoretically well-justified system that has closed-loop guarantees even with learning-based components. Furthermore, there are alternative approaches to putting guarantees on learning-based planning under different formulations and assumptions, which I have recently been exploring in projects such as [135].

General quantification of uncertainty in compositional systems. Separately, there are many techniques in deep learning that aim to capture some notion of uncertainty in the models. I plan to integrate these techniques in order to utilize the estimated uncertainties in different model components of the compositional planning stack, and use the uncertainty information to calibrate the system to plan more safely and efficiently in previously unseen settings.

13.2.3 Further Applications and Extensions to Ecology and Natural Sciences

Tackling further assumptions in species coexistence navigation. The state diagram approach is the first step of introducing the concept of sequential decision making into the species coexistence navigation problem. However, the assumptions made in the analyses are rather rudimentary, since we assume that the dynamics model is known with a theoretically well-understood GLV structure, and there are no uncertainties present. In many realistic ecology application cases, such as forrest management and human gut microbiome health improvement, the dynamics model is much more complex than the GLV structure, and there are a lot of uncertainties in the system evolution. Thus, the natural next step would be to incorporate some of these realistic assumptions on model complexity and model uncertainty, and leverage existing POMDP planners to solve these problems.

Combining domain knowledge and machine learning for ecology. On the other hand, there can be a strong synergy in applying the compositional learning framework to ecological problems. For instance, ecologists are often interested in learning the dynamics model of the system from experimental data, as well as being able to plan from the data. I believe that the compositional approach of leveraging domain knowledge combined with learning the unexplained phenomena could be extremely impactful in learning approximate dynamics model from limited data.

Further collaborations between ecologists and computer scientists. Through my personal experience collaborating with ecologists and natural scientists, I have learned the importance of cross-domain scientific communication. I would not only like to serve as an interdisciplinary scientist between these domains but also help facilitate discussions between these scientists in order to discover other problems and areas that could be interesting and important to solve and can be feasibly solved by applying existing methods.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems http://download.tensorflow.org/paper/whitepaper2015.pdf*. 2015.

[2] Ali-Akbar Agha-Mohammadi, Suman Chakravorty, and Nancy M. Amato. "FIRM: Feedback controller-based information-state roadmap - a framework for motion planning under uncertainty". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2011.

[3] Bo Ai et al. "Deep Visual Navigation under Partial Observability". In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 9439–9446.

[4] Ferran Alet, Tomas Lozano-Perez, and Leslie P. Kaelbling. "Modular meta-learning". In: *Conference on Robot Learning (CoRL)*. Vol. 87. PMLR, 2018, pp. 856–868.

[5] Pablo Almaraz, José A Langa, and Piotr Kalita. "Structural stability of invasion graphs for generalized Lotka–Volterra systems". In: *arXiv preprint 2209.09802* (2022).

[6] Jacob Andreas et al. "Neural Module Networks". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[7] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2017).

[8] Marco Tulio Angulo, Claude H. Moog, and Yang-Yu Liu. "A theoretical framework for controlling complex microbial communities". In: *Nature Communications* 10.1 (2019). ISBN: 2041-1723 Publisher: Nature Publishing Group, pp. 1–12.

[9] Marco Tulio Angulo et al. "Coexistence holes characterize the assembly and disassembly of multispecies systems". In: *Nature Ecology & Evolution* (2021). ISBN: 2397-334X Publisher: Nature Publishing Group, pp. 1–11.

[10] Iro Armeni et al. "Joint 2d-3d-semantic data for indoor scene understanding". In: *arXiv preprint 1702.01105* (2017).

[11] Anil Aswani et al. "Provably safe and robust learning-based model predictive control". In: *Automatica* 49.5 (2013). ISBN: 0005-1098 Publisher: Elsevier, pp. 1216–1226.

[12] David Auger, Adrien Couëtoux, and Olivier Teytaud. "Continuous Upper Confidence Trees with Polynomial Exploration – Consistency". In: *Machine Learning and Knowledge Discovery in Databases.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 194–209.

[13] Turgay Ayer, Oguzhan Alagoz, and Natasha K Stout. "A POMDP approach to personalize mammography screening decisions". In: *Operations Research* 60.5 (2012), pp. 1019–1034.

[14] Haoyu Bai, David Hsu, and Wee Sun Lee. "Integrated perception and planning in the continuous space: A POMDP approach". In: *International Journal of Robotics Research* 33.9 (2014), pp. 1288–1302.

[15] Haoyu Bai et al. "Intention-Aware Online POMDP Planning for Autonomous Driving in a Crowd". In: *IEEE International Conference on Robotics and Automation (ICRA).* 2015, pp. 454–460.

[16] Somil Bansal and Claire Tomlin. "DeepReach: A Deep Learning Approach to High-Dimensional Reachability". In: *arXiv preprint 2011.02082* (2020).

[17] György Barabás, Matthew J. Michalska-Smith, and Stefano Allesina. "The effect of intra-and interspecific competition on coexistence in multispecies communities". In: *The American Naturalist* 188.1 (2016), E1–E12.

[18] Mayank Baranwal et al. "Recurrent neural networks enable design of multifunctional synthetic human gut microbiome dynamics". In: *eLife* 11 (2022). ISBN: 2050-084X Publisher: eLife Sciences Publications Limited, e73870.

[19] Lars Berscheid, Pascal Meißner, and Torsten Kröger. "Self-supervised Learning for Precise Pick-and-place without Object Model". In: *IEEE Robotics and Automation Letters (RA-L)* (2020).

[20] D. Bertsekas. *Dynamic Programming and Optimal Control.* Athena, 2005.

[21] Ronald Bjarnason, Alan Fern, and Prasad Tadepalli. "Lower bounding Klondike solitaire with Monte-Carlo planning". In: *International Conference on Automated Planning and Scheduling (ICAPS).* 2009.

[22] Benjamin Blonder. *bblonder/shortcut_navigation: v1.0.0. https://doi.org/10.5281/zenodo.7527425.* Jan. 2023.

[23] Benjamin W. Blonder et al. "Navigation between initial and desired community states using shortcuts". In: *Ecology Letters* 26.4 (2023), pp. 516–528.

[24] Carl Boettiger, Marc Mangel, and Stephan Munch. "Avoiding tipping points in fisheries management through Gaussian process dynamic programming". In: *Proceedings of the Royal Society B: Biological Sciences* 282.1801 (2015). ISBN: 0962-8452 Publisher: The Royal Society, p. 20141631.

[25] J. Bresina et al. “Planning under Continuous Time and Resource Uncertainty: a challenge for AI”. In: *Conference on Uncertainty in Artificial Intelligence (UAI).* 2002.

[26] Antoine Brias and Stephan B. Munch. “Ecosystem based multi-species management using Empirical Dynamic Programming”. In: *Ecological Modelling* 441 (2021). ISBN: 0304-3800 Publisher: Elsevier, p. 109423.

[27] Greg Brockman et al. “OpenAI Gym”. In: *arXiv preprint 1606.01540* (2016).

[28] Cameron B. Browne et al. “A survey of Monte Carlo tree search methods”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43.

[29] Adam Bry and Nicholas Roy. “Rapidly-exploring random belief trees for motion planning under uncertainty”. In: *IEEE International Conference on Robotics and Automation (ICRA).* 2011, pp. 723–730.

[30] C. G. Buffie et al. “Profound alterations of intestinal microbiota following a single dose of clindamycin results in sustained susceptibility to Clostridium difficile-induced colitis”. In: *Infect Immun* 80 (2012).

[31] Francesco Carrara et al. “Dendritic connectivity controls biodiversity patterns in experimental metacommunities”. In: *Proceedings of the National Academy of Sciences* 109.15 (2012). ISBN: 0027-8424 Publisher: National Acad Sciences, pp. 5761–5766.

[32] Francesco Carrara et al. “Inferring species interactions in ecological communities: a comparison of methods at different levels of complexity”. In: *Methods in Ecology and Evolution* 6.8 (2015). ISBN: 2041-210X Publisher: Wiley Online Library, pp. 895–906.

[33] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. “Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes”. In: *Conference on Uncertainty in Artificial Intelligence (UAI).* 1997, pp. 54–61.

[34] Anthony R Cassandra. “A survey of POMDP applications”. In: *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes.* 1998.

[35] Margaret P. Chapman et al. “Risk-Sensitive Safety Analysis Using Conditional Value-at-Risk”. In: *IEEE Transactions on Automatic Control* 67.12 (2022), pp. 6521–6536.

[36] Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. “Shortest paths algorithms: Theory and experimental evaluation”. In: *Mathematical Programming* 73.2 (1996). ISBN: 1436-4646 Publisher: Springer, pp. 129–174.

[37] Gregory Chow. *Analysis and Control of Dynamic Economic Systems.* New York: John Wiley & Sons, 1975.

[38] Yinlam Chow et al. “Risk-Constrained Reinforcement Learning with Percentile Risk Criteria”. In: *arXiv preprint 1512.01629* (2017).

[39] Ryan L. Clark et al. “Design of synthetic human gut microbiome assembly and butyrate production”. In: *Nature Communications* 12.1 (2021). ISBN: 2041-1723 Publisher: Nature Publishing Group, pp. 1–16.

[40] Taco Cohen and Max Welling. “Group equivariant convolutional networks”. In: *International Conference on Machine Learning (ICML)*. 2016.

[41] Adrien Corenflos et al. “Differentiable Particle Filtering via Entropy-Regularized Optimal Transport”. In: *International Conference on Machine Learning (ICML)*. Vol. 139. PMLR, 2021, pp. 2100–2111.

[42] E. K. Costello et al. “The application of ecological theory toward an understanding of the human microbiome”. In: *Science* 336 (2012).

[43] A. Couëtoux et al. “Continuous Upper Confidence Trees”. In: *Learning and Intelligent Optimization*. Rome, Italy, 2011.

[44] Erwin Coumans and Yunfei Bai. “Pybullet, a python module for physics simulation for games, robotics and machine learning”. In: *GitHub Repository* (2016).

[45] D. Crisan and A. Doucet. “A survey of convergence results on particle filtering methods for practitioners”. In: *IEEE Transactions on Signal Processing* 50.3 (2002), pp. 736–746.

[46] Carla D’Antonio and Laura A Meyerson. “Exotic plant species as problems and solutions in ecological restoration: a synthesis”. In: *Restoration Ecology* 10.4 (2002), pp. 703–713.

[47] Sudeep Dasari and Abhinav Gupta. “Transformers for One-Shot Visual Imitation”. In: *Conference on Robot Learning (CoRL)*. 2020.

[48] Sampada Deglurkar et al. “Visual Learning-based Planning for Continuous High-Dimensional POMDPs”. In: *arXiv preprint 2112.09456* (2021).

[49] Jie Deng, Marco Tulio Angulo, and Serguei Saavedra. “Generalizing game-changing species across microbial communities”. In: *ISME Communications* 1.1 (2021). ISBN: 2730-6151 Publisher: Nature Publishing Group, pp. 1–8.

[50] Robert A. Desharnais et al. “Chaos and population control of insect outbreaks”. In: *Ecology Letters* 4.3 (2001), pp. 229–235.

[51] Coline Devin et al. “Learning modular neural network policies for multi-task and multi-robot transfer”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2169–2176.

[52] Coline Devin et al. “Self-Supervised Goal-Conditioned Pick and Place”. In: *arXiv preprint 2008.11466* (2020).

[53] Maxim Egorov et al. “POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty”. In: *Journal of Machine Learning Research* 18.26 (2017), pp. 1–5.

[54] Joseph A. Falco, Kenny Kimble, and Adam Norton. *IROS 2020 Robotic grasping and manipulation COMPETITION: Manufacturing Track.* Aug. 2020.

[55] Chelsea Finn et al. “One-Shot Visual Imitation Learning via Meta-Learning”. In: *Conference on Robot Learning (CoRL).* 2017.

[56] Peter R. Florence, Lucas Manuelli, and Russ Tedrake. “Dense Object Nets: Learning Dense Visual Object Descriptors By and For Robotic Manipulation”. In: *Conference on Robot Learning (CoRL).* 2018.

[57] Lester R. Ford Jr. *Network flow theory.* Tech. rep. Santa Monica: RAND Corporation, 1956.

[58] Eric W Frew et al. “Field observation of tornadic supercells by multiple autonomous fixed-wing unmanned aircraft”. In: *Journal of Field Robotics* 37.6 (2020), pp. 1077–1093.

[59] Tadashi Fukami. “Historical Contingency in Community Assembly: Integrating Niches, Species Pools, and Priority Effects”. In: *Annual Review of Ecology, Evolution, and Systematics* 46.1 (Dec. 2015), pp. 1–23.

[60] Beatriz García-Jiménez, Tomás de la Rosa, and Mark D Wilkinson. “MDPbiome: microbiome engineering through prescriptive perturbations”. In: *Bioinformatics* 34.17 (Sept. 2018), pp. i838–i847.

[61] Neha P. Garg, David Hsu, and Wee Sun Lee. “DESPOT-alpha: Online POMDP Planning With Large State And Observation Spaces”. In: *Robotics: Science and Systems.* 2019.

[62] Theo Gibbs, Simon A. Levin, and Jonathan M. Levine. “Coexistence in diverse communities with higher-order interactions”. In: *bioRxiv* (2022). Publisher: Cold Spring Harbor Laboratory.

[63] Daniel Golovin et al., eds. *Google Vizier: A Service for Black-Box Optimization.* 2017.

[64] Marcus Gualtieri, Andreas ten Pas, and Robert Platt. “Pick and place without geometric object models”. In: *IEEE International Conference on Robotics and Automation (ICRA).* 2018.

[65] María Elena López Guillén et al. “A Navigation System for Assistant Robots Using Visually Augmented POMDPs”. In: *Auton. Robots* 19.1 (2005), pp. 67–87.

[66] Izzeddin Gur et al. “Adversarial Environment Generation for Learning to Navigate the Web”. In: *arXiv preprint 2103.01991* (2021).

[67] David Ha and Jürgen Schmidhuber. “Recurrent World Models Facilitate Policy Evolution”. In: *Advances in Neural Information Processing Systems (NeurIPS).* Curran Associates, Inc., 2018, pp. 2451–2463.

[68] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *International Conference on Machine Learning (ICML).* Vol. 80. PMLR, 2018, pp. 1861–1870.

[69] Danijar Hafner et al. "Learning Latent Dynamics for Planning from Pixels". In: *International Conference on Machine Learning (ICML).* Vol. 97. PMLR, 2019, pp. 2555–2565.

[70] Luh Hang-Kwang and Stuart L. Pimm. "The Assembly of Ecological Communities: A Minimalist Approach". In: *Journal of Animal Ecology* 62.4 (1993). Publisher: Wiley, British Ecological Society, pp. 749–765.

[71] Harikrishna. "Kridakausalya". In: *Brihajjyotisarnava.* Bombay: Jagadishvara Printing Press, 1871.

[72] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968). ISBN: 0536-1567 Publisher: IEEE, pp. 100–107.

[73] Kaiming He et al. "Deep residual learning for image recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016.

[74] Marcus Hoerger and Hanna Kurniawati. "An On-Line POMDP Solver for Continuous Observation Spaces". In: *IEEE International Conference on Robotics and Automation (ICRA).* 2021.

[75] Jesse Hoey and Pascal Poupart. "Solving POMDPs with continuous or large discrete observation spaces". In: *International Joint Conference on Artificial Intelligence (IJCAI).* 2005, pp. 1332–1338.

[76] Josef Hofbauer and Sebastian J. Schreiber. "Permanence via invasion graphs: Incorporating community assembly into Modern Coexistence Theory". In: *arXiv preprint 2204.03773* (2022).

[77] Jessica E. Holland, Mykel J. Kochenderfer, and Wesley A. Olson. "Optimizing the Next Generation Collision Avoidance System for Safe, Suitable, and Acceptable Operational Performance". In: *Air Traffic Control Quarterly* 21.3 (2013), pp. 275–297.

[78] Drew Arad Hudson and Christopher D. Manning. "Compositional Attention Networks for Machine Reasoning". In: *International Conference on Learning Representations (ICLR).* 2018.

[79] Andrew Hundt et al. ""Good Robot!": Efficient Reinforcement Learning for Multi-Step Visual Tasks with Sim to Real Transfer". In: *IEEE Robotics and Automation Letters (RA-L)* (2020).

[80] Maximilian Igl et al. "Deep Variational Reinforcement Learning for POMDPs". In: *International Conference on Machine Learning (ICML).* Vol. 80. PMLR, 2018, pp. 2117–2126.

[81] Charles R. Johnson and Herbert A. Robinson. "Eigenvalue inequalities for principal submatrices". In: *Linear Algebra and its Applications* 37 (1981). ISBN: 0024-3795 Publisher: Elsevier, pp. 11–22.

[82] Eric W. Jones, Parker Shankin-Clarke, and Jean M. Carlson. "Navigation and control of outcomes in a generalized Lotka-Volterra model of the microbiome". In: *arXiv preprint 2003.12954* (2020).

[83] Rico Jonschkowski, Divyam Rastogi, and Oliver Brock. "Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors". In: *IEEE Robotics: Science and Systems (RSS)*. Pittsburgh, Pennsylvania, June 2018.

[84] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1 (1998), pp. 99–134.

[85] Dmitry Kalashnikov et al. "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *Conference on Robot Learning (CoRL)*. Vol. 87. PMLR, 29–31 Oct 2018, pp. 651–673.

[86] Dmitry Kalashnkov et al. "MT-OPT: Continuous Multi-Task Robotic Reinforcement Learning at Scale". In: *arXiv preprint 2104.08212* (2021).

[87] Peter Karkus, Shaojun Cai, and David Hsu. "Differentiable SLAM-Net: Learning Particle SLAM for Visual Navigation". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 2815–2825.

[88] Peter Karkus, David Hsu, and Wee Sun Lee. "Integrating Algorithmic Planning and Deep Learning for Partially Observable Navigation". In: *arXiv preprint 1807.06696* (2018).

[89] Peter Karkus, David Hsu, and Wee Sun Lee. "QMDP-Net: Deep Learning for Planning under Partial Observability". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. Curran Associates, Inc., 2017.

[90] Peter Karkus et al. "Differentiable Mapping Networks: Learning Structured Map Representations for Sparse Visual Localization". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 4753–4759.

[91] Sammie Katt, Frans A. Oliehoek, and Christopher Amato. "Learning in POMDPs with Monte Carlo tree search". In: *International Conference on Machine Learning (ICML)*. Vol. 70. 2017, pp. 1819–1827.

[92] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. "A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes". In: *Machine Learning* 49.2 (2002), pp. 193–208.

[93] Mohi Khansari et al. "Action Image Representation: Learning Scalable Deep Grasping Policies with Zero Real World Data". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[94] Beomjoon Kim et al. "Monte Carlo Tree Search in Continuous Spaces Using Voronoi Optimistic Optimization with Regret Bounds". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2020, pp. 9916–9924.

[95] Louis Kirsch, Julius Kunze, and David Barber. "Modular Networks: Learning to Decompose Neural Computation". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 31. Curran Associates, Inc., 2018.

[96] Mykel J. Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. Massachusetts: MIT Press, 2015.

[97] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. MIT Press, 2019.

[98] Levente Kocsis and Csaba Szepesvári. "Bandit based Monte-Carlo planning". In: *European Conference on Machine Learning (ECML)*. Springer. 2006, pp. 282–293.

[99] Risi Kondor and Shubhendu Trivedi. "On the generalization of equivariance and convolution in neural networks to the action of compact groups". In: *International Conference on Machine Learning (ICML)*. 2018.

[100] Paul R. Krausman, James W. Cain III, and James W. Cain. *Wildlife management and conservation: contemporary principles and practices*. Baltimore: Johns Hopkins University Press, 2013.

[101] Hanna Kurniawati and Vinay Yadav. "An online POMDP solver for uncertainty planning in dynamic environment". In: *Robotics Research*. Springer, 2016, pp. 611–629.

[102] Marcus Lapeyrolerie et al. "Deep reinforcement learning for conservation decisions". In: *Methods in Ecology and Evolution* 13.11 (Nov. 2022). Publisher: John Wiley & Sons, Ltd, pp. 2649–2662.

[103] Michael Laskey et al. "SHIV: Reducing supervisor burden in DAgger using support vectors for efficient learning from demonstrations in high dimensional state spaces". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 462–469.

[104] Jongmin Lee et al. "Monte-Carlo Tree Search in Continuous Action Spaces with Value Gradients". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2020, pp. 4561–4568.

[105] Wee Sun Lee, Nan Rong, and David Hsu. "What makes some POMDP problems easy to approximate?" In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2008, pp. 689–696.

[106] Jonathan M. Levine et al. "Beyond pairwise mechanisms of species coexistence in complex communities". In: *Nature* 546.7656 (2017), pp. 56–64.

[107] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *Journal of Machine Learning Research (JMLR)* (2016).

[108] Yunfei Li et al. "Solving Compositional Reinforcement Learning Problems via Task Reduction". In: *International Conference on Learning Representations (ICLR)*. 2021.

[109] Michael H. Lim, Claire J. Tomlin, and Zachary N. Sunberg. “Sparse Tree Search Optimality Guarantees in POMDPs with Continuous Observation Spaces”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2020.

[110] Michael H. Lim, Claire J. Tomlin, and Zachary N. Sunberg. “Voronoi Progressive Widening: Efficient Online Solvers for Continuous State, Action, and Observation POMDPs”. In: *IEEE Conference on Decision and Control (CDC)*. 2021, pp. 4493–4500.

[111] Michael H. Lim et al. “Multi-Task Learning with Sequence-Conditioned Transporter Networks”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2022, pp. 2489–2496.

[112] Michael H. Lim et al. “Optimality Guarantees for Particle Belief Approximation of POMDPs”. In: *arXiv preprint 2210.05015* (2022).

[113] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. “Learning Policies for Partially Observable Environments: Scaling Up”. In: *International Conference on Machine Learning (ICML)*. 1995.

[114] Jianlan Luo et al. “Robust Multi-Modal Policies for Industrial Assembly via Reinforcement Learning and Demonstrations: A Large-Scale Study”. In: *IEEE Robotics: Science and Systems (RSS)*. 2021.

[115] Jeffrey Mahler et al. “Learning ambidextrous robot grasping policies”. In: *Science Robotics* 4.26 (2019).

[116] Shie Mannor, Reuven Rubinstein, and Yohai Gat. “The Cross Entropy Method for Fast Policy Search”. In: *International Conference on Machine Learning (ICML)*. Washington, DC, USA: AAAI Press, 2003, pp. 512–519.

[117] Chris Mansley, Ari Weinstein, and Michael L. Littman. “Sample-Based Planning for Continuous Action Markov Decision Processes”. In: *International Conference on Automated Planning and Scheduling (ICAPS)*. Freiburg, Germany: AAAI Press, 2011, pp. 335–338.

[118] Weichao Mao et al. “POLY-HOOT: Monte-Carlo Planning in Continuous Space MDPs with Non-Asymptotic Analysis”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. Curran Associates, Inc., 2020.

[119] Margaret M. Mayfield and Daniel B. Stouffer. “Higher-order interactions capture unexplained complexity in diverse communities”. In: *Nature Ecology & Evolution* 1.3 (2017). ISBN: 2397-334X Publisher: Nature Publishing Group, pp. 1–7.

[120] Daniel S. Maynard, Zachary R. Miller, and Stefano Allesina. “Predicting coexistence in experimental ecological communities”. In: *Nature Ecology & Evolution* 4.1 (2020). ISBN: 2397-334X Publisher: Nature Publishing Group, pp. 91–100.

[121] David A. McAllester and Satinder Singh. “Approximate Planning for Factored POMDPs Using Belief State Simplification”. In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. Stockholm, Sweden, 1999, pp. 409–416.

[122] Lucas P. Medeiros et al. “Observed ecological communities are formed by species combinations that are among the most likely to persist under changing environments”. In: *The American Naturalist* 197.1 (2021). ISBN: 0003-0147 Publisher: The University of Chicago Press Chicago, IL, E17–E29.

[123] Milad Memarzadeh and Carl Boettiger. “Adaptive Management of Ecological Systems under Partial Observability”. In: *Biological Conservation* 224 (2018), pp. 9–15.

[124] John Mern et al. “Bayesian Optimized Monte Carlo Planning”. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2021.

[125] John Mern et al. “Improved POMDP Tree Search Planning with Prioritized Action Branching”. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2021.

[126] Alberto Maria Metelli et al. “Policy Optimization via Importance Sampling”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018, pp. 5442–5454.

[127] Elliot Meyerson and Risto Miikkulainen. “Beyond Shared Hierarchies: Deep Multitask Learning through Soft Layer Ordering”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[128] Mehdi Mirza and Simon Osindero. “Conditional Generative Adversarial Nets”. In: *arXiv preprint 1411.1784* (2014).

[129] Sarthak Mittal et al. “Learning to Combine Top-Down and Bottom-Up Signals in Recurrent Neural Networks with Attention over Modules”. In: *International Conference on Machine Learning (ICML)*. Vol. 119. PMLR, 2020, pp. 6972–6986.

[130] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *International Conference on Machine Learning (ICML)*. Vol. 48. PMLR, 20–22 Jun 2016, pp. 1928–1937.

[131] Volodymyr Mnih et al. “Playing Atari With Deep Reinforcement Learning”. In: *NeurIPS Deep Learning Workshop*. 2013.

[132] Sina Mohseni et al. “Practical machine learning safety: A survey and primer”. In: *arXiv preprint 2106.04823* (2021).

[133] Philippe Morere, Roman Marchant, and Fabio Ramos. “Bayesian Optimisation for solving Continuous State-Action-Observation POMDPs”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. Curran Associates, Inc., 2016.

[134] Ulrich G. Mueller and Joel L. Sachs. “Engineering microbiomes to improve plant and animal health”. In: *Trends in Microbiology* 23.10 (2015). ISBN: 0966-842X Publisher: Elsevier, pp. 606–617.

[135] Anish Muthali et al. “Multi-Agent Reachability Calibration with Conformal Prediction”. In: *arXiv preprint 2304.00432* (2023).

[136] Ofir Nachum et al. “Near-Optimal Representation Learning for Hierarchical Reinforcement Learning”. In: *arXiv preprint 1810.01257* (2019).

[137] Andrew Y. Ng and Stuart Russell. “Algorithms for Inverse Reinforcement Learning”. In: *International Conference on Machine Learning (ICML)*. 2000, pp. 663–670.

[138] Margaret A. Palmer, Joy B. Zedler, and Donald A. Falk. *Foundations of restoration ecology*. Washington, DC: Island Press, 2016.

[139] Christos H. Papadimitriou and John N. Tsitsiklis. “The Complexity of Markov Decision Processes”. In: *Mathematics of Operations Research* 12.3 (1987), pp. 441–450.

[140] Frank Pennekamp et al. “Biodiversity increases and decreases ecosystem stability”. In: *Nature* 563.7729 (2018), pp. 109–112.

[141] Michael P. Perring et al. “Advances in restoration ecology: rising to the challenges of the coming decades”. In: *Ecosphere* 6.8 (2015). Publisher: John Wiley & Sons, Ltd.

[142] Jan Peters and Stefan Schaal. “Natural Actor-Critic”. In: *Neurocomputing* 71.7 (2008). Progress in Modeling, Theory, and Application of Computational Intelligenc, pp. 1180–1190.

[143] Robert Platt Jr. et al. “Belief space planning assuming maximum likelihood observations”. In: *IEEE Robotics: Science and Systems (RSS)*. 2010.

[144] Robert Platt, Colin Kohler, and Marcus Gualtieri. “Deictic image mapping: An abstraction for learning pose invariant manipulation policies”. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2019.

[145] Martin Riedmiller. “Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method”. In: *European Conference on Machine Learning (ECML)*. Berlin, Heidelberg: Springer, 2005, pp. 317–328.

[146] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. “Routing Networks: Adaptive Selection of Non-Linear Functions for Multi-Task Learning”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[147] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning”. In: *arXiv preprint 1011.0686* (2011).

[148] Stéphane Ross et al. “Online planning algorithms for POMDPs”. In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 663–704.

[149] Serguei Saavedra et al. “A structural approach for understanding multispecies coexistence”. In: *Ecological Monographs* 87.3 (2017), pp. 470–486.

[150] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *arXiv preprint 1707.06347* (2017).

[151] John Schulman et al. “Trust Region Policy Optimization”. In: *International Conference on Machine Learning (ICML)*. Vol. 37. Lille, France: PMLR, July 2015, pp. 1889–1897.

[152] K. M. Seiler, H. Kurniawati, and S. P. N. Singh. "An online and approximate solver for POMDPs with continuous action space". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2290–2297.

[153] Konstantin M. Seiler, Hanna Kurniawati, and Surya P. N. Singh. "An online and approximate solver for POMDPs with continuous action space". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 2290–2297.

[154] Daniel Seita et al. "Learning to Rearrange Deformable Cables, Fabrics, and Bags with Goal-Conditioned Transporter Networks". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021.

[155] Pierre Sermanet et al. "Time-Contrastive Networks: Self-Supervised Learning from Video". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[156] Carlos A. Serván et al. "Coexistence of many species in random ecosystems". In: *Nature Ecology & Evolution* 2.8 (2018). ISBN: 2397-334X Publisher: Nature Publishing Group, pp. 1237–1242.

[157] Devavrat Shah, Qiaomin Xie, and Zhi Xu. "Nonasymptotic Analysis of Monte Carlo Tree Search". In: *Operations Research* 70.6 (2022), pp. 3234–3260.

[158] Guy Shani, Joelle Pineau, and Robert Kaplow. "A survey of point-based POMDP solvers". In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* 27.1 (2013), pp. 1–51.

[159] David Silver and Joel Veness. "Monte-Carlo Planning in Large POMDPs". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2010, pp. 2164–2172.

[160] Gautam Singh et al. "Structured World Belief for Reinforcement Learning in POMDP". In: *International Conference on Machine Learning (ICML)*. Vol. 139. PMLR, 18–24 Jul 2021, pp. 9744–9755.

[161] Satinder P. Singh and Richard C. Yee. "An upper bound on the loss from approximate optimal-value functions". In: *Machine Learning* 16.3 (1994), pp. 227–233.

[162] Richard D Smallwood and Edward J Sondik. "The optimal control of partially observable Markov processes over a finite horizon". In: *Operations Research* 21.5 (1973), pp. 1071–1088.

[163] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning Structured Output Representation using Deep Conditional Generative Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 28. Curran Associates, Inc., 2015.

[164] Sungryull Sohn, Junhyuk Oh, and Honglak Lee. "Hierarchical Reinforcement Learning for Zero-shot Generalization with Subtask Dependencies". In: *Neural Information Processing Systems (NeurIPS)*. 2018, pp. 7156–7166.

[165] Sungryull Sohn et al. “Meta Reinforcement Learning with Autonomous Inference of Subtask Dependencies”. In: *International Conference on Learning Representations (ICLR)*. 2020.

[166] Shuran Song et al. “Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations”. In: *IEEE Robotics and Automation Letters (RA-L)* (2020).

[167] Justin L Sonnenburg and Erica D Sonnenburg. “Vulnerability of the industrialized microbiota”. In: *Science* 366.6464 (2019).

[168] Justin L. Sonnenburg. “Microbiome engineering”. In: *Nature* 518.7540 (2015). ISBN: 1476-4687 Publisher: Nature Publishing Group, S10–S10.

[169] Richard R. Stein et al. “Ecological modeling from time-series inference: insight into dynamics and stability of intestinal microbiota”. In: *PLoS Computational Biology* 9.12 (2013). ISBN: 1553-7358 Publisher: Public Library of Science, e1003388.

[170] Zachary Sunberg, Suman Chakravorty, and R Scott Erwin. “Information space receding horizon control”. In: *IEEE Transactions on Cybernetics* 43.6 (2013), pp. 2255–2260.

[171] Zachary Sunberg and Mykel J. Kochenderfer. “Online Algorithms for POMDPs with Continuous State, Action, and Observation Spaces”. In: *International Conference on Automated Planning and Scheduling (ICAPS)*. 2018.

[172] Zachary N. Sunberg, Christopher J. Ho, and Mykel J. Kochenderfer. “The Value of Inferring the Internal State of Traffic Participants for Autonomous Freeway Driving”. In: *American Control Conference (ACC)*. Seattle, WA, USA: IEEE, 2017, pp. 3004–3010.

[173] Richard S. Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine Learning* 3.1 (1988), pp. 9–44.

[174] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[175] David Tilman. *Resource competition and community structure*. Princeton: Princeton University Press, 1982.

[176] E. Todorov and Weiwei Li. “A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems”. In: *American Control Conference (ACC)*. Vol. 1. 2005, pp. 300–306.

[177] Andrew Topsfield. “Snakes and ladders in India: Some further discoveries”. In: *Artibus Asiae* 66.1 (2006). ISBN: 0004-3648 Publisher: JSTOR, pp. 143–179.

[178] Sam Toyer et al. “The MAGICAL Benchmark for Robust Imitation”. In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[179] Jur Van Den Berg, Sachin Patil, and Ron Alterovitz. “Motion planning under uncertainty using iterative local optimization in belief space”. In: *International Journal of Robotics Research* 31.11 (2012), pp. 1263–1278.

[180] Mary N. Van Dyke, Jonathan M. Levine, and Nathan J. B. Kraft. "Small rainfall changes drive substantial changes in plant coexistence". In: *Nature* 611.7936 (2022), pp. 507–511.

[181] Ophelia S. Venturelli et al. "Deciphering microbial interactions in synthetic human gut microbiome communities". In: *Molecular Systems Biology* 14.6 (2018). ISBN: 1744-4292, e8157.

[182] Nelson Vithayathil Varghese and Qusay H. Mahmoud. "A Survey of Multi-Task Deep Reinforcement Learning". In: *Electronics* 9.9 (2020).

[183] Yunbo Wang et al. "DualSMC: Tunneling Differentiable Filtering and Planning under Continuous POMDPs". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. Main track. 2020, pp. 4190–4198.

[184] Christopher John Cornish Hellaby Watkins. "Learning from Delayed Rewards". PhD thesis. Cambridge, UK: King's College, May 1989.

[185] Ari Weinstein and Michael L. Littman. "Bandit-Based Planning and Learning in Continuous-Action Markov Decision Processes". In: *International Conference on Automated Planning and Scheduling (ICAPS)*. Atibaia, São Paulo, Brazil: AAAI Press, 2012, pp. 306–314.

[186] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3 (1992), pp. 229–256.

[187] Chenyang Wu et al. "Adaptive Online Packing-guided Search for POMDPs". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. 2021.

[188] Jimmy Wu et al. "Spatial Action Maps for Mobile Manipulation". In: *IEEE Robotics: Science and Systems (RSS)*. 2020.

[189] Yilin Wu et al. "Learning to manipulate deformable objects without demonstrations". In: *IEEE Robotics: Science and Systems (RSS)*. 2020.

[190] Ruihan Yang et al. "Multi-Task Reinforcement Learning with Soft Modularization". In: *arXiv preprint 2003.13661* (2020).

[191] Nan Ye et al. "DESPOT: Online POMDP Planning with Regularization". In: *Journal of Artificial Intelligence Research* 58 (2017), pp. 231–266.

[192] Steve Young et al. "POMDP-based statistical spoken dialog systems: A review". In: *IEEE* 101.5 (2013), pp. 1160–1179.

[193] Tianhe Yu et al. "One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning". In: *IEEE Robotics: Science and Systems (RSS)*. 2018.

[194] Kevin Zakka et al. "Form2Fit: Learning Shape Priors for Generalizable Assembly from Disassembly". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[195] Andy Zeng. “Learning Visual Affordances for Robotic Manipulation”. PhD thesis. Princeton University, 2019.

[196] Andy Zeng et al. “Learning synergies between pushing and grasping with self-supervised deep reinforcement learning”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018.

[197] Andy Zeng et al. “Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching”. In: *International Conference on Robotics and Automation (ICRA)*. 2018.

[198] Andy Zeng et al. “TossingBot: Learning to Throw Arbitrary Objects with Residual Physics”. In: *IEEE Robotics: Science and Systems (RSS)*. 2019.

[199] Andy Zeng et al. “Transporter Networks: Rearranging the Visual World for Robotic Manipulation”. In: *Conference on Robot Learning (CoRL)*. 2020.

[200] Jiakai Zhang and Kyunghyun Cho. “Query-Efficient Imitation Learning for End-to-End Autonomous Driving”. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2017.

[201] Marvin Zhang et al. “SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning”. In: *International Conference on Machine Learning (ICML)*. Vol. 97. 2019, pp. 7444–7453.

[202] Shengjia Zhao, Jiaming Song, and Stefano Ermon. “Towards Deeper Understanding of Variational Autoencoding Models”. In: *arXiv preprint 1702.08658* (2017).

Part VI

Appendix

Appendix A

Details for Partially Observable Weighted Sparse Sampling

A.1 Proof of Theorem 6.1 - SN d_∞-Concentration Bound

Theorem A.1 (SN d_∞-Concentration Bound)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be two probability measures on the measurable space $(\mathcal{X}, \mathcal{F})$ with $\mathcal{P} \ll \mathcal{Q}$ and $d_\infty(\mathcal{P}||\mathcal{Q}) < +\infty$. Let $x_1, \cdots, x_N$ be i.i.d.r.v. sampled from $\mathcal{Q}$, and $f : \mathcal{X} \to \mathbb{R}$ be a bounded Borel function ($\|f\|_\infty < +\infty$). Then, for any $\lambda > 0$ and N large enough such that $\lambda > \|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})/\sqrt{N}$, the following bound holds with probability at least $1 - 3\exp(-N \cdot t^2(\lambda, N))$:*

$$|\,\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}}| \leq \lambda \tag{A.1}$$

where $t(\lambda, N)$ is defined as:

$$t(\lambda, N) \equiv \frac{\lambda}{\|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})} - \frac{1}{\sqrt{N}} \tag{A.2}$$

Proof. This proof follows similar proof steps as in Metelli *et al.* [126]. Since we have upper bounds on the infinite Rényi divergence $d_\infty(\mathcal{P}||\mathcal{Q})$, we can start from the Hoeffding's inequality for bounded random variables applied to the regular IS estimator $\hat{\mu}_{\mathcal{P}/\mathcal{Q}} = \frac{1}{N}\sum_{i=1}^{N} w_{\mathcal{P}/\mathcal{Q}}(x_i)f(x_i)$, which is unbiased. While applying the Hoeffding's inequality, we can view importance sampling on $f(x)$ weighted by $w_{\mathcal{P}/\mathcal{Q}}(x)$ as Monte Carlo sampling on

$g(x) = w_{\mathcal{P}/\mathcal{Q}}(x)f(x)$, which is a function bounded by $\|g\|_\infty = d_\infty(\mathcal{P}||\mathcal{Q})\|f\|_\infty$:

$$\mathbb{P}\left(\hat{\mu}_{\mathcal{P}/\mathcal{Q}} - \mathbb{E}_{x\sim P}[f(x)] \geq \lambda\right) = \mathbb{P}\left(\hat{\mu}_{\mathcal{P}/\mathcal{Q}} - \mathbb{E}_{x\sim Q}[\hat{\mu}_{\mathcal{P}/\mathcal{Q}}(x)f(x)] \geq \lambda\right) \tag{A.3}$$

$$\leq \exp\left(-\frac{2N^2\lambda^2}{\sum_{i=1}^N 2(d_\infty(\mathcal{P}||\mathcal{Q})\|f\|_\infty)^2}\right) \tag{A.4}$$

$$\leq \exp\left(-\frac{N\lambda^2}{d_\infty^2(\mathcal{P}||\mathcal{Q})\|f\|_\infty^2}\right) \equiv \delta \tag{A.5}$$

$$\mathbb{P}\left(|\hat{\mu}_{\mathcal{P}/\mathcal{Q}} - \mathbb{E}_{x\sim P}[f(x)]| \geq \lambda\right) \leq 2\exp\left(-\frac{N\lambda^2}{d_\infty^2(\mathcal{P}||\mathcal{Q})\|f\|_\infty^2}\right) = 2\delta \tag{A.6}$$

We prove a similar bound for the SN estimator $\tilde{\mu}_{\mathcal{P}/\mathcal{Q}} = \sum_{i=1}^N \tilde{w}_{\mathcal{P}/\mathcal{Q}}(x_i)f(x_i)$, which is a biased estimator. However, we need to take a step further and analyze the absolute difference, requiring us to split the difference up into two terms:

$$\mathbb{P}(|\,\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}}| \geq \lambda) \tag{A.7}$$

$$\leq \mathbb{P}(\tilde{\mu}_{\mathcal{P}/\mathcal{Q}} - \mathbb{E}_{x\sim\mathcal{P}}[f(x)] \geq \lambda) + \mathbb{P}(\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}} \geq \lambda) \tag{A.8}$$

$$\leq \mathbb{P}(\tilde{\mu}_{\mathcal{P}/\mathcal{Q}} - \mathbb{E}_{x\sim\mathcal{Q}}[\tilde{\mu}_{\mathcal{P}/\mathcal{Q}}] \geq \tilde{\lambda}) + \mathbb{P}(\mathbb{E}_{x\sim\mathcal{Q}}[\tilde{\mu}_{\mathcal{P}/\mathcal{Q}}] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}} \geq \tilde{\lambda}) \tag{A.9}$$

$$\leq \tilde{\delta} + \mathbb{P}(\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}} \geq \lambda) \tag{A.10}$$

The first term is bounded by $\tilde{\delta}$ from the above bound and recasting λ to $\tilde{\lambda}$ to account for the bias of the SN estimator:

$$\tilde{\lambda} = \lambda - \left|\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \mathbb{E}_{x\sim\mathcal{Q}}[\tilde{\mu}_{\mathcal{P}/\mathcal{Q}}]\right| \tag{A.11}$$

$$\tilde{\delta} = \exp\left(-\frac{N\tilde{\lambda}^2}{d_\infty^2(\mathcal{P}||\mathcal{Q})\|f\|_\infty^2}\right) \tag{A.12}$$

Note that the bias term in the SN estimator is bounded by following through Cauchy-Schwarz inequality, closely following steps from Metelli *et al.* [126]:

$$\begin{aligned}
|\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \mathbb{E}_{x\sim\mathcal{Q}}[\tilde{\mu}_{\mathcal{P}/\mathcal{Q}}]| &= \left|\mathbb{E}_{x\sim\mathcal{Q}}[\hat{\mu}_{\mathcal{P}/\mathcal{Q}} - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}}]\right| \leq \mathbb{E}_{x\sim\mathcal{Q}}[|\hat{\mu}_{\mathcal{P}/\mathcal{Q}} - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}}|] && \text{(A.13)}\\
&\leq \mathbb{E}_{x\sim\mathcal{Q}}\left|\frac{\sum_{i=1}^N w_{\mathcal{P}/\mathcal{Q}}(x_i) f(x_i)}{\sum_{i=1}^N w_{\mathcal{P}/\mathcal{Q}}(x_i)} - \frac{1}{N}\sum_{i=1}^N w_{\mathcal{P}/\mathcal{Q}}(x_i) f(x_i)\right| && \text{(A.14)}\\
&= \mathbb{E}_{x\sim\mathcal{Q}}\left[\left|\frac{\sum_{i=1}^N w_{\mathcal{P}/\mathcal{Q}}(x_i) f(x_i)}{\sum_{i=1}^N w_{\mathcal{P}/\mathcal{Q}}(x_i)}\right|\left|1 - \frac{\sum_{i=1}^N w_{\mathcal{P}/\mathcal{Q}}(x_i)}{N}\right|\right] && \text{(A.15)}\\
&\leq \mathbb{E}_{x\sim\mathcal{Q}}\left[\left(\frac{\sum_{i=1}^N w_{\mathcal{P}/\mathcal{Q}}(x_i) f(x_i)}{\sum_{i=1}^N w_{\mathcal{P}/\mathcal{Q}}(x_i)}\right)^2\right]^{1/2} \mathbb{E}_{x\sim\mathcal{Q}}\left[\left(1 - \frac{\sum_{i=1}^N w_{\mathcal{P}/\mathcal{Q}}(x_i)}{N}\right)^2\right]^{1/2} && \text{(A.16)}\\
&\leq \|f\|_\infty \sqrt{\frac{d_2(\mathcal{P}||\mathcal{Q}) - 1}{N}} \leq \|f\|_\infty \frac{d_\infty(\mathcal{P}||\mathcal{Q})}{\sqrt{N}} && \text{(A.17)}
\end{aligned}$$

In the last step, the first term is bounded by $\|f\|_\infty$ as the function is bounded, and the second term is bounded by the fact that we can bound the square root of variance with the supremum squared, where we square it for the convenience of the definition of $t(\lambda, N)$ later on such that the $1/\sqrt{N}$ factor is nicely separated. We assume that N is chosen large enough that $\lambda > \|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})/\sqrt{N}$. Using this, we bound the $\tilde{\delta}$ term:

$$\begin{aligned}
\tilde{\delta} &\leq \exp\left(-\frac{N(\lambda - \|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})/\sqrt{N})^2}{d_\infty^2(\mathcal{P}||\mathcal{Q})\|f\|_\infty^2}\right) && \text{(A.18)}\\
&= \exp\left(-N\left(\frac{\lambda - \|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})/\sqrt{N}}{\|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})}\right)^2\right) && \text{(A.19)}\\
&\equiv \exp\left(-N \cdot t^2(\lambda, N)\right) && \text{(A.20)}
\end{aligned}$$

Here, we define $t(\lambda, N) \equiv \frac{\lambda}{\|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})} - \frac{1}{\sqrt{N}}$, which satisfies $0 < t(\lambda, N) \leq \frac{\lambda}{\|f\|_\infty d_\infty(\mathcal{P}||\mathcal{Q})}$. The second term can be bounded similarly by rebounding the bias term with $\tilde{\lambda}$, using symmetry and Hoeffding's inequality:

$$\begin{aligned}
\mathbb{P}(\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}} \geq \lambda) &\leq \mathbb{P}(\mathbb{E}_{x\sim\mathcal{Q}}[\tilde{\mu}_{\mathcal{P}/\mathcal{Q}}] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}} \geq \tilde{\lambda}) && \text{(A.21)}\\
&\leq \mathbb{P}(|\mathbb{E}_{x\sim\mathcal{Q}}[\tilde{\mu}_{\mathcal{P}/\mathcal{Q}}] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}}| \geq \tilde{\lambda}) \leq 2\tilde{\delta} && \text{(A.22)}
\end{aligned}$$

Thus, we obtain the following bound:

$$\mathbb{P}(|\mathbb{E}_{x\sim\mathcal{P}}[f(x)] - \tilde{\mu}_{\mathcal{P}/\mathcal{Q}}| \geq \lambda) \leq 3\exp(-N \cdot t^2(\lambda, N)) \tag{A.23}$$

$\square$

A.2 Proof of Lemma 6.1 (Continued) - SN Estimator Leaf Node Convergence

In the main chapter, we show that $\hat{Q}^*_{D-1}(\bar{b}_{D-1}, a)$ is an SN estimator of $Q^*_{D-1}(b_{D-1}, a)$. We apply the concentration inequality proven in Theorem 5.1 to finish the proof of Lemma 5.1.

Lemma A.1 (SN Estimator Leaf Node Convergence)**.** *$\hat{Q}^*_{D-1}(\bar{b}_{D-1}, a)$ is an SN estimator of $Q^*_{D-1}(b_{D-1}, a)$, and the following leaf-node concentration bound holds with probability at least $1 - 3\exp(-C \cdot t^2_{\max}(\lambda, C))$,*

$$|Q^*_{D-1}(b_{D-1}, a) - \hat{Q}^*_{D-1}(\bar{b}_{D-1}, a)| \leq \lambda \tag{A.24}$$

Proof. We first bound R by $3V_{\max}$, where we define $V_{\max}$:

$$V_{\max} \equiv \frac{R_{\max}}{1-\gamma} \geq R_{\max} \tag{A.25}$$

We make this crude upper bound starting at the leaf node so that the probability upper bound at other subsequent steps will be bounded by the same factor. In addition, since $d_\infty(\mathcal{P}^{D-1}||\mathcal{Q}^{D-1})$ is bounded by $d_\infty^{\max}$ a.s., we can bound the resulting $t_{D-1}(\lambda, C)$ by $t_{\max}(\lambda, C)$ a.s.:

$$t_{D-1}(\lambda, C) = \frac{\lambda}{3V_{\max} d_\infty(\mathcal{P}^{D-1}||\mathcal{Q}^{D-1})} - \frac{1}{\sqrt{C}} \geq \frac{\lambda}{3V_{\max} d_\infty^{\max}} - \frac{1}{\sqrt{C}} \equiv t_{\max}(\lambda, C) \tag{A.26}$$

Note that this algebra holds for all steps $d = 0, \cdots, D-1$, which allows us to say $t_d(\lambda, C) \geq t_{\max}(\lambda, C)$. Thus, bounding the concentration inequality probability with $t_{\max}(\lambda, C)$ is justified when we prove Lemma 5.2 later.

This probabilistic bound holds for any choice of $\{o_n\}_j$, where $\{o_n\}_j$ could be a sequence of random variables correlated with any elements of $\{s_n\}_i$.

$$|Q^*_{D-1}(b_{D-1}, a) - \hat{Q}^*_{D-1}(\bar{b}_{D-1}, a)| \leq \lambda \tag{A.27}$$

Thus, for all $\{o_n\}_j$, $\{a_n\}$ and a fixed a, Eq. (A.27) holds with probability at least $1 - 3\exp(-C \cdot t^2_{\max}(\lambda, C))$. □

A.3 Proof of Lemma 6.2 (Continued) - SN Estimator Step-by-Step Convergence

Lemma A.2 (SN Estimator Step-by-Step Convergence). $\hat{Q}^*_d(\bar{b}_d, a)$ *is an SN estimator of* $Q^*_d(b_d, a)$ *for all* $d = 0, \cdots, D-1$ *and* a*, and the following holds with probability at least* $1 - 3|A|(3|A|C)^D \exp(-C \cdot t^2_{\max})$*:*

$$|Q^*_d(b_d, a) - \hat{Q}^*_d(\bar{b}_d, a)| \leq \alpha_d \tag{A.28}$$

$$\alpha_d \equiv \lambda + \gamma\alpha_{d+1};\ \alpha_{D-1} = \lambda \tag{A.29}$$

In Lemma 2, we split the difference between the SN estimator and the Q^* function into two terms, the reward estimation error (A) and the next-step value estimation error (B):

$$|Q^*_d(b_d, a) - \hat{Q}^*_d(\bar{b}_d, a)| \leq \underbrace{\left|\mathbb{E}[R(s_d, a)|b_d] - \frac{\sum_{i=1}^{C} w_{d,i} r_{d,i}}{\sum_{i=1}^{C} w_{d,i}}\right|}_{\text{(A)}} \tag{A.30}$$

$$+ \gamma \underbrace{\left|\mathbb{E}[V^*_{d+1}(bao)|b_d] - \frac{\sum_{i=1}^{C} w_{d,i} \hat{V}^*_{d+1}(\overline{b_d a o_i})}{\sum_{i=1}^{C} w_{d,i}}\right|}_{\text{(B)}}$$

To bound these terms, we will use the SN concentration bound (Theorem 5.1) 3 times throughout the process.

For (A), we use the SN concentration bound to obtain the bound $\frac{R_{\max}}{3V_{\max}}\lambda$; rather than bounding R with $3V_{\max}$ in this step, we instead bound R with $R_{\max}$ and then augment λ to $\frac{R_{\max}}{3V_{\max}}\lambda$ in order to obtain the same uniform $t_{\max}$ factor as the other steps. This choice of bound is made to effectively combine the λ terms when we add (A) and (B).

For (B), we use the triangle inequality repeatedly to separate it into three terms; the importance sampling error bounded by $\lambda/3$, the Monte Carlo next-step integral approximation error bounded by $2\lambda/3\gamma$, and the function estimation error bounded by α_{d+1}:

$$\begin{aligned}(B) \leq\, & \underbrace{\left| \mathbb{E}[V^*_{d+1}(bao)|b_d] - \frac{\sum_{i=1}^C w_{d,i}\mathbf{V}^*_{d+1}(s_{d,i}, b_d, a)}{\sum_{i=1}^C w_{d,i}} \right|}_{\text{Importance sampling error}} \\ & + \underbrace{\left| \frac{\sum_{i=1}^C w_{d,i}\mathbf{V}^*_{d+1}(s_{d,i}, b_d, a)}{\sum_{i=1}^C w_{d,i}} - \frac{\sum_{i=1}^C w_{d,i}V^*_{d+1}(b_d a o_i)}{\sum_{i=1}^C w_{d,i}} \right|}_{\text{MC next-step integral approximation error}} \\ & + \underbrace{\left| \frac{\sum_{i=1}^C w_{d,i}V^*_{d+1}(b_d a o_i)}{\sum_{i=1}^C w_{d,i}} - \frac{\sum_{i=1}^C w_{d,i}\hat{V}^*_{d+1}(\overline{b_d a o_i})}{\sum_{i=1}^C w_{d,i}} \right|}_{\text{Function estimation error}} \end{aligned} \tag{A.31}$$

$$\leq \frac{1}{3}\lambda + \frac{2}{3\gamma}\lambda + \alpha_{d+1} \tag{A.32}$$

The following subsections justify how each of the error terms are bounded.

A.3.1 Importance Sampling Error

Before we analyze the first term, note that the conditional expectation of the optimal value function at step $d+1$ given b_d, a is calculated by the following, where we introduce $\mathbf{V}^*_{d+1}(s_{d,i}, b_d, a)$ as a shorthand for the next-step integration over (s_{d+1}, o) conditioned on $(s_{d,i}, b_d, a)$:

$$\mathbf{V}^*_{d+1}(s_{d,i}, b_d, a) \equiv \int_S \int_O V^*_{d+1}(b_d a o)\mathcal{Z}(o|a, s_{d+1})\mathcal{T}(s_{d+1}|s_{d,i}, a) ds_{d+1} do \tag{A.33}$$

$$\begin{aligned}\mathbb{E}[V^*_{d+1}(bao)|b_d] &= \int_S \int_S \int_O V^*_{d+1}(b_d a o)(\mathcal{Z}_{d+1})(\mathcal{T}_{d,d+1}) b_d \cdot ds_{d:d+1} do && \text{(A.34)} \\ &= \int_S \mathbf{V}^*_{d+1}(s_d, b_d, a) b_d \cdot ds_d && \text{(A.35)} \\ &= \frac{\int_{S^{d+1}} \mathbf{V}^*_{d+1}(s_d, b_d, a)(\mathcal{Z}_{1:d})(\mathcal{T}_{1:d}) b_0 ds_{0:d}}{\int_{S^{d+1}} (\mathcal{Z}_{1:d})(\mathcal{T}_{1:d}) b_0 ds_{0:d}} && \text{(A.36)}\end{aligned}$$

Noting that the first term is then the difference between the SN estimator and the conditional expectation, and that $||\mathbf{V}^*_{d+1}||_\infty \leq V_{\max}$, we can apply the SN inequality for the second time in Lemma 2 to bound it by the augmented $\lambda/3$.

A.3.2 Monte Carlo Next-Step Integral Approximation Error

The second term can be thought of as Monte Carlo next-step integral approximation error. To estimate $\mathbf{V}^*_{d+1}(s_{d,i}, b_d, a)$, we can simply use the quantity $V^*_{d+1}(b_d a o_i)$, as the random vector $(s_{d+1,i}, o_i)$ is jointly generated using G according to the correct probability

$\mathcal{Z}(o|a, s_{d+1})\mathcal{T}(s_{d+1}|s_{d,i}, a)$ given $s_{d,i}$ in the POWSS simulation. Consequently, the quantity $V^*_{d+1}(b_d a o_i)$ for a given $(s_{d,i}, b_d, a)$ is an unbiased 1-sample MC estimate of $\mathbf{V}^*_{d+1}(s_{d,i}, b_d, a)$. We define the difference between these two quantities as Δ_{d+1}, which is implicitly a function of random variables $(s_{d+1,i}, o_i)$:

$$\Delta_{d+1}(s_{d,i}, b_d, a) \equiv \mathbf{V}^*_{d+1}(s_{d,i}, b_d, a) - V^*_{d+1}(b_d a o_i) \tag{A.37}$$

Then, we note that $||\Delta_{d+1}||_\infty \leq 2V_{\max}$ and $\mathbb{E}\,\Delta_{d+1} = 0$ by the Tower property conditioning on $(s_{d,i}, b_d, a)$ and integrating over $(s_{d+1,i}, o_i)$ first, which holds for any choice of well-behaved sampling distributions on $\{s_{0:d}\}_i$. Using this fact, we can then consider the second term as an SN estimator for the bias $\mathbb{E}\,\Delta_{d+1} = 0$, and use our SN concentration bound for the third time. Since $||\Delta_{d+1}||_\infty \leq 2V_{\max}$, our λ factor is then augmented by 2/3:

$$\begin{aligned}
&\left| \frac{\sum_{i=1}^C w_{d,i} \mathbf{V}^*_{d+1}(s_{d,i}, b_d, a)}{\sum_{i=1}^C w_{d,i}} - \frac{\sum_{i=1}^C w_{d,i} V^*_{d+1}(b_d a o_i)}{\sum_{i=1}^C w_{d,i}} \right| \\
&= \left| \frac{\sum_{i=1}^C w_{d,i} \Delta_{d+1}(s_{d,i}, b_d, a)}{\sum_{i=1}^C w_{d,i}} - 0 \right| \leq \frac{2}{3}\lambda \leq \frac{2}{3\gamma}\lambda
\end{aligned} \tag{A.38}$$

A.3.3 Function Estimation Error

Lastly, the third term is bounded by the inductive hypothesis, since each i-th absolute difference of the Q-function and its estimate at step $d+1$, and furthermore the value function and its estimate at step $d+1$, are all bounded by α_{d+1}.

A.4 Proof of Theorem 6.3 - POWSS Policy Convergence

A.4.1 Belief State Policy Convergence Lemma

Before we prove Theorem 5.3, we first prove the following lemma, which is an adaptation of Kearns *et al.* [92] and Singh and Yee [161] for belief states b.

Lemma A.4. *Suppose we obtain a greedy policy implemented by some approximation $\tilde{V}_{d,t}(b) \approx V^*_{t+d}(b)$ by generating a tree at online step t and obtaining a value function estimate at tree depth d for a belief b. Define the total loss $L_{\tilde{V},t} \equiv V^*_t(b) - V_{\tilde{V}_{0,t}}(b)$ as the difference between the value obtained by the optimal policy and the value obtained by the $\tilde{V}_{d,t}$ approximation at online step t. If $|\tilde{V}_{d,t}(b) - V^*_{t+d}(b)| \leq \beta$ for all online steps $t \in [0, D-1]$ and its corresponding tree depth $d = 0, \cdots, D-1-t$, then the total loss by implementing the greedy policy from the beginning is bounded by the following:*

$$L_{\tilde{V},0}(b) \leq \frac{3}{1-\gamma}\beta \tag{A.39}$$

Proof. We mirror the proof strategies given in Kearns *et al.* [92] and Singh and Yee [161] for belief states b.

Consider the optimal action $a = \pi^*_d(b)$ and the greedy action $\tilde{a} = \pi_{\tilde{V},t}(b)$. Here, we denote $R(b,a)$ as the shorthand notation for $\mathbb{E}[R(s,a)|b]$. Since $\tilde{a}$ is greedy, it must look at least as good as a under $\tilde{V}$:

$$R(b,a) + \gamma\,\mathbb{E}[\tilde{V}_{1,t}(bao)|b] \leq R(b,\tilde{a}) + \gamma\,\mathbb{E}[\tilde{V}_{1,t}(b\tilde{a}\tilde{o})|b] \tag{A.40}$$

Since we have $|\tilde{V}_{d,t}(b) - V^*_{t+d}(b)| \leq \beta$,

$$R(b,a) + \gamma\,\mathbb{E}[V^*_{t+1}(bao) - \beta|b] \leq R(b,\tilde{a}) + \gamma\,\mathbb{E}[V^*_{t+1}(b\tilde{a}\tilde{o}) + \beta|b] \tag{A.41}$$

$$R(b,a) - R(b,\tilde{a}) \leq 2\gamma\beta + \gamma\,\mathbb{E}[V^*_{t+1}(b\tilde{a}\tilde{o})|b] - \gamma\,\mathbb{E}[V^*_{t+1}(bao)|b] \tag{A.42}$$

Then, the loss for b at time t is:

$$L_{\tilde{V},t}(b) = V^*_t(b) - V_{\tilde{V}_{0,t}}(b) \tag{A.43}$$

$$= R(b,a) - R(b,\tilde{a}) + \gamma\,\mathbb{E}[V^*_{t+1}(bao)|b] - \gamma\,\mathbb{E}[V_{\tilde{V}_{0,t+1}}(b\tilde{a}\tilde{o})|b] \tag{A.44}$$

Substituting the reward function into the loss expression,

$$L_{\tilde{V},t}(b) = R(b,a) - R(b,\tilde{a}) + \gamma\,\mathbb{E}[V^*_{t+1}(bao)|b] - \gamma\,\mathbb{E}[V_{\tilde{V}_{0,t+1}}(b\tilde{a}\tilde{o})|b] \tag{A.45}$$

$$\leq 2\gamma\beta + \gamma\,\mathbb{E}[V^*_{t+1}(b\tilde{a}\tilde{o})|b] - \gamma\,\mathbb{E}[V^*_{t+1}(bao)|b] + \gamma\,\mathbb{E}[V^*_{t+1}(bao)|b] - \gamma\,\mathbb{E}[V_{\tilde{V}_{0,t+1}}(b\tilde{a}\tilde{o})|b] \tag{A.46}$$

$$\leq 2\gamma\beta + \gamma\,\mathbb{E}[V^*_{t+1}(b\tilde{a}\tilde{o})|b] - \gamma\,\mathbb{E}[V_{\tilde{V}_{0,t+1}}(b\tilde{a}\tilde{o})|b] \tag{A.47}$$

$$\leq 2\gamma\beta + \gamma\,\mathbb{E}[L_{\tilde{V},t+1}(b\tilde{a}\tilde{o})|b] \tag{A.48}$$

Note that we have $L_{\tilde{V},D-1}(b) \leq \beta$ from the root node estimate at the last step, which means we obtain the bound with some over-approximations:

$$L_{\tilde{V},0}(b) \leq \sum_{d=1}^{D-1} 2\beta\gamma^d + \gamma^{D-1}\beta \leq \sum_{d=0}^{D-1} 3\beta\gamma^d \leq \frac{3}{1-\gamma}\beta \tag{A.49}$$

These over-approximations are done in order to generate constants that can be easily calculated. □

A.4.2 Proof of Theorem 3

We reiterate the conditions and Theorem 5.3 below:

(i) S and O are continuous spaces, and the action space has a finite number of elements, $|A| < +\infty$.

(ii) For any observation sequence $\{o_n\}_j$, the densities $\mathcal{Z}, \mathcal{T}, b_0$ are chosen such that the Rényi divergence of the target distribution $\mathcal{P}^d$ and sampling distribution $\mathcal{Q}^d$ (Eqs. (5.15) and (5.16)) is bounded above by $d_\infty^{\max} < +\infty$ a.s. for all $d = 0, \cdots, D-1$:

$$d_\infty(\mathcal{P}^d || \mathcal{Q}^d) = \text{ess sup}_{x \sim \mathcal{Q}^d} w_{\mathcal{P}^d/\mathcal{Q}^d}(x) \leq d_\infty^{\max}$$

(iii) The reward function R is Borel and bounded by a finite constant $||R||_\infty \leq R_{\max} < +\infty$ a.s., and $V_{\max} \equiv \frac{R_{\max}}{1-\gamma} < +\infty$.

(iv) We can evaluate the generating function G as well as the observation probability density $\mathcal{Z}$.

(v) The POMDP terminates after $D < \infty$ steps.

Theorem A.3 (POWSS Policy Convergence)**.** *In addition to conditions (i)-(v), assume that the closed-loop POMDP Bayesian belief update step is exact. Then, for any $\epsilon > 0$, we can choose a C such that the value obtained by POWSS is within ϵ of the optimal value function at b_0 a.s.:*

$$V^*(b_0) - V^{\text{POWSS}}(b_0) \leq \epsilon \tag{A.50}$$

Proof. In our main report, we have proved Lemmas 5.1 and 5.2, which gets us the root node convergence for all actions. We apply these lemmas as well as Lemma A.4 to prove the policy convergence.

From Lemma 5.2, we have that the error in estimating Q^* with our POWSS policy is bounded by $\lambda/(1-\gamma)$ for all d, a with probability at least $1-\delta$. This directly implies that the V-function estimation errors are bounded as well for all steps d; if $|Q_d^*(b_d, a) - \hat{Q}_d^*(\bar{b}_d, a)| \leq \frac{\lambda}{1-\gamma}$ for all d, a, then:

$$|\max_{a\in A} Q_d^*(b_d, a) - \max_{a\in A} \hat{Q}_d^*(\bar{b}_d, a)| = |V_d^*(b_d) - \hat{V}_d^*(\bar{b}_d)| \leq \frac{\lambda}{1-\gamma} \tag{A.51}$$

For online planning, we require that the POWSS trees generated at each online planning step must satisfy the concentration inequalities for all of its nodes. As each of the trees generated for D steps need to have good estimates, we worst-case upper bound the union bound probability by multiplying D to δ. Applying Lemma A.4, we get that if all the nodes satisfy the concentration inequality, which happens with probability at least $1 - D\delta$, the following holds:

$$V^*(b_0) - V^{\text{POWSS}}(b_0) = L_{\hat{V}^*,0}(b_0) \leq \frac{3\lambda}{(1-\gamma)^2} \tag{A.52}$$

Note that the maximum difference between the values obtained by the two policies is bounded by $2V_{\max}$. At each online step, you can have $2R_{\max}$ as the maximum possible difference between the two rewards the agent can obtain via the greedy POWSS policy and the optimal policy generated at each online planning step, and at each online step there exists a discount γ. We can use this bound for the bad case probability $D\delta$. Using all the definitions of the constants defined in Theorem 5.2:

$$V^*(b_0) - V^{\text{POWSS}}(b_0) = \mathbb{E}\left[\sum_{i=0}^{D-1} \gamma^i R(s_i, \pi_i^*(s_i)) \middle| b_0\right] - \mathbb{E}\left[\sum_{i=0}^{D-1} \gamma^i R(s_i, \pi_i^{\text{POWSS}}(s_i)) \middle| b_0\right] \tag{A.53}$$

$$\leq (1 - D\delta)\frac{3\lambda}{(1-\gamma)^2} \tag{A.54}$$

$$+ D\delta \sup\left\{\sum_{i=0}^{D-1} \gamma^i |R(s_i, \pi_i^*(s_i)) - R(s_i, \pi_i^{\text{POWSS}}(s_i))| \middle| b_0\right\}$$

$$= (1 - D\delta)\frac{3\lambda}{(1-\gamma)^2} + D\delta \sum_{i=0}^{D-1} \gamma^i (2R_{\max}) \tag{A.55}$$

$$\leq (1 - D\delta)\frac{3\lambda}{(1-\gamma)^2} + D\delta \frac{2R_{\max}}{1-\gamma} \tag{A.56}$$

$$\leq \frac{3\lambda}{(1-\gamma)^2} + 2D\delta V_{\max} = \frac{5\lambda}{(1-\gamma)^2} = \epsilon \tag{A.57}$$

Therefore, we obtain our desired bound on the values obtained by POWSS policy. □

Appendix B

Details for Optimality Guarantees for Particle Belief Approximation of POMDPs

B.1 Proof of Lemma 7.1 (Continued) - Particle Likelihood SN Estimator Convergence

In the main chapter, we show that $\tilde{\mu}_{\bar{b}_d}[f]$ is an SN estimator of $\mathbb{E}_{s\sim b_d}[f(s)]$. We apply the concentration inequality proven in Theorem 6.1 to finish the proof of Lemma 6.1.

Lemma B.1 (Particle Likelihood SN Estimator Convergence)**.** *Suppose a function f is bounded by a finite constant $\|f\|_\infty \leq f_{\max}$, and a particle belief state $\bar{b}_d = \{(s_{d,i}, w_{d,i})\}_{i=1}^C$ at depth d represents b_d with particle likelihood weighting that is recursively updated as $w_{d,i} = w_{d-1,i} \cdot \mathcal{Z}(o_d \mid a, s_d)$. Then, for all $d = 0, \dots, D-1$, the following weighted average is the SN estimator of f under the belief b_d corresponding to the actions $\{a_n\}_{n=0}^{d-1}$ and observations $\{o_n\}_{n=1}^d$, for all beliefs $b_d \in B$ that are realizable given the initial belief b_0:*

$$\tilde{\mu}_{\bar{b}_d}[f] = \frac{\sum_{i=1}^C w_{d,i} f(s_{d,i})}{\sum_{i=1}^C w_{d,i}}, \tag{B.1}$$

and the following concentration bound holds with probability at least $1-3\exp(-C\cdot t_{\max}^2(\lambda, C))$,

$$|\,\mathbb{E}_{s\sim b_d}[f(s)] - \tilde{\mu}_{\bar{b}_d}[f]| \leq \lambda, \tag{B.2}$$

$$t_{\max}(\lambda, C) \equiv \frac{\lambda}{f_{\max} d_\infty^{\max}} - \frac{1}{\sqrt{C}}. \tag{B.3}$$

Proof. In this proof, we will take advantage of the fact that the state particles trajectories $\{s_n\}_1, \dots \{s_n\}_C$ of depth d are independent of each other, as GenPF independently generates each state sequence i according to the transition density $\mathcal{T}$.

In the subsequent analysis, we abbreviate some terms of interest with the following notation:

$$\mathcal{T}^i_{1:d} \equiv \prod_{n=1}^{d} \mathcal{T}(s_{n,i} \mid s_{n-1,i}, a_n); \quad \mathcal{Z}^{i,j}_{1:d} \equiv \prod_{n=1}^{d} \mathcal{Z}(o_{n,j} \mid a_n, s_{n,i}). \tag{B.4}$$

Here d denotes the depth, i denotes the index of the state sample, and j denotes the index of the observation sample. Absence of indices i, j means that $\{s_n\}$ and/or $\{o_n\}$ appear as regular variables. Intuitively, $\mathcal{T}^i_{1:d}$ is the transition density of state sequence i from the root node to depth d, and $\mathcal{Z}^{i,j}_{1:d}$ is the conditional density of observation sequence j given state sequence i from the root node to depth d. Additionally, b^i_d denotes $b_d(s_{d,i})$ and $w_{d,i}$ the weight of $s_{d,i}$.

First, we show that $\tilde{\mu}_{\bar{b}_d}[f]$ is an SN estimator of $\mathbb{E}_{s\sim b_d}[f(s)]$. By following the recursive belief update, the belief term can be fully expanded:

$$b_{D-1}(s_{D-1}) = \frac{\int_{S^{D-1}} (\mathcal{Z}_{1:D-1})(\mathcal{T}_{1:D-1}) b_0 ds_{0:D-2}}{\int_{S^D} (\mathcal{Z}_{1:D-1})(\mathcal{T}_{1:D-1}) b_0 ds_{0:D-1}} \tag{B.5}$$

Then, $\mathbb{E}_{s\sim b_d}[f(s)]$ is equal to the following:

$$\mathbb{E}_{s\sim b_d}[f(s)] = \int_S f(s_{D-1}) b_{D-1} ds_{D-1} = \frac{\int_{S^D} f(s_{D-1})(\mathcal{Z}_{1:D-1})(\mathcal{T}_{1:D-1}) b_0 ds_{0:D-1}}{\int_{S^D} (\mathcal{Z}_{1:D-1})(\mathcal{T}_{1:D-1}) b_0 ds_{0:D-1}} \tag{B.6}$$

We approximate the $\mathbb{E}_{s\sim b_d}[f(s)]$ function with importance sampling by using problem requirement (iv), where the target density is b_{D-1}. First, we sample the sequences $\{s_{n,i}\}$ according to the joint probability $(\mathcal{T}_{1:D-1})b_0$. Afterwards, we weight the sequences by the corresponding observation density $\mathcal{Z}_{1:D-1}$, obtained from the generated observation sequences $\{o_{n,j}\}$. Normally, these generated observation sequences through GenPF will be correlated. For now, we assume the observation sequences $\{o_{n,j}\}$ are fixed.

Applying the importance sampling formalism to our system for all depths $d = 0, \ldots, D-1$, $\mathcal{P}^d$ is the normalized measure incorporating the probability of observation sequence j on top of the state sequence i and action sequence until the node at depth d, and $\mathcal{Q}^d$ is the measure of the state sequence. We can think of $\mathcal{P}^d$ corresponding to the observation sequence $\{o_{n,j}\}$.

$$\mathcal{P}^d = \mathcal{P}^d_{\{a_n, o_{n,j}\}}(\{s_{n,i}\}) = \frac{(\mathcal{Z}^{i,j}_{1:d})(\mathcal{T}^i_{1:d}) b^i_0}{\int_{S^{d+1}} (\mathcal{Z}^j_{1:d})(\mathcal{T}_{1:d}) b_0 ds_{0:d}} \tag{B.7}$$

$$\mathcal{Q}^d = \mathcal{Q}^d_{\{a_n\}}(\{s_{n,i}\}) = (\mathcal{T}^i_{1:d}) b^i_0 \tag{B.8}$$

$$w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\}) = \frac{(\mathcal{Z}^{i,j}_{1:d})}{\int_{S^{d+1}} (\mathcal{Z}^j_{1:d})(\mathcal{T}_{1:d}) b_0 ds_{0:d}} \tag{B.9}$$

Here, the integral to calculate the normalizing constant is taken over S^{d+1}, the Cartesian product of the state space S over $d+1$ steps.

The weighing step is done by updating the self-normalized weights given in GenPF algorithm. We define $w_{d,i}$ and $r_{d,i}$ as the weights and rewards obtained at step d for state sequence i from GenPF simulation. With our recursive definition of the empirical weights, we obtain the full weight of each state sequence i for a fixed observation sequence j:

$$w_{d,i} = w_{d-1,i} \cdot \mathcal{Z}(o_{d,j} \mid a_d, s_{d,i}) \propto \mathcal{Z}^{i,j}_{1:d}. \tag{B.10}$$

Realizing that the marginal observation probability is independent of indexing by i, we show that $\tilde{\mu}_{\bar{b}_d}[f]$ is an SN estimator of $\mathbb{E}_{s\sim b_d}[f(s)]$:

$$\tilde{\mu}_{\bar{b}_d}[f] = \frac{\sum_{i=1}^{C}(\mathcal{Z}^{i,j}_{1:d})f(s_{d,i})}{\sum_{i=1}^{C}(\mathcal{Z}^{i,j}_{1:d})} = \frac{\sum_{i=1}^{C}\frac{(\mathcal{Z}^{i,j}_{1:d})}{\int_{S^D}(\mathcal{Z}^{j}_{1:d})(\mathcal{T}_{1:d})b_0 ds_{0:d}}f(s_{d,i})}{\sum_{i=1}^{C}\frac{(\mathcal{Z}^{i,j}_{1:d})}{\int_{S^D}(\mathcal{Z}^{j}_{1:d})(\mathcal{T}_{1:d})b_0 ds_{0:d}}} \tag{B.11}$$

$$= \frac{\sum_{i=1}^{C} w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\})f(s_{d,i})}{\sum_{i=1}^{C} w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\})} = \sum_{i=1}^{C}\tilde{w}_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_{n,i}\})f(s_{d,i}) \tag{B.12}$$

Since $\{s_n\}_1, \ldots, \{s_n\}_C$ are i.i.d.r.v. sequences of depth d, and f is a bounded function, we can apply the SN concentration bound in Theorem 6.1 to obtain the concentration inequality. Since $d_\infty(\mathcal{P}^d||\mathcal{Q}^d)$ is bounded by $d^{\max}_\infty$ a.s., we can bound the resulting $t_d(\lambda, C)$ by $t_{\max}(\lambda, C)$ a.s.:

$$t_d(\lambda, C) = \frac{\lambda}{f_{\max} d_\infty(\mathcal{P}^d||\mathcal{Q}^d)} - \frac{1}{\sqrt{C}} \geq \frac{\lambda}{f_{\max} d^{\max}_\infty} - \frac{1}{\sqrt{C}} \equiv t_{\max}(\lambda, C) \tag{B.13}$$

This means that for all d, we can bound $t_d(\lambda, C) \geq t_{\max}(\lambda, C)$. Thus, bounding the concentration inequality probability with $t_{\max}(\lambda, C)$ for any step d is justified when we prove Lemma 6.2 later. This probabilistic bound holds for any choice of $\{o_{n,j}\}$, where $\{o_{n,j}\}$ could be a sequence of random variables correlated with any elements of $\{s_{n,i}\}$. Thus, for any $\{o_{n,j}\}$,

$$|\,\mathbb{E}_{s\sim b_d}[f(s)] - \tilde{\mu}_{\bar{b}_d}[f]| \leq \lambda \tag{B.14}$$

holds with probability at least $1 - 3\exp(-C \cdot t^2_{\max}(\lambda, C))$. □

B.2 Proof of Lemma 7.2 (Continued) - Sparse Sampling-ω Q-Value Coupled Convergence

Lemma B.2 (Sparse Sampling-ω Estimator Q-Value Coupled Convergence)**.** *For all $d = 0, \dots, D-1$ and a, the following bounds hold with probability at least $1 - 6|A|(4|A|C)^D \exp(-C \cdot \tilde{t}^2)$:*

$$|Q^*_{\mathbf{P},d}(b_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \le \alpha_d,\ \alpha_d = \lambda + \gamma\alpha_{d+1},\ \alpha_{D-1} = \lambda, \tag{B.15}$$

$$|Q^*_{\mathbf{M_P},d}(\bar{b}_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \le \beta_d,\ \beta_d = \gamma(\lambda + \beta_{d+1}),\ \beta_{D-1} = 0, \tag{B.16}$$

$$t_{\max}(\lambda, C) = \frac{\lambda}{4V_{\max}d^{\max}_{\infty}} - \frac{1}{\sqrt{C}},\ \tilde{t} = \min\{t_{\max}, \lambda/4\sqrt{2}V_{\max}\} \tag{B.17}$$

Before we proceed with the proof, note that in our definition of $t_{\max}$, we set the maximum of the $f_{\max}$ to be equal to $4V_{\max}$. While this may seem very conservative to bound most reasonable functions resulting from reward and value estimation with 4 times the $V_{\max}$, it serves to uniformly bound the probability for each of the SN estimator terms with convenient coefficients. Furthermore, individual concentration bounds may be adjusted to account for this generous upper bound by multiplying a factor in front of λ.

POMDP Value Convergence: We split the difference between the SN estimator and $Q^*_{\mathbf{P}}$ into two terms, the reward estimation error (A) and the next-step value estimation error (B):

$$\begin{aligned} |Q^*_{\mathbf{P},d}(b_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \le& \underbrace{\left| \mathbb{E}_{\mathbf{P}}[R(s_d, a) \mid b_d] - \frac{\sum_{i=1}^{C} w_{d,i} r_{d,i}}{\sum_{i=1}^{C} w_{d,i}} \right|}_{\text{(A)}} \\ &+ \gamma \underbrace{\left| \mathbb{E}_{\mathbf{P}}[V^*_{\mathbf{P},d+1}(b_d a o) \mid b_d] - \frac{1}{C}\sum_{i=1}^{C} \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1}) \right|}_{\text{(B)}} \end{aligned} \tag{B.18}$$

Here, the I_i notation represents that random variables I_i are sampled C times from the finite discrete distribution $p_{w,d}$ with probability mass $p_{w,d}(I = i) = (w_{d,i}/\sum_j w_{d,j})$, and particle belief state $\bar{b}'^{[I_i]}_{d+1}$ is updated by an observation generated from s_{d,I_i}. This reflects the fact that GenPF randomly selects a state particle s_o with probability $w_{d,o}/\sum_j w_{d,j}$ C times independently to generate a new observation for the next step particle belief state. Similarly, a particle belief state $\bar{b}'^{[i]}_{d+1}$ is updated by an observation generated from $s_{d,i}$, which is a notation we will use to represent beliefs that are generated through iterating upon each state particle $s_{d,i}$.

To prove the base case $d = D-1$, we note that we only need to bound the first term (A) since $d = D-1$ corresponds to the leaf node of Sparse Sampling-ω tree and no further next step value estimation is performed:

$$|Q^*_{\mathbf{P},D-1}(b_{D-1},a) - \hat{Q}^*_{\omega,D-1}(\bar{b}_{D-1},a)| \leq \underbrace{\left|\mathbb{E}_{\mathbf{P}}[R(s_{D-1},a) \mid b_{D-1}] - \frac{\sum_{i=1}^{C} w_{D-1,i} r_{D-1,i}}{\sum_{i=1}^{C} w_{D-1,i}}\right|}_{\text{(A)}}. \tag{B.19}$$

This term is simply a particle likelihood weighted average estimation term where the function is $R(\cdot,a)$, and does not need any inductive step. Below, we will show how to bound both terms (A) and (B), so the base case proof naturally follows from the proof of concentration bound for (A).

For (A), we use the particle likelihood SN concentration bound in Lemma 6.1 to obtain the bound $\frac{R_{\max}}{4V_{\max}}\lambda$; rather than bounding R with $4V_{\max}$ in this step, we instead bound R with $R_{\max}$ and then augment λ to $\frac{R_{\max}}{4V_{\max}}\lambda$ in order to obtain the same uniform $t_{\max}$ factor as the other steps. This choice of bound is made to effectively combine the λ terms when we add (A) and (B). This also covers the base case since $\alpha_{D-1} = \lambda \geq \frac{R_{\max}}{4V_{\max}}\lambda$.

For (B), we use the triangle inequality repeatedly to separate it into four terms; (1) the importance sampling error bounded by $\lambda/4$, (2) the Monte Carlo weighted sum approximation error bounded by $\lambda/4$, (3) the Monte Carlo next-step integral approximation error bounded by $\lambda/2$, and (4) the inductive function estimation error bounded by α_{d+1}:

$$\underbrace{\left|\mathbb{E}_{\mathbf{P}}[V^*_{\mathbf{P},d+1}(b_d a o) \mid b_d] - \frac{1}{C}\sum_{i=1}^{C} \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1})\right|}_{\text{(B)}} \leq \tag{B.20}$$

$$\underbrace{\left|\mathbb{E}_{\mathbf{P}}[V^*_{\mathbf{P},d+1}(b_d a o) \mid b_d] - \frac{\sum_{i=1}^{C} w_{d,i}\mathbf{V}^*_{\mathbf{P},d+1}(b_d,a)^{[i]}}{\sum_{i=1}^{C} w_{d,i}}\right|}_{\text{(1) Importance sampling error}} + \underbrace{\left|\frac{\sum_{i=1}^{C} w_{d,i}\mathbf{V}^*_{\mathbf{P},d+1}(b_d,a)^{[i]}}{\sum_{i=1}^{C} w_{d,i}} - \frac{1}{C}\sum_{i=1}^{C}\mathbf{V}^*_{\mathbf{P},d+1}(b_d,a)^{[I_i]}\right|}_{\text{(2) MC weighted sum approximation error}}$$

$$+ \underbrace{\left|\frac{1}{C}\sum_{i=1}^{C}\mathbf{V}^*_{\mathbf{P},d+1}(b_d,a)^{[I_i]} - \frac{1}{C}\sum_{i=1}^{C} V^*_{\mathbf{P},d+1}(b_d a o^{[I_i]})\right|}_{\text{(3) MC next-step integral approximation error}} + \underbrace{\left|\frac{1}{C}\sum_{i=1}^{C} V^*_{\mathbf{P},d+1}(b_d a o^{[I_i]}) - \frac{1}{C}\sum_{i=1}^{C}\hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1})\right|}_{\text{(4) Inductive function estimation error}}.$$

$$\leq \underbrace{\frac{1}{4}\lambda}_{(1)} + \underbrace{\frac{1}{4}\lambda}_{(2)} + \underbrace{\frac{1}{2}\lambda}_{(3)} + \underbrace{\alpha_{d+1}}_{(4)}. \tag{B.21}$$

The following subsections justify how each error term is bounded.

(1) **Importance Sampling Error:** Before we analyze the first term, note that the conditional expectation of the optimal value function at step $d+1$ given b_d, a is calculated by the following, where we introduce $\mathbf{V}^*_{\mathbf{P},d+1}(b_d,a,s_{d,i}) \equiv \mathbf{V}^*_{\mathbf{P},d+1}(b_d,a)^{[i]}$ as a shorthand for the next-step integration over (s_{d+1},o) conditioned on $(b_d,a,s_{d,i})$. Once again, we denote $[i]$ to indicate that $s_{d,i}$ was the particle chosen to generate the observation o, and

if we are conditioning on a generic particle s_d, then we simply denote all the variables $\mathbf{V}^*_{\mathbf{P},d+1}(b_d, a, s_d)$:

$$\mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[i]} \equiv \int_S \int_O V^*_{\mathbf{P},d+1}(b_d ao)\mathcal{Z}(o \mid a, s_{d+1})\mathcal{T}(s_{d+1} \mid s_{d,i}, a) ds_{d+1} do \tag{B.22}$$

$$\begin{aligned}\mathbb{E}_{\mathbf{P}}[V^*_{\mathbf{P},d+1}(b_d ao) \mid b_d] &= \int_S \int_S \int_O V^*_{\mathbf{P},d+1}(b_d ao)(\mathcal{Z}_{d+1})(\mathcal{T}_{d,d+1}) b_d \cdot ds_{d:d+1} do &\text{(B.23)}\\ &= \int_S \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a, s_d) b_d \cdot ds_d &\text{(B.24)}\\ &= \frac{\int_{S^{d+1}} \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a, s_d)(\mathcal{Z}_{1:d})(\mathcal{T}_{1:d}) b_0 ds_{0:d}}{\int_{S^{d+1}} (\mathcal{Z}_{1:d})(\mathcal{T}_{1:d}) b_0 ds_{0:d}}. &\text{(B.25)}\end{aligned}$$

Noting that the term (1) is then the difference between the SN estimator and the conditional expectation, and that $\left\|\mathbf{V}^*_{\mathbf{P},d+1}\right\|_\infty \leq V_{\max}$, we can apply the SN inequality for the second time in Lemma 2 to bound it by the augmented $\lambda/4$. Thus, with our definition of $t_{\max}$, the bound holds with probability at least $1 - 3\exp(-C \cdot t^2_{\max}(\lambda, C))$.

(2) **Monte Carlo Weighted Sum Approximation Error:** The second term is the error resulting from estimating the sum with a Monte Carlo sum, which can be bounded by a Hoeffding-type bound. First, we assume that all the variables except I are given, which are $\{s_{d,i}, w_{d,i}\}, b_d, a$. Then, we note that $\mathbf{V}^*_{\mathbf{P},d+1}(b_d, a, \cdot)$ is a function bounded by $V_{\max}$. For convenience of notation and conceptual clarity, we will denote $\mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[i]} \equiv \mathbf{V}(i)$, which means the value estimate realization for the i-th state index. Noting that the probability mass is $p_{w,d}(I = i) = (w_{d,i} / \sum_j w_{d,j})$, the Monte Carlo summation error can be simplified as the following:

$$\left| \frac{\sum_{i=1}^C w_{d,i} \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[i]}}{\sum_{i=1}^C w_{d,i}} - \frac{1}{C} \sum_{i=1}^C \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[I_i]} \right| \tag{B.26}$$

$$\implies \left| \sum_{i=1}^C p_{w,d}(I = i) \cdot \mathbf{V}(i) - \frac{1}{C} \sum_{i=1}^C \mathbf{V}(I_i) \right|. \tag{B.27}$$

The first term in the difference is the expectation of $\mathbf{V}(\cdot)$ under the probability measure $p_{w,d}$:

$$\left| \sum_{i=1}^C p_{w,d}(I = i) \cdot \mathbf{V}(i) - \frac{1}{C} \sum_{i=1}^C \mathbf{V}(I_i) \right| = \left| \mathbb{E}_{p_{w,d}}[\mathbf{V}(I)] - \frac{1}{C} \sum_{i=1}^C \mathbf{V}(I_i) \right|. \tag{B.28}$$

This is precisely the form of the double-sided Hoeffding-type bound on the function values $\mathbf{V}(I)$, where a Monte Carlo summation, or the Monte Carlo average in this case, attempts to approximate the expected value. Therefore, we can choose λ such that the absolute difference is bounded by λ with probability at least $1 - 2\exp(-C\lambda^2/2V_{\max}^2)$ for an arbitrary fixed set of $\{s_{d,i}, w_{d,i}\}, b_d, a$:

$$\left| \frac{\sum_{i=1}^C w_{d,i} \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[i]}}{\sum_{i=1}^C w_{d,i}} - \frac{1}{C} \sum_{i=1}^C \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[I_i]} \right| \leq \lambda. \tag{B.29}$$

The previous calculation was done by conditioning on $\{s_{d,i}, w_{d,i}\}, b_d, a$. However, this bound does not depend on the specific values of these weights nor the particle belief sets, since Hoeffding bound only takes advantage of the fact that the random variables I_i are sampled i.i.d. and the corresponding $\mathbf{V}(I_i)$ are bounded. Thus, we can revert this back into a general statement by applying the Tower property, and noting that the expectation of an indicator random variable is the probability of the associated event. By denoting the difference as $\Delta(\{s_{d,i}, w_{d,i}\}, b_d, a, \{I_i\})$, we obtain the unconditional Hoeffding-type bound:

$$\begin{aligned}
&\mathbb{P}\left\{\Delta(\{s_{d,i}, w_{d,i}\}, b_d, a, \{I_i\}) \leq \lambda\right\} && \text{(B.30)}\\
&= \mathbb{E}[\mathbf{1}_{\{\Delta(\{s_{d,i},w_{d,i}\},b_d,a,\{I_i\})\leq\lambda\}}] && \text{(B.31)}\\
&= \mathbb{E}\left[\mathbb{E}\left[\mathbf{1}_{\{\Delta(\{s_{d,i},w_{d,i}\},b_d,a,\{I_i\})\leq\lambda\}} \mid \{s_{d,i}, w_{d,i}\}, b_d, a\right]\right] && \text{(B.32)}\\
&= \mathbb{E}\left[\mathbb{P}\left\{\Delta(\{s_{d,i}, w_{d,i}\}, b_d, a, \{I_i\}) \leq \lambda \mid \{s_{d,i}, w_{d,i}\}, b_d, a\right\}\right] && \text{(B.33)}\\
&\geq \mathbb{E}[1 - 2\exp(-C\lambda^2/2V_{\max}^2)] && \text{(B.34)}\\
&= 1 - 2\exp(-C\lambda^2/2V_{\max}^2). && \text{(B.35)}
\end{aligned}$$

Here, we use the factor augmentation once again to choose $\lambda/4$ such that the absolute difference is bounded by $\lambda/4$ with probability at least $1 - 2\exp(-C\lambda^2/32V_{\max}^2)$, which gets us our desired result:

$$\left| \frac{\sum_{i=1}^C w_{d,i} \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[i]}}{\sum_{i=1}^C w_{d,i}} - \frac{1}{C} \sum_{i=1}^C \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[I_i]} \right| \leq \frac{\lambda}{4}. \tag{B.36}$$

(3) **Monte Carlo Next-Step Integral Approximation Error:** The third term can be thought of as Monte Carlo next-step integral approximation error. To estimate $\mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[I_i]}$, we can simply use the quantity $V^*_{\mathbf{P},d+1}(b_d a o^{[I_i]})$, as the random vector (s_{d+1,I_i}, o_{I_i}) is jointly generated using G according to the correct probability $\mathcal{Z}(o \mid a, s_{d+1})\mathcal{T}(s_{d+1} \mid s_{d,I_i}, a)$ given s_{d,I_i} in the simulation realized in the tree. Consequently, the quantity $V^*_{\mathbf{P},d+1}(b_d a o^{[I_i]})$ for a given (s_{d,I_i}, b_d, a) is an unbiased 1-sample MC estimate of $\mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[I_i]}$. We define the difference between these two quantities as Δ_{d+1}, which is implicitly a function of random variables (s_{d+1,I_i}, o_{I_i}):

$$\Delta_{d+1}(b_d, a)^{[I_i]} \equiv \mathbf{V}^*_{\mathbf{P},d+1}(b_d, a)^{[I_i]} - V^*_{\mathbf{P},d+1}(b_d a o^{[I_i]}). \tag{B.37}$$

Then, we note that $\|\Delta_{d+1}\|_\infty \leq 2V_{\max}$ and $\mathbb{E}\,\Delta_{d+1} = 0$ by the Tower property conditioning on (s_{d,I_i}, b_d, a) (which is implicitly conditioning on I_i, but this does not matter greatly as everything cancels out) and integrating over (s_{d+1,I_i}, o_{I_i}) first, which holds for any choice of well-behaved sampling distributions on $\{s_{0:d}\}_i$. Using this fact, we can then consider this term as a Monte Carlo estimator for the bias $\mathbb{E}\,\Delta_{d+1} = 0$, and use another Hoeffding bound. Since $\|\Delta_{d+1}\|_\infty \leq 2V_{\max}$, our λ factor is then augmented by 1/2 to once again obtain probability at least $1 - 2\exp(-C\lambda^2/32V_{\max}^2)$:

$$\left|\frac{1}{C}\sum_{i=1}^{C}\mathbf{V}^*_{\mathbf{P},d+1}(b_d,a)^{[I_i]} - \frac{1}{C}\sum_{i=1}^{C}V^*_{\mathbf{P},d+1}(b_d a o^{[I_i]})\right| \tag{B.38}$$

$$= \left|\frac{1}{C}\sum_{i=1}^{C}(\mathbf{V}^*_{\mathbf{P},d+1}(b_d,a)^{[I_i]} - V^*_{\mathbf{P},d+1}(b_d a o^{[I_i]})) - 0\right|$$

$$= \left|\frac{1}{C}\sum_{i=1}^{C}\Delta_{d+1}(b_d,a)^{[I_i]} - \mathbb{E}\,\Delta_{d+1}\right| \leq \frac{\lambda}{2}. \tag{B.39}$$

(4) **Inductive Function Estimation Error:** The fourth term is bounded by the inductive hypothesis, since each i-th absolute difference of the Q-function and its estimate at step $d+1$, and furthermore the value function and its estimate at step $d+1$, are all bounded by α_{d+1}.

Thus, each of the error terms are bound by $(A) \leq \frac{R_{\max}}{4V_{\max}}\lambda$ and $(B) \leq \frac{1}{4}\lambda + \frac{1}{4}\lambda + \frac{1}{2}\lambda + \alpha_{d+1}$, which uses the SN concentration bound 2 times and Hoeffding bound 2 times. Combining (A) and (B), we can obtain the desired bound:

$$|Q^*_{\mathbf{P},d}(b_d,a) - \hat{Q}^*_d(\bar{b}_d,a)| \leq \frac{R_{\max}}{4V_{\max}}\lambda + \gamma\left[\frac{1}{4}\lambda + \frac{1}{4}\lambda + \frac{1}{2}\lambda + \alpha_{d+1}\right] \tag{B.40}$$

$$\leq \frac{1-\gamma}{4}\lambda + \gamma\left[\frac{1}{4}\lambda + \frac{3}{4\gamma}\lambda + \alpha_{d+1}\right] \tag{B.41}$$

$$= \lambda + \gamma\alpha_{d+1} = \alpha_d. \tag{B.42}$$

Now, we derive the worst case union bound probability. First, we want to ensure that the SN concentration inequality holds with probability $1 - 3\exp(-C \cdot t_{\max}^2(\lambda, C))$ whenever it is used at any given step d and action a. Similarly, we also want to ensure that the Hoeffding-type inequality holds with probability at least $1 - 2\exp(-C\lambda^2/32V_{\max}^2)$ whenever it is used at any given step d and action a. This means we can bound the worst case probability of using either bound by

$$\max(3\exp(-C \cdot t_{\max}^2(\lambda, C)), 2\exp(-C\lambda^2/32V_{\max}^2)) \tag{B.43}$$

$$\leq 3\exp(-C \cdot t_{\max}^2(\lambda, C)) + 2\exp(-C\lambda^2/32V_{\max}^2) \tag{B.44}$$

$$\leq 5\exp(-C \cdot \tilde{t}^2). \tag{B.45}$$

Furthermore, we multiply the worst-case union bound factor $(4|A|C)^D$, since we want the function estimates to be within their respective concentration bounds for all the actions $|A|$ and child nodes C at each step $d = 0, \dots, D-1$, for the 2 times we use SN concentration bound and 2 times we use the double-sided Hoeffding-type bound in the induction step. We once again multiply the final probability by $|A|$ to account for the root node Q-value estimates also satisfying their respective concentration bounds for all actions. Thus, the worst case union bound probability of all bad events is bounded by probability $5|A|(4|A|C)^D \exp(-C \cdot \tilde{t}^2)$. Therefore, we have shown that the concentration bounds for both the particle likelihood SN estimator and Monte Carlo estimator components converge with probability at least $1 - 5|A|(4|A|C)^D \exp(-C \cdot \tilde{t}^2)$ for all levels d:

$$|Q^*_{\mathbf{P},d}(b_d, a) - \tilde{Q}^*_d(\bar{b}_d, a)| \leq \tilde{\alpha}_d. \tag{B.46}$$

PB-MDP Value Convergence: Once again, we split the difference between the SN estimator and the $Q^*_{\mathbf{M_P}}$ function into two terms, the reward estimation error (A) and the next-step value estimation error (B):

$$\begin{aligned} |Q^*_{\mathbf{M_P},d}(\bar{b}_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \leq & \underbrace{\left|\rho(\bar{b}_d, a) - \rho(\bar{b}_d, a)\right|}_{(A)=0} \\ & + \gamma \underbrace{\left| \mathbb{E}_{\mathbf{M_P}}[V^*_{\mathbf{M_P},d+1}(\bar{b}_{d+1}) \mid \bar{b}_d, a] - \frac{1}{C}\sum_{i=1}^{C} \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1}) \right|}_{(B)}. \end{aligned} \tag{B.47}$$

Since our particle belief MDP induces no reward estimation error, the term (A) is always 0 and proving the base case $d = D-1$ is trivial as (A) and (B) are both 0.

We now prove that the difference (B) is bounded for all $d = 0, \dots, D-1$. We use the triangle inequality repeatedly to separate it into two terms; (1) the MC transition approximation error bounded by λ, and (2) the inductive function estimation error bounded by β_{d+1}:

$$\underbrace{\left| \mathbb{E}_{\mathbf{M_P}}[V^*_{\mathbf{M_P},d+1}(\bar{b}_{d+1}) \mid \bar{b}_d, a] - \frac{1}{C}\sum_{i=1}^{C} \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1}) \right|}_{(B)} \tag{B.48}$$

$$\leq \underbrace{\left| \mathbb{E}_{\mathbf{M_P}}[V^*_{\mathbf{M_P},d+1}(\bar{b}_{d+1}) \mid \bar{b}_d, a] - \frac{1}{C}\sum_{i=1}^{C} V^*_{\mathbf{M_P},d+1}(\bar{b}'^{[I_i]}_{d+1}) \right|}_{\text{(1) MC transition approximation error}} + \underbrace{\left| \frac{1}{C}\sum_{i=1}^{C} V^*_{\mathbf{M_P},d+1}(\bar{b}'^{[I_i]}_{d+1}) - \frac{1}{C}\sum_{i=1}^{C} \hat{V}^*_{\omega,d+1}(\bar{b}'^{[I_i]}_{d+1}) \right|}_{\text{(2) Inductive function estimation error}}$$

$$\leq \underbrace{\lambda}_{(1)} + \underbrace{\beta_{d+1}}_{(2)}. \tag{B.49}$$

We justify how each error term is bounded.

(1) **MC Transition Approximation Error:** The Monte Carlo summation over the next step particle belief state samples $\{\bar{b}'^{[I_i]}_{d+1}\}$ given $(\bar{b}_d, a)$ is essentially approximating the integration over the transition density $\tau(\bar{b}_{d+1} \mid \bar{b}_d, a)$. Since the value function and its estimate are both bounded by $V_{\max}$, we can invoke Hoeffding bound here to obtain the following exponential probabilistic bound on the difference:

$$\mathbb{P}\left\{\left|\mathbb{E}_{\mathbf{M_P}}[V^*_{\mathbf{M_P},d+1}(\bar{b}_{d+1}) \mid \bar{b}_d, a] - \frac{1}{C}\sum_{i=1}^{C} V^*_{\mathbf{M_P},d+1}(\bar{b}'^{[I_i]}_{d+1})\right| \leq \lambda\right\} \geq 1 - 2\exp(-C\lambda^2/2V_{\max}^2). \tag{B.50}$$

(2) **Inductive Function Estimation Error:** The second term is bounded by the inductive hypothesis, since each i-th absolute difference of the Q-function and its estimate at step $d+1$, and furthermore the value function and its estimate at step $d+1$, are all bounded by β_{d+1}.

By applying similar logic of ensuring that every particle belief state node and action pairs can satisfy the concentration inequality, we note that the particle belief MDP approximation concentration bound is satisfied with probability at least $1 - |A|(|A|C)^D \exp(-C\lambda^2/2V_{\max}^2)$. Thus, since (A) is 0 and (B) is bounded by $\lambda + \beta_{d+1}$, the Q-value estimation error with respect to $\mathbf{M_P}$ is bounded as desired:

$$|Q^*_{\mathbf{M_P},d}(\bar{b}_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \leq \gamma(\lambda + \beta_{d+1}) = \beta_d. \tag{B.51}$$

Combining both concentration bounds: In order to enable the two concentration inequalities to happen simultaneously, we bound the worst case union probability by using the definition of $\tilde{t}$ and combining the upper bounding terms together:

$$\begin{aligned} 5|A|(4|A|C)^D \exp(-C\cdot\tilde{t}^2)+|A|(|A|C)^D \exp(-C\lambda^2/2V_{\max}^2) & \qquad (B.52)\\ \leq 5|A|(4|A|C)^D \exp(-C\cdot\tilde{t}^2) + |A|(|A|C)^D \exp(-C\cdot\tilde{t}^2) & \qquad (B.53)\\ \leq 5|A|(4|A|C)^D \exp(-C\cdot\tilde{t}^2) + |A|(4|A|C)^D \exp(-C\cdot\tilde{t}^2) & \qquad (B.54)\\ = 6|A|(4|A|C)^D \exp(-C\cdot\tilde{t}^2). & \qquad (B.55) \end{aligned}$$

Therefore, we conclude that the Q-value concentration inequalities for both POMDP approximation error and particle belief approximation error are bounded by α_d, β_d at every node, respectively, with probability at least $1 - 6|A|(4|A|C)^D \exp(-C\cdot\tilde{t}^2)$.

B.3 Proof of Theorem 7.2 - Sparse Sampling-ω Coupled Optimality

We reiterate the conditions and Theorem 6.2 below:

(i) S and O are continuous spaces, and the action space has a finite number of elements, $|A| < +\infty$.

(ii) For any observation sequence $\{o_{n,j}\}_{n=1}^{d}$, the densities $\mathcal{Z}, \mathcal{T}, b_0$ are chosen such that the Rényi divergence of the target $\mathcal{P}^d$ and sampling distribution $\mathcal{Q}^d$ (Eqs. (6.17) and (6.18)) is bounded above by $d_\infty^{\max}$ for all $d = 0, \dots, D-1$:

$$d_\infty(\mathcal{P}^d || \mathcal{Q}^d) = \text{ess sup}_{x\sim\mathcal{Q}^d}\, w_{\mathcal{P}^d/\mathcal{Q}^d}(x) \le d_\infty^{\max} \tag{B.56}$$

(iii) The reward function R is bounded by a finite constant $R_{\max}$, and hence the value function is bounded by $V_{\max} \equiv \frac{R_{\max}}{1-\gamma}$.

(iv) We can sample from the generating function G and evaluate the observation density $\mathcal{Z}$.

(v) The POMDP terminates after no more than $D < \infty$ steps.

(vi) We restrict our analysis to all the beliefs $b \in B$ that are realizable from the initial belief b_0 through Bayesian updates with action sequences $\{a_n\}$ and observation sequences $\{o_n\}$.

Theorem B.1 (Sparse Sampling-ω Coupled Optimality)**.** *Suppose conditions (i)-(vi) are satisfied. Then, for any desired policy optimality $\epsilon > 0$, choosing constants C, λ, δ that satisfy:*

$$\lambda = \epsilon(1-\gamma)^2/8, \tag{B.57}$$

$$\delta = \lambda/(V_{\max} D(1-\gamma)^2), \tag{B.58}$$

$$C = \max\left\{\left(\frac{4V_{\max}d_\infty^{\max}}{\lambda}\right)^2, \frac{64V_{\max}^2}{\lambda^2}\left(D\log\frac{24|A|^{\frac{D+1}{D}}V_{\max}^2 D}{\lambda^2} + \log\frac{1}{\delta}\right)\right\}, \tag{B.59}$$

the Q-function estimates $\hat{Q}^*_{\omega,d}(\bar{b}_d, a)$ obtained for all depths $d = 0, \dots, D-1$, realized beliefs or histories b_d encountered in the Sparse Sampling-ω tree, and actions a are jointly near-optimal with respect to $Q^*_{\mathbf{P},d}$ and $Q^*_{\mathbf{M_P},d}$ with probability at least $1-\delta$:

$$|Q^*_{\mathbf{P},d}(b_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \le \frac{\lambda}{1-\gamma}, \tag{B.60}$$

$$|Q^*_{\mathbf{M_P},d}(\bar{b}_d, a) - \hat{Q}^*_{\omega,d}(\bar{b}_d, a)| \le \frac{\lambda}{1-\gamma}. \tag{B.61}$$

Proof. This proof has two parts. First, we show that the choice of C is valid given the assumptions in Lemma 6.2. Then, we use Lemmas 6.2 and 3A to prove the Q-value estimate claim.

The conditions necessary for C from Lemma 6.2 are the following:

$$t_{\max}(\lambda, C) = \frac{\lambda}{4V_{\max} d_{\infty}^{\max}} - \frac{1}{\sqrt{C}} > 0 \tag{B.62}$$

$$\delta \geq 6|A|(4|A|C)^D \exp(-C \cdot \tilde{t}^2) \tag{B.63}$$

$$\tilde{t}_{\max}(\lambda, C) \equiv \max\left\{t_{\max}(\lambda, C), \lambda/4\sqrt{2}V_{\max}\right\} \tag{B.64}$$

Note that the constraint on $t_{\max}$ implies that the following must be true:

$$\frac{\lambda}{4V_{\max} d_{\infty}^{\max}} - \frac{1}{\sqrt{C}} > 0 \Longrightarrow C > \left(\frac{4V_{\max} d_{\infty}^{\max}}{\lambda}\right)^2, \tag{B.65}$$

which gives us the first option of C in the maximum.

For the next option of C, we show that substituting the formula yields Eq. (B.63). We note that due to the definition of $\tilde{t}_{\max}$, the following is true:

$$\tilde{t}_{\max}(\lambda, C) \geq \lambda/4\sqrt{2}V_{\max}. \tag{B.66}$$

Let us denote $T \equiv (\lambda/4\sqrt{2}V_{\max})^2$ for convenience. Then, since T is upper-bounded by $\tilde{t}_{\max}$,

$$6|A|(4|A|C)^D \exp(-C \cdot \tilde{t}^2) \leq 6|A|(4|A|C)^D \exp(-C \cdot T) \tag{B.67}$$

$$\leq (24|A|^{\frac{D+1}{D}} C)^D \exp(-C \cdot T) \tag{B.68}$$

Consequently, if we show that Eq. (B.68) is bounded by δ, then we automatically show that the original quantity is bounded by δ as well. By defining $X \equiv 24|A|^{\frac{D+1}{D}}$, we want to show that this simplified formula is bounded above by δ:

$$\delta \geq (X \cdot C)^D \exp(-C \cdot T). \tag{B.69}$$

We will show that our second option of C satisfies the following, where the simplified formula equals:

$$\frac{64V_{\max}^2}{\lambda^2}\left(D \log \frac{24|A|^{\frac{D+1}{D}} V_{\max}^2 D}{\lambda^2} + \log\frac{1}{\delta}\right) \Longrightarrow \frac{2}{T}\left(D \log \frac{XD}{T} + \log \frac{1}{\delta}\right) \tag{B.70}$$

Substituting in the second option of C:

$$(X \cdot C)^D \exp(-C \cdot T) = \delta^2 \frac{\left(\frac{2XD}{T} \log \frac{XD}{T} + \frac{2X}{T} \log \frac{1}{\delta}\right)^D}{\left(\frac{XD}{T}\right)^{2D}} \tag{B.71}$$

$$= \delta^2 \frac{\left(\frac{XD}{T} \log \left(\frac{XD}{T}\right)^2 + \frac{XD}{T} \log \left(\frac{1}{\delta}\right)^{2/D}\right)^D}{\left(\frac{XD}{T}\right)^{2D}} \tag{B.72}$$

$$= \delta^2 \frac{\left(\log \left(\frac{XD}{T\delta^{1/D}}\right)^2\right)^D}{\left(\frac{XD}{T}\right)^D} = \delta \left(\frac{\log \left(\frac{XD}{T\delta^{1/D}}\right)^2}{\left(\frac{XD}{T\delta^{1/D}}\right)}\right)^D \tag{B.73}$$

Note that the function $f(x) = \log x^2/x$ is less than 1 for $x > 0$ (in fact, the maximum value of $f(x)$ is exactly $2/e$, attained by setting $x = e$). This means that the quantity inside the parentheses is less than 1, which lets us obtain our desired result

$$(X \cdot C)^D \exp(-C \cdot T) \leq \delta. \tag{B.74}$$

Therefore, each of our option of C satisfies the respective conditions, and taking the maximum of the options will yield valid results for both inequality constraints:

$$C = \max \left\{ \left(\frac{4V_{\max} d_\infty^{\max}}{\lambda}\right)^2, \frac{64V_{\max}^2}{\lambda^2} \left(D \log \frac{24|A|^{\frac{D+1}{D}} V_{\max}^2 D}{\lambda^2} + \log \frac{1}{\delta}\right) \right\}. \tag{B.75}$$

This concludes the first part of the proof; we have shown that C is a valid choice. Next, we prove the value bounds.

With our choice of C, from Lemma 6.2, the error in estimating Q^* with our Sparse Sampling-ω policy is bounded by α_d for all d, a with probability at least $1 - \delta$. Since $\alpha_d \leq \alpha_0$, the following holds for all $d = 0, \ldots, D-1$ with probability at least $1 - \delta$ through Lemmas 6.1 and 6.2:

$$|Q^*_{\mathbf{P},d}(b_d, a) - \hat{Q}^*_d(\bar{b}_d, a)| \leq \alpha_0 \leq \sum_{d=0}^{D-1} \gamma^d \lambda \leq \frac{\lambda}{1-\gamma}, \tag{B.76}$$

$$|Q^*_{\mathbf{M_P},d}(\bar{b}_d, a) - \hat{Q}^*_d(\bar{b}_d, a)| \leq \beta_0 \leq \sum_{d=1}^{D} \gamma^d \lambda \leq \frac{\lambda}{1-\gamma}. \tag{B.77}$$

□

B.4 Proof of Theorem 7.4 - Particle Belief MDP Approximate Policy Convergence

Before we prove Theorem 6.4, we first prove the following lemma, which is an adaptation of Kearns *et al.* (2002) and Singh and Yee (1994) for belief states b.

Lemma 3A. *Consider a POMDP with a finite horizon of D steps and policy $\pi(b) = \arg\max_a \hat{Q}(b,a)$ where $\hat{Q}$ is a stochastic value function approximator with errors bounded by a positive constant ξ: $|Q^*(b,a) - \hat{Q}(b,a)| \leq \xi$. Let $V^\pi(b_0)$ denote the value of executing π starting at belief b_0 with an exact Bayesian belief update, $b_{t+1} = b_t ao$, between each call to the policy. Then*

$$V^*(b_0) - V^\pi(b_0) \leq \frac{2\xi}{1-\gamma}. \tag{B.78}$$

Proof. First, note that if an action is chosen by π, it must appear better than π^* according to $\hat{Q}$, i.e. $\hat{Q}(b,\pi(b)) \geq \hat{Q}(b,\pi^*(b))$. The worst case is when $\hat{Q}(b,\pi(b)) = Q^*(b,\pi(b)) + \xi$ and $\hat{Q}(b,\pi^*(b)) = Q^*(b,\pi^*(b)) - \xi$. Thus, for any t, we have the bound

$$Q^*\left(b_t,\pi^*(b_t)\right) - \mathbb{E}[Q^*\left(b_t,\pi(b_t)\right)] \leq 2\xi. \tag{B.79}$$

Next, we prove that $V^*(b_t) - V^\pi(b_t) \leq \sum_{d=t}^{D-1} \gamma^{d-t} 2\xi$ using induction from $t = D-1$ to $t = 0$. We verify the base case, $t = D-1$, by observing that both $Q^\pi(b_{D-1},\pi(b_{D-1}))$ and $Q^*(b_{D-1},\pi(b_{D-1}))$ are equal to $R(b_{D-1},\pi(b_{D-1}))$ since no further reward can be accumulated and using Eq. (B.79):

$$\begin{aligned}
V^*(b_{D-1}) - V^\pi(b_{D-1}) &= Q^*(b_{D-1},\pi^*(b_{D-1})) - \mathbb{E}[Q^\pi(b_{D-1},\pi(b_{D-1}))] && \text{(B.80)}\\
&= Q^*(b_{D-1},\pi^*(b_{D-1})) - \mathbb{E}[Q^*(b_{D-1},\pi(b_{D-1}))] && \text{(B.81)}\\
&\leq 2\xi. && \text{(B.82)}
\end{aligned}$$

The inductive step is verified by subtracting and adding $\mathbb{E}[Q^*\left(b,\pi(b)\right)]$, using the bound in Eq. (B.79), and applying the inductive hypothesis:

$$\begin{aligned}
V^*(b_t) - V^\pi(b_t) &= Q^*\left(b,\pi^*(b)\right) - \mathbb{E}[Q^\pi(b,\pi(b))] && \text{(B.83)}\\
&= \underbrace{Q^*\left(b,\pi^*(b)\right) - \mathbb{E}[Q^*\left(b,\pi(b)\right)]}_{\text{Bounded by Eq. (B.79)}} + \mathbb{E}[Q^*\left(b,\pi(b)\right)] - \mathbb{E}[Q^\pi(b,\pi(b))] && \text{(B.84)}\\
&\leq 2\xi + \mathbb{E}[Q^*\left(b,\pi(b)\right)] - \mathbb{E}[Q^\pi(b,\pi(b))] && \text{(B.85)}\\
&= 2\xi + \mathbb{E}[R\left(b,\pi(b)\right)] + \gamma\,\mathbb{E}[V^*(b_{t+1}))] - \mathbb{E}[R(b,\pi(b))] - \gamma\,\mathbb{E}[V^\pi(b_{t+1}))] && \text{(B.86)}\\
&= 2\xi + \gamma\,\mathbb{E}[V^*(b_{t+1})) - V^\pi(b_{t+1}))] && \text{(B.87)}\\
&= 2\xi + \gamma \sum_{d=t+1}^{D-1} \gamma^{d-t} 2\xi = \sum_{d=t}^{D-1} \gamma^{d-t} 2\xi. && \text{(B.88)}
\end{aligned}$$

Now, by applying the result above to $t = 0$, we prove the lemma:

$$V^*(b_0) - V^\pi(b_0) \leq \sum_{d=0}^{D-1} \gamma^d 2\xi \leq \frac{2\xi}{1-\gamma}.$$

□

Theorem B.4 (Particle Belief MDP Approximate Policy Convergence)**.** *In addition to regularity conditions for particle belief MDP and the MDP planning algorithm $\mathcal{A}$, assume that the closed-loop POMDP Bayesian belief update step is exact. Then, for any $\epsilon > 0$, we can choose a C such that the value obtained by planning with $\mathcal{A}$ in the PB-MDP is within ϵ of the optimal POMDP value function at b_0:*

$$V^*_{\mathbf{P}}(b_0) - V^{\mathcal{A}}_{\mathbf{M_P}}(b_0) \leq \epsilon. \tag{B.89}$$

Proof. During policy execution, we create a new independent tree and choose an action based on the estimated Q-values. At each of the D steps of the POMDP, there is at most δ probability that $|Q^*_{\mathbf{P},0}(b_0, a) - \hat{Q}^{\mathcal{A}}_{\mathbf{M_P},0}(\bar{b}_0, a)| > \frac{2\lambda}{1-\gamma} + \epsilon_{\mathcal{A}}$. We further choose C', δ', λ' such that $\frac{2\lambda'}{1-\gamma} = \frac{2\lambda}{1-\gamma} + \epsilon_{\mathcal{A}}$.

Thus, with probability at least $1 - D\delta'$, we execute a policy that meets the assumptions of Lemma 3A with $\xi = \frac{2\lambda'}{1-\gamma}$, and hence by Lemma 3A, the difference between the optimal value and the average accumulated reward for this case is at most $\frac{4\lambda'}{(1-\gamma)^2}$. In the other case, which occurs with at most probability $D\delta'$, an arbitrarily bad policy can be executed, resulting in an accumulated reward difference of up to $2V_{\max}$ from the optimal policy. Combining these two cases and using the definition of δ' in Eq. (6.23), we have

$$V^*_{\mathbf{P}}(b_0) - V^{\mathcal{A}}_{\mathbf{M_P}}(b_0) \leq (1 - D\delta')\frac{4\lambda'}{(1-\gamma)^2} + D\delta' 2V_{\max} \tag{B.90}$$

$$\leq \frac{4\lambda'}{(1-\gamma)^2} + \frac{4\lambda'}{(1-\gamma)^2} \tag{B.91}$$

$$= \epsilon. \tag{B.92}$$

□

B.5 Experiment Details

	c_{UCB}	k_a	α_a	k_o	α_o	$m_{\min}$	δ	C	Depth
Laser Tag (D, D, D)									
Sparse-PFT	15	-	-	15	-	-	-	96	37
PFT-DPW	25	-	-	5	0.33	-	-	25	48
POMCPOW	26	-	-	4	0.03	-	-	-	50
AdaOPS	-	-	-	-	-	30	0.1	-	90
POMCP	26	-	-	-	-	-	-	-	50
QMDP	-	-	-	-	-	-	-	-	-
Light Dark (D, D, C)									
Sparse-PFT	95	-	-	24	-	-	-	134	28
PFT-DPW	93	-	-	13	0.08	-	-	33	20
POMCPOW	90	-	-	5	0.07	-	-	-	20
AdaOPS	-	-	-	-	-	30	0.1	-	90
POMCP	83	-	-	-	-	-	-	-	20
QMDP	-	-	-	-	-	-	-	-	-
Sub Hunt (D, D, C)									
Sparse-PFT	20	-	-	27	-	-	-	23	20
PFT-DPW	85	-	-	10	0.08	-	-	79	20
POMCPOW	17	-	-	6	0.01	-	-	-	50
AdaOPS	-	-	-	-	-	30	0.1	-	90
POMCP	17	-	-	-	-	-	-	-	84
QMDP	-	-	-	-	-	-	-	-	-
VDP Tag (C, C, C)									
Sparse-PFT	16	28	-	28	-	-	-	385	33
PFT-DPW	23	22	0.32	21	0.04	-	-	132	44
POMCPOW	110	30	0.03	5	0.01	-	-	-	10
VDP Tag^D (C, D, C)									
Sparse-PFT	76	-	-	25	-	-	-	444	46
PFT-DPW	10	-	-	9	0.11	-	-	330	22
POMCPOW	31	-	-	5	0.05	-	-	-	10
AdaOPS	-	-	-	-	-	40	0.25	-	90

Table B.1: **Summary of hyperparameters used in continuous observation POMDP experiments.**

For UCT methods, we vary c_{UCB}, the UCB exploration parameter. k_a and α_a are action progressive widening parameters, where new actions are added if widening criterion $|C(h)| \leq k_a N(h)^{\alpha_a}$ is met. Similarly, k_o and α_o are observation progressive widening parameters, where new actions are added if widening criterion $|C(ha)| \leq k_o N(ha)^{\alpha_o}$ is met. Sparse-PFT uses $\alpha_a = \alpha_o = 0$. C is the number of particles constituting internal tree beliefs for

PFT methods. m_{min} is the minimum number of particles required to approximate a belief for AdaOPS. Finally, δ is the maximum distance distance between beliefs resulting from observation branches required to merge the branches for AdaOPS.

	$\hat{V}$	L_0	U_0
Laser Tag (D, D, D)			
Sparse-PFT	QMDP PO-Rollout	-	-
PFT-DPW	QMDP PO-Rollout	-	-
POMCPOW	FO-Value	-	-
AdaOPS	-	Random Rollout	QMDP
POMCP	Random Rollout	-	-
QMDP	-	-	-
Light Dark (D, D, C)			
Sparse-PFT	QMDP PO-Rollout	-	-
PFT-DPW	QMDP PO-Rollout	-	-
POMCPOW	FO-Value	-	-
AdaOPS	-	Random Rollout	QMDP
POMCP	Random Rollout	-	-
QMDP	-	-	-
Sub Hunt (D, D, C)			
Sparse-PFT	QMDP PO-Rollout	-	-
PFT-DPW	QMDP PO-Rollout	-	-
POMCPOW	FO-Value	-	-
AdaOPS	-	Random Rollout	QMDP
POMCP	Random Rollout	-	-
QMDP	-	-	-
VDP Tag (C, C, C)			
Sparse-PFT	Random Rollout	-	-
PFT-DPW	Random Rollout	-	-
POMCPOW	Random Rollout	-	-
VDP Tag^D (C, D, C)			
Sparse-PFT	Random Rollout	-	-
PFT-DPW	Random Rollout	-	-
POMCPOW	Random Rollout	-	-
AdaOPS	-	Random Rollout	10^6
POMCP	Random Rollout	-	-

Table B.2: **Summary of leaf node value estimators used in continuous observation POMDP experiments.**

QMDP PO-Rollout corresponds to sampling a "true state" from a leaf node particle belief and simulating the state/belief dynamics with a particle filter as a belief updater and QMDP as a policy. The returns following the trajectory of the sampled "true state" are taken as a value estimate for the leaf node. FO-Value corresponds to using the MDP value for the state representation of a particle. QMDP corresponds to using the belief value estimate given by a QMDP policy. Random Rollout corresponds to sampling a state from a particle belief and simulating it forward using a random policy. The returns of this simulation are used as the initial leaf node value estimate. A constant number (e.g. 10^6) indicates a belief-independent static initial value estimate. UCT solvers only require a single value estimate ($\hat{V}$), whereas AdaOPS requires lower and upper bounds on belief value, L_0 and U_0 respectively.

Appendix C

Details for Voronoi Progressive Widening

C.1 VOWSS Convergence Conditions

Before we prove the VOWSS inequality theorem, we present the necessary conditions for this analysis. The conditions required for the proof is the union of regularity conditions required for POWSS and VOO. The POWSS conditions are the following, with appropriate adaptations to our formalism:

(i) (Continuous spaces) S, A, O are continuous spaces.

(ii) (Bounded Rényi divergence) For any observation sequence $\{o_n\}_j$, the densities $\mathcal{Z}, \mathcal{T}, b_0$ are chosen such that the Rényi divergence of the target distribution $\mathcal{P}^d$ and sampling distribution $\mathcal{Q}^d$ is bounded above by $d_\infty^{\max} < +\infty$ a.s. for all $d = 0, \cdots, D-1$:

$$d_\infty(\mathcal{P}^d || \mathcal{Q}^d) = \text{ess sup}_{x\sim\mathcal{Q}^d} w_{\mathcal{P}^d/\mathcal{Q}^d}(x) \leq d_\infty^{\max}$$

(iii) (Bounded reward function) The reward function R is Borel and bounded by a finite constant $||R||_\infty \leq R_{\max} < +\infty$ a.s., and $V_{\max} \equiv \frac{R_{\max}}{1-\gamma} < +\infty$.

(iv) (Generative model) We can sample from the generating function G and evaluate the observation probability density $\mathcal{Z}$.

(v) (Finite horizon) The POMDP terminates after $D < \infty$ steps.

In our context, the target distribution $\mathcal{P}^d$ corresponds to the conditional observation density of the state trajectory, $\mathcal{Q}^d$ the marginal state trajectory density, and $w_{\mathcal{P}^d/\mathcal{Q}^d}$ the importance weights:

$$\mathcal{P}^d = \mathcal{P}^d_{\{o_n\}_j}(\{s_n\}_i) = \frac{(\mathcal{Z}^{i,j}_{1:d})(\mathcal{T}^i_{1:d})b^i_0}{\int_{S^{d+1}}(\mathcal{Z}^j_{1:d})(\mathcal{T}_{1:d})b_0 ds_{0:d}}, \tag{C.1}$$

$$\mathcal{Q}^d = \mathcal{Q}^d(\{s_n\}_i) = (\mathcal{T}^i_{1:d})b^i_0, \tag{C.2}$$

$$w_{\mathcal{P}^d/\mathcal{Q}^d}(\{s_n\}_i) = \frac{(\mathcal{Z}^{i,j}_{1:d})}{\int_{S^{d+1}}(\mathcal{Z}^j_{1:d})(\mathcal{T}_{1:d})b_0 ds_{0:d}}. \tag{C.3}$$

Essentially, condition (ii) means that the conditional observation density cannot be much larger than the marginal density for any given state trajectory. With the above conditions, the state sampling width C_s is chosen such that the following holds:

$$p \geq 3C_a(3C_a \cdot C_s)^D \exp(-C_s \cdot t^2_{\max}), \tag{C.4}$$

$$p = \frac{\epsilon}{(1-\gamma)V_{\max}D},\ t_{\max}(\epsilon, C_s) = \frac{\epsilon(1-\gamma)}{3V_{\max}d^{\max}_{\infty}} - \frac{1}{\sqrt{C_s}}. \tag{C.5}$$

This ensures the POWSS type bound $|Q^\star_d(b,a) - \hat{Q}^\star_d(\bar{b},a)| \leq \epsilon$ holds with probability at least $1-p$ for each b, a at depth d.

The VOO conditions are the following, with appropriate adaptations to our formalism. Note that with our notation, VOO aims to optimize a function $Q(a)$ over some space A:

(i) (Translation-invariant semi-metric) $D : A \times A \to \mathbb{R}^+$ is such that $\forall x, y, z \in A,\ D(x,y) = D(y,x),\ D(x,y) = 0$ iff $x = y$, and $D(x+z, y+z) = D(x,y)$.

(ii) (Local smoothness of Q) There exists at least one global optimum $a^* \in A$ of Q such that $\forall a \in A, Q(a^*) - Q(a) \leq L \cdot D(a, a^*)$ for some $L > 0$.

(iii) (Shrinkage ratio of the Voronoi cells) Consider any point a' inside the Voronoi cell C generated by the point a_0, and denote $d_0 = D(a', a_0)$. If we randomly sample a point a_1 from C, we have $\mathbb{E}[\min(d_0, D(a', a_1))] \leq \lambda d_0$ for $\lambda \in (0,1)$.

(iv) (Well-shaped Voronoi cells) There exists $\eta > 0$ such that for any Voronoi cell generated by a with expected diameter d_0 contains a ball of radius ηd_0 centered at a.

(v) (Local symmetry near optimum) The set of global optima A_* consists of finite number of disjoint and connected components $\{A^{(l)}_*\}^k_{l=1}, k < \infty$. For each component, there exists an open ball $B_{\nu_l}(a^{(l)}_*)$ for some $a^{(l)}_* \in A^{(l)}_*$ such that $D(a, a^{(l)}_*) \leq D(a', a^{(l)}_*)$ implies $Q(a) \geq Q(a')$ for any $a, a' \in B_{\nu_l}(a^{(l)}_*)$.

Here, we define $\bar{\mu}_B(r) = \mu(B_r(\cdot))/\mu(A)$, with μ the Borel measure on A, $\delta_{\max}$ the largest distance between two points in A, and $\nu_{\min} = \min_{l \in [k]} \nu_l$. Then, the action sampling width C_a is chosen such that the following holds:

$$\omega \geq \frac{1 - \lambda^{1/k}}{\bar{\mu}_B(\nu_{\min}) + 1 - \bar{\mu}_B(\eta \cdot \lambda \nu_{\min})} \tag{C.6}$$

$$C_a \geq \max_{d=0,\cdots,D-1} \left[\log\left(\frac{\eta_{C_a}(d) - \gamma \eta_{C_a}(d+1)}{2L\delta_{\max}C_{\max}} \right) \cdot \min(G_{\lambda,\omega}, K_{\nu,\omega,\lambda}) \right]. \tag{C.7}$$

Here, $C_{\max}, G_{\lambda,\omega}, K_{\nu,\omega,\lambda}$ are problem specific quantities/functions that are explicitly defined in [94]. Satisfying these constraints for a decreasing sequence $\{\eta_{C_a}(d)\}$ will allow us to use the VOO type bound $V_d^\star(b) - \hat{V}_d^\star(b) \leq \eta_{C_a}(d)$ that holds in expectation.

C.2 Proof of Theorem 8.1 - VOWSS Inequality

Theorem C.1 (VOWSS Inequality)**.** *Suppose we choose the action sampling width C_a and state sampling width C_s such that under the union of regularity conditions specified by [94] and [94], the intermediate POWSS bounds and VOO bounds in Lemma 1A are satisfied at every depth of the tree. Then, the following bounds for the VOWSS estimator $\hat{V}_{\text{VOWSS},d}(\bar{b})$ hold for all $d \in [0, D-1]$ in expectation:*

$$\left| V_d^\star(b) - \hat{V}_{\text{VOWSS},d}(\bar{b}) \right| \leq \eta_{C_a} + \alpha_{C_s}. \tag{C.8}$$

In order to prove Theorem C.1, we first prove an intermediate lemma which will allow us to obtain the bound through triangle inequality. We introduce and prove the following lemma first. All of the following calculations are done in expectation. We also denote $\bar{b}$ for a particle representation of belief b that POWSS and VOWSS take as an argument.

Lemma 1A (VOWSS Intermediate Inequality)**.** *Suppose with our notation, the POWSS estimators at all depths d are within ϵ of their mean values with probability $1-p$, and the VOO agents have regret bounds of $\mathcal{R}_{C_a}$. The following inequalities hold for all $d \in [0, D-1]$ in expectation:*

$$\left| V_d^\star(b) - \hat{V}_d^{C_a}(b) \right| \leq \eta_{C_a}(d), \tag{C.9}$$

$$\left| \hat{V}_d^{C_a}(b) - \hat{V}_{\text{VOWSS},d}(\bar{b}) \right| \leq \alpha_{C_s}(d). \tag{C.10}$$

$\eta_{C_a}(d)$ and $\alpha_{C_s}(d)$ are sequences that satisfy the following properties:

$$\eta_{C_a}(d) \geq \gamma \cdot \eta_{C_a}(d+1) + \mathcal{R}_{C_a},\ \eta_{C_a}(D) = 0, \tag{C.11}$$

$$\eta_{C_a} \equiv \max_{d=0,\cdots,D-1} \eta_{C_a}(d) < +\infty, \tag{C.12}$$

$$\alpha_{C_s}(d) \equiv (1+\gamma)(\epsilon + 2p \cdot V_{\max}) + \gamma(\alpha_{C_s}(d+1) + 2p \cdot V_{\max}), \tag{C.13}$$

$$\alpha_{C_s}(D-1) = \epsilon + 2p \cdot V_{\max}, \tag{C.14}$$

$$\alpha_{C_s} \equiv \max_{d=0,\cdots,D-1} \alpha_{C_s}(d) < +\infty. \tag{C.15}$$

Proof. This proof proceeds through induction by assuming that the bounds hold for all depths from d+1 to D−1, and then proving they also hold for depth d (induction hypothesis is omitted as the bounds trivially hold as per the definitions of VOOT and POWSS). We divide the main inequality into VOO bound and POWSS bound by introducing an intermediate term $\hat{V}_d^{C_a}(b)$:

$$\left|V_d^\star(b) - \hat{V}_{\text{VOWSS},d}(b)\right| \le \underbrace{\left|V_d^\star(b) - \hat{V}_d^{C_a}(b)\right|}_{\text{VOO bound}} + \underbrace{\left|\hat{V}_d^{C_a}(b) - \hat{V}_{\text{VOWSS},d}(\bar{b})\right|}_{\text{POWSS bound}} \tag{C.16}$$

Essentially, we have two main layers of inequality, caused by the stochastic nature of VOO action selection, and the uncertainties in state transition and observation. We will first analyze the VOO bound, then the POWSS bound.

Step 1: VOO Bound The VOO bound can once again be further decomposed into the following terms as [94] do in their work:

$$\left|V_d^\star(b) - \hat{V}_d^{C_a}(b)\right| \le \underbrace{\left|V_d^\star(b) - \hat{V}_d(b)\right|}_{\text{VOO Recursive bound}} + \underbrace{\left|\hat{V}_d(b) - \hat{V}_d^{C_a}(b)\right|}_{\text{VOO Regret bound}} \tag{C.17}$$

Here, the intermediate random variables are defined in the following manner:

$$V_d^\star(b) \equiv \max_{a\in A} Q_d^\star(b,a) = \max_{a\in A} \left\{R(b,a) + \gamma\, \mathbb{E}[V_{d+1}^\star(bao)|b]\right\} \tag{C.18}$$

$$\hat{V}_d(b) \equiv \max_{a\in A} \hat{Q}_d(b,a) = \max_{a\in A} \left\{R(b,a) + \gamma\, \mathbb{E}[\hat{V}_{d+1}^{C_a}(bao)|b]\right\} \tag{C.19}$$

$$\hat{V}_d^{C_a}(b) \equiv \max_{a\in VOO(A,C_a)} \hat{Q}_d(b,a) = \max_{a\in VOO(A,C_a)} \left\{R(b,a) + \gamma\, \mathbb{E}[\hat{V}_{d+1}^{C_a}(bao)|b]\right\} \tag{C.20}$$

As a notation, $a \in VOO(A, C_a)$ indicates that the actions a are chosen sequentially through the VOO algorithm over the action space A for C_a iterations of VOO. Here, we compare the two quantities with a reference action $a^\star = \arg\max Q_d^\star(b,a)$, which results in a looser bound but allows us to directly compare the quantities inside the max operations. We closely follow the calculations from the proof of Lemma 5 in [94]:

$$\underbrace{\left|V_d^\star(b) - \hat{V}_d(b)\right|}_{\text{VOO Recursive bound}} = \left|\max_{a\in A}\left\{R(b,a) + \gamma\, \mathbb{E}[V_{d+1}^\star(bao)|b]\right\} - \max_{a\in A}\left\{R(b,a) + \gamma\, \mathbb{E}[\hat{V}_{d+1}^{C_a}(bao)|b]\right\}\right| \tag{C.21}$$

$$\le \left|R(b,a^\star) + \gamma\, \mathbb{E}[V_{d+1}^\star(ba^\star o)|b] - R(b,a^\star) + \gamma\, \mathbb{E}[\hat{V}_{d+1}^{C_a}(ba^\star o)|b]\right|$$

$$(a^\star = \arg\max Q_d^\star(b,a))$$

$$\le \gamma\, \mathbb{E}\left[\left|V_{d+1}^\star(bao) - \hat{V}_{d+1}^{C_a}(bao)\right|\middle| b\right] \tag{C.22}$$

$$\le \gamma \cdot \eta_{C_a}(d+1) \tag{C.23}$$

$$\underbrace{\left|\hat{V}_d(b) - \hat{V}_d^{C_a}(b)\right|}_{\text{VOO Regret bound}} \le \mathcal{R}_{C_a}. \tag{C.24}$$

In the VOO Regret bound, the difference cannot be less than zero since the global maximum is at least as big as the VOO maximum, so the absolute value disappears and we get the regret of VOO. Thus, with our choice of C_a that is designed to satisfy the recurrence relation:

$$\left|V_d^\star(b) - \hat{V}_d^{C_a}(b)\right| \leq \underbrace{\left|V_d^\star(b) - \hat{V}_d(b)\right|}_{\text{VOO Recursive bound}} + \underbrace{\left|\hat{V}_d(b) - \hat{V}_d^{C_a}(b)\right|}_{\text{VOO Regret bound}} \tag{C.25}$$

$$\leq \gamma \cdot \eta_{C_a}(d+1) + \mathcal{R}_{C_a} \leq \eta_{C_a}(d). \tag{C.26}$$

Step 2: POWSS Bound The POWSS bound can also be further decomposed into the following terms:

$$\left|\hat{V}_d^{C_a}(b) - \hat{V}_{\text{VOWSS},d}(b)\right| \leq \underbrace{\left|\hat{V}_d^{C_a}(b) - \tilde{V}_{\text{VOWSS},d}(b)\right|}_{\text{POWSS Concentration bound}} + \underbrace{\left|\tilde{V}_{\text{VOWSS},d}(b) - \hat{V}_{\text{VOWSS},d}(b)\right|}_{\text{POWSS Recursive bound}}. \tag{C.27}$$

Here, the extra intermediate random variable and the VOWSS estimator are defined in the following manner:

$$\tilde{V}_{\text{VOWSS},d}(b) \equiv \max_{a \in VOO(A,C_a)} \tilde{Q}_{\text{VOWSS},d}(b,a) = \max_{a \in VOO(A,C_a)} \left\{ \frac{\sum_{i=1}^{C_s} w_{d,i}(r_{d,i} + \gamma \hat{V}_{d+1}^{C_a}(bao_i))}{\sum_{i=1}^{C_s} w_{d,i}} \right\}, \tag{C.28}$$

$$\hat{V}_{\text{VOWSS},d}(\bar{b}) \equiv \max_{a \in VOO(A,C_a)} \hat{Q}_{\text{VOWSS},d}(\bar{b},a) = \max_{a \in VOO(A,C_a)} \left\{ \frac{\sum_{i=1}^{C_s} w_{d,i}(r_{d,i} + \gamma \hat{V}_{\text{VOWSS},d+1}(\overline{bao_i}))}{\sum_{i=1}^{C_s} w_{d,i}} \right\} \tag{C.29}$$

We compare the two quantities by picking a reference action to directly compare the quantities inside the max operations.

For the concentration bound term:

$$\underbrace{\left|\hat{V}_d^{C_a}(b) - \tilde{V}_{\text{VOWSS},d}(b)\right|}_{\text{POWSS Concentration bound}} \leq \left| \max_{a \in VOO(A,C_a)} \left\{ R(b,a) + \gamma\, \mathbb{E}[\hat{V}_{d+1}^{C_a}(bao)|b] \right\} \right. \tag{C.30}$$

$$\left. - \max_{a \in VOO(A,C_a)} \left\{ \frac{\sum_{i=1}^{C_s} w_{d,i}(r_{d,i} + \gamma \hat{V}_{d+1}^{C_a}(bao_i))}{\sum_{i=1}^{C_s} w_{d,i}} \right\} \right| \tag{C.31}$$

$$\leq \left| R(b,a^*) + \gamma\, \mathbb{E}[\hat{V}_{d+1}^{C_a}(ba^*o)|b] - \frac{\sum_{i=1}^{C_s} w_{d,i}(r_{d,i} + \gamma \hat{V}_{d+1}^{C_a}(ba^*o_i))}{\sum_{i=1}^{C_s} w_{d,i}} \right|.$$

$$(a^* = \arg\max_{VOO} \hat{Q}_d(b,a))$$

As a small note, picking the maximizing action with respect to $\hat{Q}_d$ does not guarantee that we have bounded the term in this case. We also need to consider the case of picking the maximizing action with respect to $\tilde{Q}_{\text{VOWSS},d}$, since $\tilde{Q}_{\text{VOWSS},d}$ could achieve larger values than $\hat{Q}_d$ due to sparse sampling of states. However, our overall result does not change, since choosing the other maximizing action will only result in a change in sign within the absolute value, which we can still bound with the recursive bound. Thus, in the following calculations whenever we pick the reference action for difference in quantities that do not have strict magnitude hierarchy, we do not consider this possibility as the terms can still be bounded with their respective recursive bounds by choosing an appropriate reference action.

Decomposing the quantity into the reward difference and the next-step value difference:

$$\underbrace{\left|\hat{V}_d^{C_a}(b) - \tilde{V}_{\text{VOWSS},d}(b)\right|}_{\text{POWSS Concentration bound}} \leq \underbrace{\left|R(b,a^*) - \frac{\sum_{i=1}^{C_s} w_{d,i} r_{d,i}}{\sum_{i=1}^{C_s} w_{d,i}}\right|}_{\text{Reward difference}} + \gamma \underbrace{\left|\mathbb{E}[\hat{V}_{d+1}^{C_a}(ba^*o)|b] - \frac{\sum_{i=1}^{C_s} w_{d,i} \hat{V}_{d+1}^{C_a}(ba^*o_i)}{\sum_{i=1}^{C_s} w_{d,i}}\right|}_{\text{Value difference}}. \tag{C.32}$$

For the reward difference, we can crudely upper bound it by the POWSS Q-value estimate concentration bound in Theorem 2 of [94], since this is effectively the same structure as the leaf node estimate. In the POWSS concentration bound in Theorem 2, the difference has an upper bound of $\lambda/(1-\gamma)$, which we will define as ϵ, and we denote the corresponding probability δ as p. This bounds the quantity by $(\epsilon + 2p \cdot V_{\max})$ using the expectation version of the POWSS concentration bound.

$$\underbrace{\left|R(b,a^*) - \frac{\sum_{i=1}^{C_s} w_{d,i} r_{d,i}}{\sum_{i=1}^{C_s} w_{d,i}}\right|}_{\text{Reward difference}} \leq \epsilon + 2p \cdot V_{\max}. \tag{C.33}$$

We could instead use Lemma 1 of [94], the leaf node estimate concentration bound, but we use the more general Theorem 2. This allows us to consistently use the same theorem throughout this analysis and effectively combine terms. Similarly, while the worst case bound is $2R_{\max}$ since we are taking the difference of two reward functions, we crudely upper bound that with $2V_{\max}$ for algebraic convenience of combining it with the value difference term.

$$\underbrace{\left|\mathbb{E}[\hat{V}_{d+1}^{C_a}(ba^*o)|b] - \frac{\sum_{i=1}^{C_s} w_{d,i} \hat{V}_{d+1}^{C_a}(ba^*o_i)}{\sum_{i=1}^{C_s} w_{d,i}}\right|}_{\text{Value difference}} \leq \epsilon + 2p \cdot V_{\max}. \tag{C.34}$$

On the other hand, for the value difference, the POWSS concentration bound also turns out to be an upper bound, but with more sophisticated calculations. During this part of the proof, we will refer heavily back to the continued proof of Lemma 2 in the Appendix C of [94] and give a general overview of how the steps apply here.

Step 2-i: Value Difference While Lemma 2 in [94] is calculated with respect to the theoretically optimal value function V^*_{d+1}, the calculation steps themselves and the theorems and lemmas used there can apply exactly the same way for $\hat{V}^{C_a}_{d+1}$. We will briefly illustrate how the steps are parallel to the proof of Lemma 2 in the Appendix C of [94].

The value difference corresponds to the difference between the expected value of $\hat{V}^{C_a}_{d+1}$ and the self-normalized importance sampling estimator of the expected value. This value difference specifically corresponds to the first two error terms in the continued proof of Lemma 2 in the Appendix C of [94]. The decomposition looks like the following:

$$\left|\mathbb{E}[\hat{V}^{C_a}_{d+1}(ba^*o)|b] - \frac{\sum_{i=1}^{C_s} w_{d,i}\hat{V}^{C_a}_{d+1}(ba^*o_i)}{\sum_{i=1}^{C_s} w_{d,i}}\right| \tag{C.35}$$

$$\leq \underbrace{\left|\mathbb{E}[\hat{V}^{C_a}_{d+1}(ba^*o)|b] - \frac{\sum_{i=1}^{C_s} w_{d,i}\hat{\mathbf{V}}^{C_a}_{d+1}(s_{d,i},b,a^*)}{\sum_{i=1}^{C_s} w_{d,i}}\right|}_{\text{Importance sampling error}} + \underbrace{\left|\frac{\sum_{i=1}^{C_s} w_{d,i}(\hat{\mathbf{V}}^{C_a}_{d+1}(s_{d,i},b,a^*) - \hat{V}^{C_a}_{d+1}(ba^*o_i))}{\sum_{i=1}^{C_s} w_{d,i}}\right|}_{\text{MC next-step integral approximation error}}. \tag{C.36}$$

The next-step marginal integral $\hat{\mathbf{V}}^{C_a}_{d+1}(s_{d,i},b,a^*)$ is defined as

$$\hat{\mathbf{V}}^{C_a}_{d+1}(s_{d,i},b,a^*) \equiv \int_S \int_O \hat{V}^{C_a}_{d+1}(ba^*o)\mathcal{Z}(o|a^*,s_{d+1})\mathcal{T}(s_{d+1}|s_{d,i},a^*)ds_{d+1}do. \tag{C.37}$$

With this analogous definition, the self-normalized estimator identities in [94] can be exactly applied to this setting once again, now instead for the function $\hat{V}^{C_a}_d$ because the algebraic steps taken in the proof should hold for our V function estimator as well. Intuitively, the next-step marginal integral is defined to be the marginal random variable for $\hat{V}^{C_a}_d$, instead of the optimal V function as it is done in the works of [94].

First, following [94], we define the following notation for products of transition and observation densities:

$$\mathcal{T}^i_{1:d} \equiv \prod_{n=1}^{d} \mathcal{T}(s_{n,i}|s_{n-1,i},a_n), \tag{C.38}$$

$$\mathcal{Z}^{i,j}_{1:d} \equiv \prod_{n=1}^{d} \mathcal{Z}(o_{n,j}|a_n,s_{n,i}). \tag{C.39}$$

Specifically, i denotes the index of the state sample, and j denotes the index of the observation sample. Absence of any of the indices i or j means that the state trajectory $\{s_n\}$ or the observation history $\{o_n\}$ appear as regular variables, mostly for the purposes of integration.

Step 2-ii: Importance Sampling Error For the importance sampling error, note that for a belief b at depth d,

$$\begin{aligned}\mathbb{E}[\hat{V}_{d+1}^{C_a}(ba^*o)|b] &= \int_S\int_S\int_O \hat{V}_{d+1}^{C_a}(ba^*o)(\mathcal{Z}_{d+1})(\mathcal{T}_{d,d+1})b\cdot ds_{d:d+1}do &\text{(C.40)}\\ &= \int_S \hat{\mathbf{V}}_{d+1}^{C_a}(s_{d,i},b,a^*)b\cdot ds_d &\text{(C.41)}\\ &= \frac{\int_{S^d}\hat{\mathbf{V}}_{d+1}^{C_a}(s_{d,i},b,a^*)(\mathcal{Z}_{1:d})(\mathcal{T}_{1:d})b_0 ds_{0:d}}{\int_{S^d}(\mathcal{Z}_{1:d})(\mathcal{T}_{1:d})b_0 ds_{0:d}}. &\text{(C.42)}\end{aligned}$$

Consequently, the weighted average of the next-step marginal integral $\hat{\mathbf{V}}_{d+1}^{C_a}(s_{d,i},b,a^*)$ is a self-normalized importance sampling estimator of $\hat{V}_{d+1}^{C_a}(ba^*o)$ given b, and we can apply the augmented self-normalized estimator concentration bound in the same way as [94] to get the concentration bound of $\lambda/3$.

Step 2-iii: Monte Carlo Next-Step Integral Approximation Error For the MC next-step integral approximation error, generating estimates of $\hat{V}_{d+1}^{C_a}(ba^*o_i)$ for a given tuple $(s_{d,i},b,a^*)$ also results in an unbiased Monte Carlo estimator of the next-step marginal integral, and the following difference is mean zero conditioned on $(s_{d,i},b,a^*)$:

$$\Delta_{d+1}(s_{d,i},b,a^*) \equiv \hat{\mathbf{V}}_{d+1}^{C_a}(s_{d,i},b,a^*) - \hat{V}_{d+1}^{C_a}(ba^*o_i). \tag{C.43}$$

Thus, the same calculation steps hold. Note that in the works of [94], the MC next-step integral approximation error is further crudely bound by $\frac{2}{3\gamma}\lambda$, when the actual bounds also hold for $\frac{2}{3}\lambda$, for the convenience of being able to combine the γ multiplied terms in the main proof of Lemma 2. Thus, we can bound the MC next-step integral approximation error with the stricter bound $\frac{2}{3}\lambda$.

In our work, we used the variable ϵ to denote the POWSS concentration bound, which corresponds to $\epsilon = \lambda/(1-\gamma) \geq \lambda$. Since the sum of importance sampling error and MC next-step integral approximation error are bounded by $(1/3+2/3)\lambda = \lambda$, this can also further be crudely bounded by the POWSS concentration inequality with the upper bound ϵ.

We have now obtained bounds for the value difference term using the expectation version of the POWSS concentration inequality where the extreme probability event is once again bounded with the term $2p\cdot V_{\max}$:

$$\left|\mathbb{E}[\hat{V}_{d+1}^{C_a}(ba^*o)|b] - \frac{\sum_{i=1}^{C_s} w_{d,i}\hat{V}_{d+1}^{C_a}(ba^*o_i)}{\sum_{i=1}^{C_s} w_{d,i}}\right| \leq \epsilon + 2p\cdot V_{\max}. \tag{C.44}$$

Finally, we can bound the POWSS Concentration bound term:

$$\underbrace{\left|\hat{V}_d^{C_a}(b) - \tilde{V}_{\text{VOWSS},d}(b)\right|}_{\text{POWSS Concentration bound}} \leq (\epsilon + 2p\cdot V_{\max}) + \gamma(\epsilon + 2p\cdot V_{\max}) = (1+\gamma)(\epsilon + 2p\cdot V_{\max}). \tag{C.45}$$

For the POWSS Recursive bound term, we simply apply the inductive hypothesis for step $d+1$:

$$\underbrace{\left|\tilde{V}_{\text{VOWSS},d}(b) - \hat{V}_{\text{VOWSS},d}(\bar{b})\right|}_{\text{POWSS Recursive bound}} \leq \left| \max_{a \in VOO(A,C_a)} \left\{ \frac{\sum_{i=1}^{C_s} w_{d,i}(r_{d,i} + \gamma \hat{V}_{d+1}^{C_a}(bao_i))}{\sum_{i=1}^{C_s} w_{d,i}} \right\} \right. \tag{C.46}$$

$$\left. - \max_{a \in VOO(A,C_a)} \left\{ \frac{\sum_{i=1}^{C_s} w_{d,i}(r_{d,i} + \gamma \hat{V}_{\text{VOWSS},d+1}(\overline{bao_i}))}{\sum_{i=1}^{C_s} w_{d,i}} \right\} \right| \tag{C.47}$$

$$\leq \left| \gamma \frac{\sum_{i=1}^{C_s} w_{d,i}(\hat{V}_{d+1}^{C_a}(b\tilde{a}o_i) - \hat{V}_{\text{VOWSS},d+1}(\overline{b\tilde{a}o_i}))}{\sum_{i=1}^{C_s} w_{d,i}} \right| \qquad (\tilde{a} = \arg\max\nolimits_{VOO} \tilde{Q}_{VOWSS,d}(b,a))$$

$$\leq \gamma \frac{\sum_{i=1}^{C_s} w_{d,i} \left| \hat{V}_{d+1}^{C_a}(b\tilde{a}o_i) - \hat{V}_{\text{VOWSS},d+1}(\overline{b\tilde{a}o_i}) \right|}{\sum_{i=1}^{C_s} w_{d,i}} \tag{C.48}$$

$$\leq \gamma \cdot \alpha_{C_s}(d+1). \tag{C.49}$$

Putting the POWSS components together, we prove the POWSS bound by induction:

$$\left|\hat{V}_d^{C_a}(b) - \hat{V}_{\text{VOWSS},d}(\bar{b})\right| \leq \underbrace{\left|\hat{V}_d^{C_a}(b) - \tilde{V}_{\text{VOWSS},d}(b)\right|}_{\text{POWSS Concentration bound}} + \underbrace{\left|\tilde{V}_{\text{VOWSS},d}(b) - \hat{V}_{\text{VOWSS},d}(\bar{b})\right|}_{\text{POWSS Recursive bound}} \tag{C.50}$$

$$\leq (1+\gamma)(\epsilon + 2p \cdot V_{\max}) + \gamma \cdot \alpha_{C_s}(d+1) = \alpha_{C_s}(d). \tag{C.51}$$

□

Proof of Theorem C.1. Finally, we prove the main theorem by combining the two terms:

$$\left|V_d^{\star}(b) - \hat{V}_{\text{VOWSS},d}(\bar{b})\right| \leq \underbrace{\left|V_d^{\star}(b) - \hat{V}_d^{C_a}(b)\right|}_{\text{VOO bound}} + \underbrace{\left|\hat{V}_d^{C_a}(b) - \hat{V}_{\text{VOWSS},d}(b)\right|}_{\text{POWSS bound}} \tag{C.52}$$

$$\leq \eta_{C_a}(d) + \alpha_{C_s}(d) \leq \eta_{C_a} + \alpha_{C_s}. \tag{C.53}$$

□

Here, the $\eta_{C_a}(d)$ corresponds to the VOO bound in [94]. Thus, we can always find a corresponding C_a for an arbitrary decreasing sequence of $\eta_{C_a}(d)$ which satisfies the properties in our lemma, and this sequence can be made closer to 0 for each depth by choosing bigger C_a values.

On the other hand, $\alpha_{C_s}(d)$ is comprised of POWSS bound components in [94], which decreases in both p and ϵ with more samples C_s. Thus, it can also be made to decrease to 0 as we increase C_s.

C.3 Experiment Hyperparameters

Hyperparameters were taken from the references if they were given, and otherwise the set of hyperparmaters for a system was obtained by first training POMCPOW, and then training VOMCPOW and BOMCP centered around POMCPOW hyperparameters as initial estimates. This usually augmented the performances of VOMCPOW and BOMCP compared to directly borrowing the POMCPOW hyperparameters. For BOMCP, buffer and k were fixed at 100 and 5, respectively, as it was done in the original paper by [124]. Table C.1 shows the final hyperparameters used in the experiments. Specifically for the lunar lander problem, we have chosen the dynamics time step $dt = 0.4$, which gave us the most consistent performance of BOMCP with 100 queries over 1000 iterations that is in line with the results of [124].

We give a brief intuitive explanation for each of the parameters utilized. c is the critical factor or the Upper Confidence Bound exploration parameter, which governs how much we should explore among the action samples we have collected. Since VPW needs to balance local and global search, on top of estimating relatively faithful Q-values, it usually worked better to set the c on the equal magnitude/lower than the c value for POMCPOW, usually around 10-100. k_a is the constant factor of the action widening parameter, which sets the overall width of the action widening. α_a is the exponential factor of the action widening parameter, which determines how adaptive we want to progressively widen the action width. k_o, α_o are the state/observation widening parameters, which function similarly to the action widening parameters. ω is the VOO exploration probability, which governs how much we should explore in the action space. Since in global search, we need to see enough samples to explore the action space sufficiently, it usually worked better to set ω to be relatively high around 0.7-0.9. Lastly, Σ is the Gaussian covariance matrix for VOO rejection sampling.

The only hyperparameter that was manually picked is the Gaussian covariance matrix for VOO rejection sampling. When picking this covariance matrix, we usually found that it was most effective to use a diagonal matrix with entries that are around 10 to 20 times less than the maximum action space bounds for each dimension. In theory, the diagonal covariance matrix can also be fitted via CEM, but we chose to handpick these values after some inspection in order to reduce the number of hyperparameters that needed to be fit.

To limit the number of rejection sampling iterations, we set the maximum number of rejection sampling iterations to 20 and automatically choose the closest action we have sampled when we reach 20 iterations. We also manually set automatic sample acceptance regions that was usually on the order of a tenth of the sampling radius, which we found was not strictly necessary in conjunction with the sampling iteration limit.

The open source code is available at github.com/michaelhlim/VOOTreeSearch.jl.

	c	k_a	α_a	k_o	α_o	C_s	C_a	ω	$\Sigma = \text{diag}(\sigma^2)$	l	λ	γ_a
LQG (Depth 3)												
POMCPOW	65.0	30.0	$\frac{1}{2.5}$	30.0	$\frac{1}{4}$							
VOMCPOW	60.0	25.0	$\frac{1}{5.5}$	25.0	$\frac{1}{2.5}$			0.8	[0.5, 0.5]			
BOMCP	135.0	30.0	$\frac{1}{4}$	20.0	$\frac{1}{4}$					log(15)	0.4	
VOWSS						10	200	0.8	[0.5, 0.5]			0.4
VDP Tag (Depth 10)												
POMCPOW	110.0	30.0	$\frac{1}{30}$	5.0	$\frac{1}{100}$							
VOMCPOW	85.0	30.0	$\frac{1}{30}$	2.5	$\frac{1}{100}$			0.7	[0.1]			
Lunar Lander (Depth 250)												
POMCPOW	10.0	3.0	$\frac{1}{4}$	2.0	$\frac{1}{10}$							
VOMCPOW	30.0	4.0	$\frac{1}{4}$	1.5	$\frac{1}{5}$			0.9	[0.2, 0.5, 0.05]			
BOMCP	10.0	3.0	$\frac{1}{4}$	2.0	$\frac{1}{10}$					log(15)	0.5	

Table C.1: **Summary of hyperparameters used in experiments for POMCPOW, VOMCPOW, BOMCP, and VOWSS.**

C.4 Voronoi Optimistic Sparse Sampling (VOSS) for Stochastic MDPs

Voronoi Optimistic Sparse Sampling (VOSS) is an application of VPW to the sparse sampling algorithm [92] to tackle the stochastic MDP case. Like VOWSS, it can be defined by an EstimateQ function that estimates the Q-value with next-step state samples. Since VOSS does not have observation uncertainty, it only needs to take the arithmetic average instead of the observation likelihood weighted average. The EstimateQ function that takes in a state s instead of a belief particle set $\bar{b}$ is outlined in Algorithm C.1. Note that this is analogous definition to the Q-function estimation algorithm in [92]. The EstimateV function should function similarly, except working with s instead of $\bar{b}$.

Algorithm C.1 Value estimation algorithm for VOSS

```
Algorithm: EstimateQ$(s, a, d)$
Input: State $s$, action $a$, depth $d$.
Output: A scalar $\hat{Q}^\star_d(s,a)$ that is an estimate of $Q^\star_d(s,a)$.
1: For $i = 1, \cdots, C_s$, generate $s'_i, r = G(s,a)$.
2: Return the $Q$-value estimate:
   $$\hat{Q}^\star_d(s,a) = r + \gamma \frac{1}{C_s} \sum_{i=1}^{C_s} \text{EstimateV}(s'_i, d+1).$$
```

The conditions required for the proof is the union of regularity conditions required for sparse sampling and VOO. The sparse sampling conditions are the subset of POWSS conditions. Namely, we only require the conditions (i), (iii), (iv), only for the state and action spaces. Since in this proof we simply use the Chernoff bound for the sparse sampling type bound, we just need to make sure that $p \geq \exp(-\epsilon^2 C_s/(2V_{\max})^2)$ holds when picking C_s.

Theorem C.2 (VOSS Inequality)**.** *Suppose we choose the action sampling width C_a and state sampling width C_s such that under the union of regularity conditions specified by [92] and [94], the intermediate sparse sampling bounds and VOO bounds in Lemma 2A are satisfied at every depth of the tree. Then, the following bounds for the VOSS estimator $\hat{V}_{\text{VOSS},d}(s)$ hold for all $d \in [0, D-1]$ in expectation:*

$$\left| V^\star_d(s) - \hat{V}_{\text{VOSS},d}(s) \right| \leq \eta_{C_a} + \alpha_{C_s}. \tag{C.54}$$

Similar to proving Theorem C.1, to prove Theorem C.2, we first prove an intermediate lemma which will allow us to obtain the bound through triangle inequality. The proof is easier than that of Theorem C.1, since we do not explicitly need to deal with the observation uncertainty. We introduce and prove the following lemma first. All of the following calculations are done in expectation.

Lemma 2A (VOSS Intermediate Inequality)**.** *Suppose with our notation, the sparse sampling estimators at all depths d are within ϵ of their mean values with probability $1-p$, and the VOO agents have regret bounds of $\mathcal{R}_{C_a}$. The following inequalities hold for all $d \in [0, D-1]$ in expectation:*

$$\left|V_d^\star(s) - \hat{V}_d^{C_a}(s)\right| \leq \eta_{C_a}(d), \tag{C.55}$$

$$\left|\hat{V}_d^{C_a}(s) - \hat{V}_{\text{VOSS},d}(s)\right| \leq \alpha_{C_s}(d). \tag{C.56}$$

$\eta_{C_a}(d)$ and $\alpha_{C_s}(d)$ are sequences that satisfy the following properties:

$$\eta_{C_a}(d) \geq \gamma \cdot \eta_{C_a}(d+1) + \mathcal{R}_{C_a},\ \eta_{C_a}(D) = 0, \tag{C.57}$$

$$\eta_{C_a} \equiv \max_{d=0,\cdots,D-1} \eta_{C_a}(d) < +\infty, \tag{C.58}$$

$$\alpha_{C_s}(d) \equiv \gamma(\alpha_{C_s}(d+1) + \epsilon + 2p \cdot V_{\max}),\ \alpha_{C_s}(D-1) = \epsilon + 2p \cdot V_{\max}, \tag{C.59}$$

$$\alpha_{C_s} \equiv \max_{d=0,\cdots,D-1} \alpha_{C_s}(d) < +\infty. \tag{C.60}$$

Proof. This proof proceeds through induction by assuming that the bounds hold for all depths from $d+1$ to $D-1$, and then proving they also hold for depth d (induction hypothesis is omitted as the bounds trivially hold as per the definitions of VOOT and sparse sampling). We divide the main inequality into VOO bound and sparse sampling bound (SS bound) by introducing an intermediate term $\hat{V}_d^{C_a}(s)$:

$$\left|V_d^\star(s) - \hat{V}_{\text{VOSS},d}(s)\right| \leq \underbrace{\left|V_d^\star(s) - \hat{V}_d^{C_a}(s)\right|}_{\text{VOO bound}} + \underbrace{\left|\hat{V}_d^{C_a}(s) - \hat{V}_{\text{VOSS},d}(s)\right|}_{\text{SS bound}}. \tag{C.61}$$

We now have two main layers of inequality, caused by the stochastic nature of VOO action selection and the uncertainty in state transition. We will first analyze the VOO bound, then the SS bound.

Step 1: VOO Bound The VOO bound can be further decomposed into the following terms as [94] do in their work:

$$\left|V_d^\star(s) - \hat{V}_d^{C_a}(s)\right| \leq \underbrace{\left|V_d^\star(s) - \hat{V}_d(s)\right|}_{\text{VOO Recursive bound}} + \underbrace{\left|\hat{V}_d(s) - \hat{V}_d^{C_a}(s)\right|}_{\text{VOO Regret bound}}. \tag{C.62}$$

This step is very close to our previous procedure in VOWSS. Here, the intermediate random variables are defined in the following manner:

$$V_d^\star(s) \equiv \max_{a\in A} Q_d^\star(s,a) = \max_{a\in A} \left\{R(s,a) + \gamma\, \mathbb{E}_{s'\sim T(s,a)}[V_{d+1}^\star(s')]\right\}, \tag{C.63}$$

$$\hat{V}_d(s) \equiv \max_{a\in A} \hat{Q}_d(s,a) = \max_{a\in A} \left\{R(s,a) + \gamma\, \mathbb{E}_{s'\sim T(s,a)}[\hat{V}_{d+1}^{C_a}(s')]\right\}, \tag{C.64}$$

$$\hat{V}_d^{C_a}(s) \equiv \max_{a\in VOO(A,C_a)} \hat{Q}_d(s,a) = \max_{a\in VOO(A,C_a)} \left\{R(s,a) + \gamma\, \mathbb{E}_{s'\sim T(s,a)}[\hat{V}_{d+1}^{C_a}(s')]\right\}. \tag{C.65}$$

Repeating the procedures in Lemma 1A, we closely follow the calculations from Lemma 5 in [94]:

$$\begin{aligned}
\underbrace{\left|V_d^\star(s) - \hat{V}_d(s)\right|}_{\text{VOO Recursive bound}} &= \left| \max_{a\in A} \left\{ R(s,a) + \gamma\, \mathbb{E}_{s'\sim T(s,a)}[V_{d+1}^\star(s')] \right\} \right. && \text{(C.66)} \\
&\quad \left. - \max_{a\in A} \left\{ R(s,a) + \gamma\, \mathbb{E}_{s'\sim T(s,a)}[\hat{V}_{d+1}^{C_a}(s')] \right\} \right| && \text{(C.67)} \\
&\leq \left| R(s,a^\star) + \gamma\, \mathbb{E}_{s'\sim T(s,a^\star)}[V_{d+1}^\star(s')] - R(s,a^\star) - \gamma\, \mathbb{E}_{s'\sim T(s,a^\star)}[\hat{V}_{d+1}^{C_a}(s')] \right| \\
&&& (a^\star = \arg\max Q_d^\star(s,a)) \\
&\leq \gamma\, \mathbb{E}_{s'\sim T(s,a^\star)} \left| V_{d+1}^\star(s') - \hat{V}_{d+1}^{C_a}(s') \right| && \text{(C.68)} \\
&\leq \gamma \cdot \eta_{C_a}(d+1) && \text{(C.69)} \\
\underbrace{\left|\hat{V}_d(s) - \hat{V}_d^{C_a}(s)\right|}_{\text{VOO Regret bound}} &\leq \mathcal{R}_{C_a}. && \text{(C.70)}
\end{aligned}$$

Thus, with our choice of C_a that is designed to satisfy the recurrence relation, we obtain the bound:

$$\begin{aligned}
\left|V_d^\star(s) - \hat{V}_d^{C_a}(s)\right| &\leq \underbrace{\left|V_d^\star(s) - \hat{V}_d(s)\right|}_{\text{VOO Recursive bound}} + \underbrace{\left|\hat{V}_d(s) - \hat{V}_d^{C_a}(s)\right|}_{\text{VOO Regret bound}} && \text{(C.71)} \\
&\leq \gamma \cdot \eta_{C_a}(d+1) + \mathcal{R}_{C_a} \leq \eta_{C_a}(d). && \text{(C.72)}
\end{aligned}$$

Step 2: Sparse Sampling Bound Similar to Lemma 1A, the SS bound can also be further decomposed into the following terms:

$$\left|\hat{V}_d^{C_a}(s) - \hat{V}_{\text{VOSS},d}(s)\right| \leq \underbrace{\left|\hat{V}_d^{C_a}(s) - \tilde{V}_{\text{VOSS},d}(s)\right|}_{\text{SS Concentration bound}} + \underbrace{\left|\tilde{V}_{\text{VOSS},d}(s) - \hat{V}_{\text{VOSS},d}(s)\right|}_{\text{SS Recursive bound}}. \tag{C.73}$$

Here, the extra intermediate random variable and the VOSS estimator are defined in the following manner:

$$\tilde{V}_{\text{VOSS},d}(s) \equiv \max_{a\in VOO(A,C_a)} \tilde{Q}_{\text{VOSS},d}(s,a) = \max_{a\in VOO(A,C_a)} \left\{ R(s,a) + \gamma \frac{1}{C_s} \sum_{i=1}^{C_s} \hat{V}_{d+1}^{C_a}(s_i') \right\}, \tag{C.74}$$

$$\hat{V}_{\text{VOSS},d}(s) \equiv \max_{a\in VOO(A,C_a)} \hat{Q}_{\text{VOSS},d}(s,a) = \max_{a\in VOO(A,C_a)} \left\{ R(s,a) + \gamma \frac{1}{C_s} \sum_{i=1}^{C_s} \hat{V}_{\text{VOSS},d+1}(s_i') \right\}. \tag{C.75}$$

We now apply the sparse sampling bound as well as the recursive bound in order to bound the SS bound components. In our case, since our intermediate sparse sampling term $\tilde{V}_{\text{VOSS},d}(s)$ is merely swapping out the expectation of $\hat{V}_{d+1}^{C_a}$ with a sample average under the appropriate sampling density, this turns out to be simply the Chernoff bound. We transform the sparse sampling concentration bound to an expected value bound by using the fact that the difference of V functions/estimators is bounded above by $2V_{\max}$, setting the concentration bound to be ϵ as per the Lemma statement and assigning the worst case result with probability p. Once again, we compare the two quantities by picking a reference action to directly compare the quantities inside the max operations:

$$\begin{aligned}
\underbrace{\left|\hat{V}_d^{C_a}(s) - \tilde{V}_{\text{VOSS},d}(s)\right|}_{\text{SS Concentration bound}} &\leq \Bigg| \max_{a\in VOO(A,C_a)} \{R(s,a) + \gamma\, \mathbb{E}_{s'\sim T(s,a)}[\hat{V}_{d+1}^{C_a}(s')]\} \qquad (C.76)\\
&\quad - \max_{a\in VOO(A,C_a)} \{R(s,a) + \gamma \frac{1}{C_s}\sum_{i=1}^{C_s} \hat{V}_{d+1}^{C_a}(s_i')\}\Bigg| \\
&\leq \gamma \left| \mathbb{E}_{s'\sim T(s,a^*)}[\hat{V}_{d+1}^{C_a}(s')] - \frac{1}{C_s}\sum_{i=1}^{C_s} \hat{V}_{d+1}^{C_a}(s_i') \right| \\
&\qquad\qquad (a^* = \arg\max\nolimits_{VOO} \hat{Q}_d(s,a)) \\
&\leq \gamma\cdot((1-p)\epsilon + 2p\cdot V_{\max}) \leq \gamma\cdot(\epsilon + 2p\cdot V_{\max}). \qquad (C.77)
\end{aligned}$$

For the SS Recursive bound, we proceed with the similar recursive calculation as SS Recursive bound term done in Lemma 1A by using the inductive hypothesis for step $d+1$:

$$\begin{aligned}
\underbrace{\left|\tilde{V}_{\text{VOSS},d}(s) - \hat{V}_{\text{VOSS},d}(s)\right|}_{\text{SS Recursive bound}} &\leq \Bigg| \max_{a\in VOO(A,C_a)} \left\{R(s,a) + \gamma \frac{1}{C_s}\sum_{i=1}^{C_s} \hat{V}_{d+1}^{C_a}(s_i')\right\} \qquad (C.78)\\
&\quad - \max_{a\in VOO(A,C_a)} \left\{R(s,a) + \gamma \frac{1}{C_s}\sum_{i=1}^{C_s} \hat{V}_{\text{VOSS},d+1}(s_i')\right\}\Bigg| \\
&\leq \gamma \frac{1}{C_s}\sum_{i=1}^{C_s} \left|\hat{V}_{d+1}^{C_a}(s_i') - \hat{V}_{\text{VOSS},d+1}(s_i')\right| \\
&\qquad\qquad (\tilde{a} = \arg\max\nolimits_{VOO} \tilde{Q}_{\text{VOSS},d}(s,a)) \\
&\leq \gamma\cdot\alpha_{C_s}(d+1). \qquad (C.79)
\end{aligned}$$

Putting the sparse sampling components together by applying the recursive definition of

$\alpha_{C_s}(d)$, we obtain the recurring concentration inequality by induction:

$$\left|\hat{V}_d^{C_a}(s) - \hat{V}_{\text{VOSS},d}(s)\right| \leq \underbrace{\left|\hat{V}_d^{C_a}(s) - \tilde{V}_{\text{VOSS},d}(s)\right|}_{\text{SS Concentration bound}} + \underbrace{\left|\tilde{V}_{\text{VOSS},d}(s) - \hat{V}_{\text{VOSS},d}(s)\right|}_{\text{SS Recursive bound}} \tag{C.80}$$

$$\leq \gamma \cdot (\epsilon + 2p \cdot V_{\max}) + \gamma \cdot \alpha_{C_s}(d+1) = \alpha_{C_s}(d). \tag{C.81}$$

□

Proof of Theorem C.2. Finally, we prove the main theorem by combining the two terms:

$$\left|V_d^{\star}(s) - \hat{V}_{\text{VOSS},d}(s)\right| \leq \underbrace{\left|V_d^{\star}(s) - \hat{V}_d^{C_a}(s)\right|}_{\text{VOO bound}} + \underbrace{\left|\hat{V}_d^{C_a}(s) - \hat{V}_{\text{VOSS},d}(s)\right|}_{\text{SS bound}} \tag{C.82}$$

$$\leq \eta_{C_a}(d) + \alpha_{C_s}(d) \leq \eta_{C_a} + \alpha_{C_s}. \tag{C.83}$$

□

Here, the $\eta_{C_a}(d)$ corresponds to the VOO bound in [94]. Thus, we can always find a corresponding C_a for an arbitrary decreasing sequence of $\eta_{C_a}(d)$ which satisfies the properties in our lemma, and this sequence can be made closer to 0 for each depth by choosing bigger C_a values.

On the other hand, $\alpha_{C_s}(d)$ corresponds to the sparse sampling bound/Chernoff bound which decreases in both p and ϵ with more samples C_s. Thus, it can also be made to decrease to 0 as we increase C_s.

Appendix D

Details for Visual Tree Search

D.1 Experiment Details

For all methods, the episode length was 200 steps. If the agent reached the goal in under 200 steps, this was considered a success with a positive reward of 100 and a termination of the episode. Entrance into traps incurred a negative reward of -100 per entry. This is with the exception of DVRL, in which rewards were scaled to lie between -1 and 1 to ensure stability of the training. The DVRL rewards reported here are scaled by 100.

In addition to statistics such as the success rate, reward, and so on, we have provided values for the number of successful trials for the method. This is because certain seeds for DualSMC failed to produce any goal-reaching behavior at all during training. Since the provided statistics are only for episodes in which the goal was reached, those seeds of DualSMC were dropped entirely.

The prohibitive training time of PlaNet allowed for only 4 trials to be run with reasonable computational resources. Training for all methods was done on NVIDIA Tesla K80 GPUs.

Note that the "planning time" reported for DVRL is actually the time to run the forward pass of the DVRL policy, since the method does not perform planning. The number has been provided to display that aspect of the algorithm.

D.2 VTS Training Data Generation Details

The open source code for our implementations will be available at https://github.com/michaelhlim/VisualTreeSearch.

D.2.1 Floor Positioning

We trained the Z_θ, P_ϕ, and G_ξ models with random data batches. Z_θ and P_ϕ were provided with random batches of samples $(s_t, o_t, \{\hat{s}_{t,i}\})$, with $s_t = (x_t, y_t)$ being the agent's position. The $\{\hat{s}_{t,i}\}$ were generated from a normal distribution centered at s_t with standard deviation

0.01. G_ξ was provided with random batches of samples (s_t, o_t), where for each batch we sampled the s_t and generated each o_t from the corresponding s_t. The sampling procedure for the states was biased towards the wall states, since those states were more difficult to learn from. Specifically, for Z_θ and P_ϕ, in each batch a state would be chosen from a wall with 0.5 probability and from anywhere in the environment with 0.5 probability. For G_ξ, in each batch a state would be chosen from a wall with 0.5 probability, from a non-wall area with 0.375 probability, and from anywhere in the environment with 0.125 probability.

D.2.2 3D Light-Dark

To train the Z_θ and P_ϕ models, we employed the following procedure. Each training batch consisted of random batches of samples $(s_t, \theta_t, o_t, \{\hat{s}_{t,i}\})$, where $s_t = (x_t, y_t)$. The agent's location could be described by both its position s_t and its heading θ_t, though only s_t was unknown. In each batch, θ_t was discretely chosen from $[0, 2\pi)$ at intervals of $\pi/4$. This is because the agent's actions were in the form of discrete commands instantaneously changing its heading. If s_t was in the dark region, then the $\{\hat{s}_{t,i}\}$ were generated from a normal distribution centered at s_t with standard deviation 0.1, and 0.01 for the light region. The difference in standard deviation was to enable the Z_θ and P_ϕ models to cause better localization in the light region than in the dark region. The two models were trained on a schedule that gradually introduced noise in the image observations o_t over the training epochs. In contrast, G_ξ was trained simply with the random batches (s_t, θ_t, o_t) without the noise schedule, as if it were data collected by a randomly exploring agent. However, both sets of models could be trained either way in principle. For these conditional generative models, the concatenation of s_t and θ_t formed the conditional variable.

D.2.3 On-Policy Refining

Technically, we could employ additional on-policy learning to refine the networks to focus on parts of the problem that are relevant for a given task as done in training DPF-based policies [83]. However, we found that on-policy learning does not significantly improve performance in addition to regular training, since the planner often successfully and quickly reaches the goal and does not provide many new samples in areas where the models need improvement.

D.3 Hyperparameters and Computation Details

Table D.1 shows the hyperparameters used in training and filtering for VTS. Our models were trained on NVIDIA Tesla K80 GPUs. The hyperparameters for DualSMC training remained the same as the original work [183]. For 3D Light-Dark, we maintained a training pool of data to draw random batches of samples from instead of generating image observations at every training step.

Hyperparameters	Floor Positioning	3D Light-Dark
Training gradient steps	500,000	100,000
Training sample pool size	-	16,000
Learning rate	0.001	0.0003
Batch size	64	64
Filtering particles	100	100
Percent of particles proposed	0.3	0.3
Resampling frequency	3	3

Table D.1: **Summary of hyperparameters used in training and filtering for vision POMDP.**

Unlike DualSMC, VTS does not use an LSTM as one of the intermediate layers in Z_θ because Z_θ is trained offline from randomly collected data. We observed that this would result in particle de-localization, as the observation density could not propagate forward the localized information. In essence, the VTS solver would be quick to localize, but sometimes would need to take a detour in order to re-localize. Particle de-localization has been observed before in the literature and there are many potential methods to remedy this problem. We found an exponential decay of the number of particles proposed, as suggested by [183], to provide the best results and solve the issue. That is, for episode step number n and nominal particle proposal amount p, the number of particles proposed is $p * \gamma_d^n$. We set the decay rate γ_d to 0.9.

Table D.2 shows the hyperparameters used for PFT-DPW planner. While hyperparameters for tree search planners are often chosen with hyperparameter sweeps and/or optimization, such meta-optimization in our algorithm would mean having to run our algorithm on a GPU numerous times, which was practically very inefficient. Thus, the hyperparameters were manually chosen via a combination of inspection and prior experience. For both Floor Positioning and 3D Light-Dark, we found the rollout strategy that yielded the best results to be one in which we assume that the uncertainty in the current belief collapses in the next step. Essentially, we select a random particle from the belief and calculate its straight-line direction and distance to the goal. We use this distance to estimate the discounted reward associated with that particle. We add that with the weighted sum of rewards obtained when all other particles move in that same direction and distance.

Table D.3 shows some parameters for the Floor Positioning and 3D Light-Dark environments. The environment parameters for Floor Positioning were left unchanged from [183].

	n	c	k_a	α_a	k_o	α_o	m	H	γ
Floor Positioning	100	10.0	3.0	$\frac{1}{4}$	4.0	$\frac{1}{4}$	100	10	0.99
3D Light-Dark	100	10.0	3.0	$\frac{1}{4}$	4.0	$\frac{1}{4}$	100	10	0.99

Table D.2: **Summary of hyperparameters used in PFT-DPW for VTS.**

Across all baseline planning methods, the dimension of the action varied according to the problem in accordance with [183]; for the Floor Positioning problem, the action dimension was 2, representing a change in the 2D state. For the 3D Light-Dark problem, the action dimension was 1, representing a change in the agent's orientation. DVRL, however, was found to produce the best performance when the action was 2D for both problems.

Hyperparameters	Floor Positioning	3D Light-Dark
Observation size	4	$32 \times 32 \times 3$
Agent velocity	0.05	0.2
Noise amount (image)	-	40%
Occlusion amount (image)	-	15×15

Table D.3: **Summary of vision POMDP environment parameters.**

For the 3D Light-Dark problem, the Z_θ and P_ϕ modules were trained with a gradual noise schedule that was as follows: for the first 1/4 of the number of training epochs, the images in the training data had 0% noise. For the next 1/4 of the epochs, the images corresponding to states in the dark region had $final\ noise\ \% * 1/4$ noise. For the next 1/4 of the epochs, the images had $final\ noise\ \% * 1/2$ noise. For the final 1/4 of the epochs, the images had the full $final\ noise\ \%$. (As seen in Table D.3, $final\ noise\ \%$ was 40%.) This percentage corresponds to the percent of locations on the 32×32 image that were corrupted by the salt-and-pepper noise.

Finally, Table D.4 shows the neural network architecture details for the Z, P, G models we trained on different experimental domains. We note here that although VTS and DualSMC differ in the architectures of their Z_θ and P_ϕ models in that the networks for VTS have deeper structures than those for DualSMC, we did not see any noticeable increase in performance for DualSMC with the deeper architectures in a preliminary exploration.

Model		Layers	# Channels
Floor Positioning			
Z_θ		5-layer MLP	$256 \times 3, 128, 16$
		Concat: state	18
		3-layer MLP	$256 \times 2, 1$
P_ϕ		3-layer MLP	$256 \times 2, 64$
		Concat: $z(64) \sim \mathcal{N}(0, 1)$	128
		4-layer MLP	$256 \times 3, 2$
G_ξ	Encoder	5-layer MLP	256×4, 64
		1 layer to output μ, σ	64
	Decoder	5-layer MLP	256×4, 4
3D Light-Dark			
Z_θ		Conv2d, filter (3, 3), stride 1	num image channels
		Conv2d $\times$ 4, filter (3, 3), stride 2	32, 64, 128, 256
		Conv2d, filter (3, 3), stride 1	512
		3-layer MLP	1024, 512, 256
		Normalize output (*)	256
		3-layer MLP	256 $\times$ 3
		Concat: state, orientation	259
		4-layer MLP	128 $\times$ 3, 1
P_ϕ		Same as Z_θ up to (*)	256
		1-layer MLP	256
		Concat: $z(256) \sim \mathcal{N}(0, 1)$, orientation	513
		3-layer MLP	128 $\times$ 2, 2
G_ξ	Encoder	Same as Z_θ up to (*)	256
		Concat: state, orientation	259
		5-layer MLP	256 $\times$ 4, 128
		1 layer MLP to output μ, σ	128
	Decoder	Concat $z(128) \sim \mathcal{N}(0, 1)$, state, orientation	131
		5-layer MLP	256 $\times$ 5
		3-layer MLP	256, 512, 1024
		Conv2dTranspose $\times$ 3, filter (3, 3), stride 2	128, 64, 32
		Conv2d, filter (3, 3), stride 2	num image channels

Table D.4: **VTS Network details.**

D.4 Stanford 3D Light-Dark Environment Implementation Details

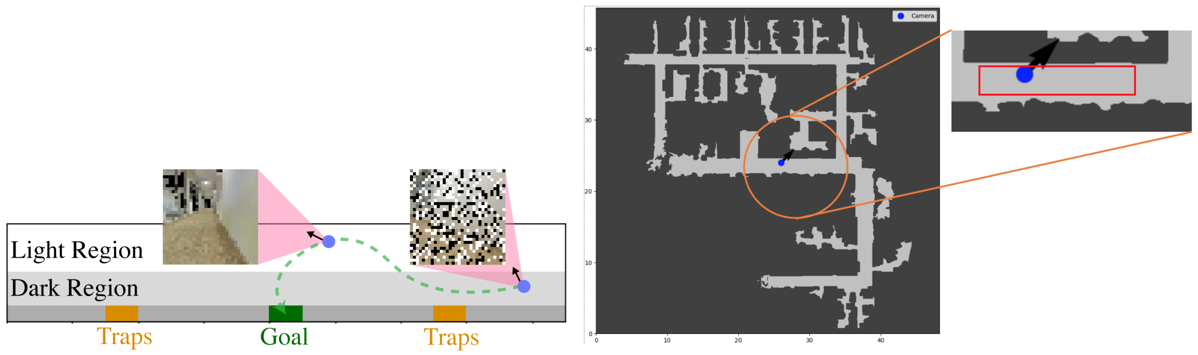


Figure D.1: **The overview diagram of the 3D Light-Dark problem** implemented using the Stanford dataset (left), and a top-down map of a floor in a building in the Stanford dataset that we use to recreate the 3D Light-Dark problem (right).

Fig. D.1 shows the overview diagram of the 3D Light-Dark problem implemented using the Stanford dataset (left), and a top-down map of a floor in a building in the Stanford dataset (right).

The environment diagram depicts the "trap" regions in orange, the goal region in green, and the agent in blue. The agent's location is given by (x, y, θ), where x, y are the unknown absolute position coordinates and θ is the known heading. The shaded part of the environment depicts the "dark" region, where the image observations received by the agent are corrupted by salt-and-pepper noise. The rest of the environment is in the "light", where the image observations are uncorrupted. The darkest gray region is a wall region that the agent cannot enter. The agent's initial state is randomly chosen from a strip in the dark region.

In the top-down map, the red rectangle denotes the part of the building that forms our environment (top right). We chose a subset of the Stanford dataset region in order to replicate the exact settings of the 3D Light-Dark problem in [183]. However, since our 3D Light-Dark implementation uses realistic images rendered with the Stanford dataset as opposed to using synthetic backgrounds generated with the DeepMind simulation platform, our problem is more realistic and challenging.

D.5 Baseline Planner Implementation Details

We compare the performance of VTS against two baselines in addition to DualSMC: Deep Variational Reinforcement Learning for POMDPs (DVRL), and Deep Planning Network (PlaNet).

We evaluated DVRL in both the Floor Positioning and 3D Light-Dark environments and PlaNet in the 3D Light-Dark environment. This was so that we could minimally modify the model architectures provided by the two methods; while DVRL architectures supported both vector and image observations, the PlaNet architecture only supported image observations. This was also the reason for running the PlaNet 3D Light-Dark experiments with 64×64 images while all other baselines and VTS used 32×32 images. In this way, the baselines were advantaged in our comparisons.

D.5.1 DVRL

Table D.5 shows the parameters modified when running DVRL, with all other parameters unchanged from the original work. For Floor Positioning, a sweep was performed over the encoder channel dimensions, presence of the BatchNorm, multiplier backprop length, and number of particles. For 3D Light-Dark, a sweep was performed over the number of environment processes, number of environment steps per gradient step, and multiplier backprop length to balance performance and computation time.

Since the DVRL implementation is designed to produce trajectories in the environment by running multiple environment processes in parallel, we performed both training and testing in this multiprocessing framework. For testing, we ran 2 environment processes in parallel and collected the 1000 testing episodes for Floor Positioning and 500 testing episodes for 3D Light-Dark from their combined results. Also, the DVRL policy produces actions for these parallel environments at once. Therefore, the "planning time" provided is the time for the policy to produce 2 actions.

D.5.2 PlaNet

Table D.6 shows the parameters modified when running PlaNet, with all other parameters unchanged from the original work. A sweep was performed over action repeat and learning rate. The occlusion amount was increased to 30 as compared to 15 for the other baselines to maintain consistency with the larger image size.

Since the action repeat for the PlaNet baseline is 2, the agent only needs to perform the planning, which is done via the Cross Entropy Method (CEM), every 2 steps. This is because the action resulting from the plan is applied twice. To reflect this aspect of the algorithm in the results, we divided the obtained planning time by 2 to produce the final numbers.

Hyperparameters	Floor Positioning	3D Light-Dark
Environment		
Observation size	4	32 × 32 × 3
Agent velocity	0.05	0.2
Noise amount (image)	–	40%
Occlusion amount (image)	–	15 × 15
Training		
Number of training frames	2.5e7	1.2e6
Number of env steps per gradient step	25	5
Multiplier backprop length	1	100
Number of particles	10	10
Number of environment processes	16	4
Model		
Encoder channel dimensions	$[64, 64]$	$[32, 64, 32]$
BatchNorm present	Yes	Yes
RNN latent state size (h)	256	256
Additional latent state size (z)	256	256
Action encoding size	64	128

Table D.5: **Summary of DVRL environment, training, and model hyperparameters.**

Hyperparameters	3D Light-Dark
Environment	
Observation size	64 × 64 × 3
Agent velocity	0.2
Action repeat	2
Noise amount (image)	40%
Occlusion amount (image)	30 × 30
Training	
Number of training steps	1e6
Number of training steps per data collection phase	100
Environment steps per data collection phase	200
Learning rate	1e-3

Table D.6: **Summary of PlaNet environment and training hyperparameters.**

Appendix E

Details for Sequence-Conditioned Transporter Networks

E.1 Covid-19 Details

Due to the ongoing COVID-19 pandemic, performing physical robotics experiments became a huge challenge. The resulting challenges include: restrictions in traveling to the robotic setup locations, restrictions in collecting demonstrations and performing experiments at the locations as per COVID-19 safety protocols, and difficulties and delays in sourcing materials to devise and build realistic yet tractable pick-and-place experiments. On that end, we continue to utilize the PyBullet simulation environment like Zeng et al. [199] and Seita et al. [154], and modify it to load multiple task modules at once. We hope to investigate the performance of SCTN on realistic experiment setups in future works.

E.2 Task Details

In *MultiRavens*, we propose 10 multi-task problems that are made up of 4 basic task modules. These task modules take inspiration from National Institute of Standards and Technology (NIST) Assembly Task Boards [54] as shown in Fig. E.1. In this section, we give further details of the task, including evaluation metrics and corresponding demonstrator policies. The visualizations of these tasks are shown in Fig. 10.2, and quick overview of unique task attributes of each task is given in Table 10.1. All problems have task modules anchored at the same coordinates each time they are generated for consistency.

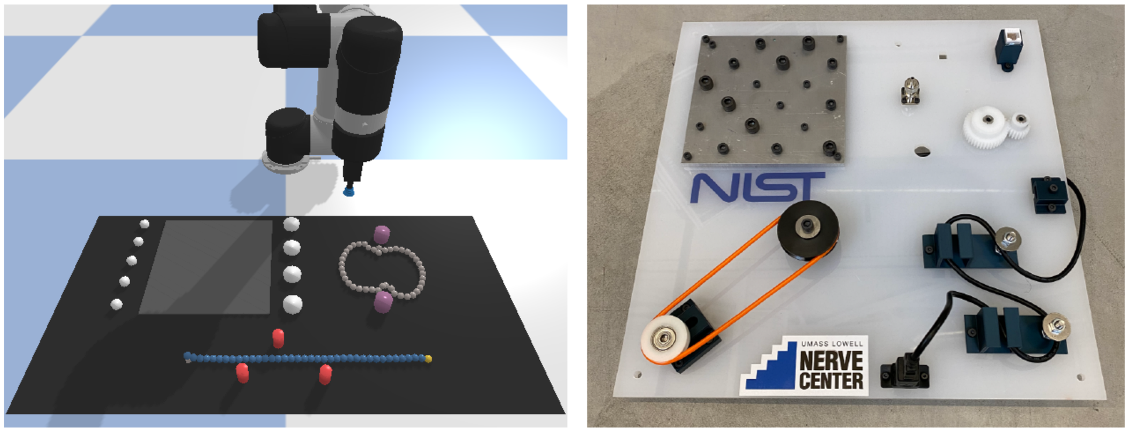


Figure E.1: **The new modularized tasks in MultiRavens**, *placing, chaining* and *routing* (left), take inspiration from the National Institute of Standards and Technology (NIST) Assembly Task Boards (right).

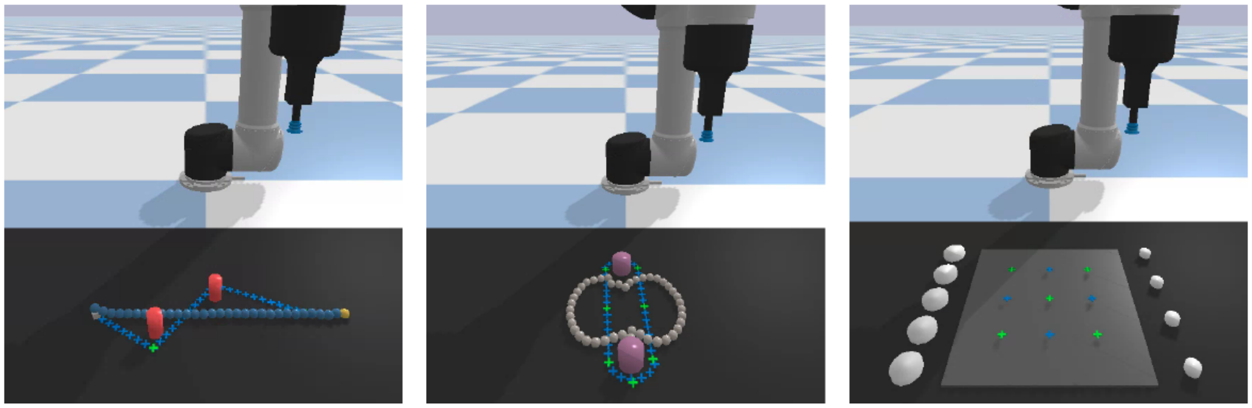


Figure E.2: **The explicitly visualized evaluation goal positions and/or object goal positions for each task in MultiRavens**: *routing* (left), *chaining* (middle) and *placing* (right). For *routing* and *chaining*, the green '+' corresponds to the evaluation goal positions for reward calculation, and the blue '+' to the bead goal positions to aid oracle planning. For *placing*, the green '+' corresponds to the goal positions for even disks, and the blue '+' for odd disks.

E.2.1 Placing

E.2.1.1 Overview

For the *placing* task, the agent is tasked to place the 9 disks at their goal positions on top of the dark gray square plate. The 9 disks are initialized such that 5 of the "even" index disks are placed on the left of the plate, and 4 "odd" disks placed on the right. The even and odd disks can either be big or small for a given parity, which gives us the total of 4 possible task goal configurations.

E.2.1.2 Metrics

The evaluation metric checks whether the disks are placed in their designated zone (even or odd) and whether they are placed close enough to the 3×3 grid of goal positions, where visualizations of green '+' corresponds to goal position for even disks and blue '+' for odd disks as shown in Fig. E.2. Delta reward of 1/9 is given whenever a disk is placed successfully. The metric is robust to stacking disks, which means that it does not award additional delta reward if there is already a disk at the goal position and another disk is stacked on top. This ensures that the task is complete only when all the disks are in their designated goal configurations with high enough accuracy.

E.2.1.3 Demonstrator Policy

The even and odd disks have randomized oracle goal positions, which means that despite there being only 4 possible task goal configurations, the demonstration policy roll outs are highly varied and randomized. The demonstrator randomly picks the disks to be placed at their designated oracle goal positions, which ensures that the Transporter agents cannot simply copy or memorize the sequences of moves. The demonstrator often needs exactly 9 steps to successfully execute the demonstration.

E.2.2 Chaining

E.2.2.1 Overview

For the *chaining* task, the agent is tasked to wrap the chain of beads around the two fixed pegs. The peg distances and radii are randomly selected with 4 possible combinations, while the orientation of the task is set randomly with angle $\theta \in [-\pi/4, \pi/4]$. The closed-loop chain is made up of connected beads in a manner similar to the implementations in the Transporter works [154, 199]. Depending on the initial configuration, the bead chain length is variable, usually in the range of around 30-40 beads.

E.2.2.2 Metrics

The evaluation metric checks whether the chain is wrapped around the pegs, with each successful wrap resulting in a delta reward of 1/2. The evaluation goal poses check for whether there are any beads close to the points, shown as green '+' in Fig. E.2. The evaluation poses will only be fully satisfied if the chain is wrapped around both pegs without any twisting.

E.2.2.3 Demonstrator Policy

The demonstrator policy attempts to place the beads at each of their pre-specified bead-level goal poses shown as blue '+' in Fig. E.2. The demonstrator policy plans in two stages, where in each stage the demonstrator tries to place and wrap the chain over each peg. Unlike other tasks, the *chaining* task often requires more than 2 steps to successfully execute the demonstration, as the demonstration policy may sometimes need a couple of steps to properly place the chain around each peg. However, it usually suffices to let the demonstrator policy plan for 3 steps in the demonstration roll outs for a reasonable demonstration success rate.

E.2.3 Routing

E.2.3.1 Overview

For the *routing* task, the agent is tasked to route the cable of beads around the fixed pegs. The peg positions are generated at random, with the randomized parameters set to be alternating such that the cable needs to wind in different directions every peg. With the current randomization settings, the environment is initialized with 2 or 3 pegs at various positions. Similar to the *chaining* task, the cable is made up of connected beads, where the left end of the cable is anchored to the workspace. The cable length once again varies with the task initialization, but are programmed such that the cable is sufficiently long to route the cable fully around all the pegs with some slack.

E.2.3.2 Metrics

Similar to the *chaining* task, the evaluation metric checks whether the cable is wrapped around the pegs, which are shown as green '+' in Fig. E.2. There is another evaluation goal position after the pegs, which checks whether the cable is fully stretched out after routing around the pegs. This ensures that the cable is properly routed around the peg, which demonstrator policy sometimes fails to do for the last peg. It also adds on an additional challenge for the agents to learn, as the resulting cable stretching move at the last step is a different type of maneuver compared to placing the cables over the pegs via Reidemeister-type moves.

Unlike other tasks, the *routing* task requires that all the pegs be routed sequentially, where each sequentially correct moves will result in a delta reward of $1/(n+1)$ where n is the number of pegs (we divide by $(n+1)$, as the task requires n routing moves and 1 stretching move to be completed). This means that routing the cable around first and third pegs will only result in a reward for first peg. We enforce this sequential planning rule, as we noticed that out-of-order peg routing can often cause the *routing* task to fail due to the cable often getting stuck in some configuration and the demonstrator and agents cannot effectively finish the task without undoing the routing moves.

E.2.3.3 Demonstrator Policy

The demonstrator policy attempts to place the beads at each of their pre-specified bead-level goal poses shown as blue '+' in Fig. E.2, similar to the *chaining* task. The demonstrator policy plans in multiple stages, where in each stage the demonstrator attempts to route the cable around the peg sequentially from left to right, using Reidemeister-type moves to lift the cable over the pegs and place it in the desired position. For the last stage, the demonstrator then stretches the cable such that the last yellow-colored bead is in the desired goal position.

E.2.4 Stacking

E.2.4.1 Overview

For the *stacking* task, it is similar to how it is defined in Ravens. We make an additional modification that now the block goal positions are randomized with color such that goal-conditioning is necessary. We also initialize the blocks in a 2×3 array for easy initialization and placement of task when loading multiple task modules together to avoid bad initialization that overlaps with other tasks. The orientation for the base is randomized at 8 cardinal half-directions: $\theta \in \{\theta : \theta \equiv \pi k/4, k \in [0, \cdots, 7]\}$.

E.2.4.2 Metrics

The metrics are simple goal positioned metrics for each box, checking that each boxes are stacked in the right place within small margin of error.

E.2.4.3 Demonstrator Policy

The demonstrator policy operates similar to how it does in Ravens, where it stacks all three bottom boxes in the first stage, the next two middle boxes in the second stage, and the one top box in the third stage.

E.2.5 Visualizations

In this section, we provide visualizations of the task initialization, intermediate states, and goal configurations for all 10 single and multi-task problems. For each problem, we provide sequences of 3 images, where the first image corresponds to the initial state, the middle image corresponds an arbitrary intermediate state, and the last image corresponds to the final goal configuration. The complete set of visualizations is given in Fig. E.3.

E.3 Determining Sampling Weights

In Section 10.3.2, we have determined the task-level and step-level weights through black box optimization. For the black box optimization, we fix the number of demos to 1000, train the SCTN agent for 20K iterations, with snapshot evaluations taken every 2K iterations that is evaluated only on one random seed. The evaluation metric for a given weighting scheme is the maximum evaluation performance on 20 test-time episodes over all 10 snapshots.

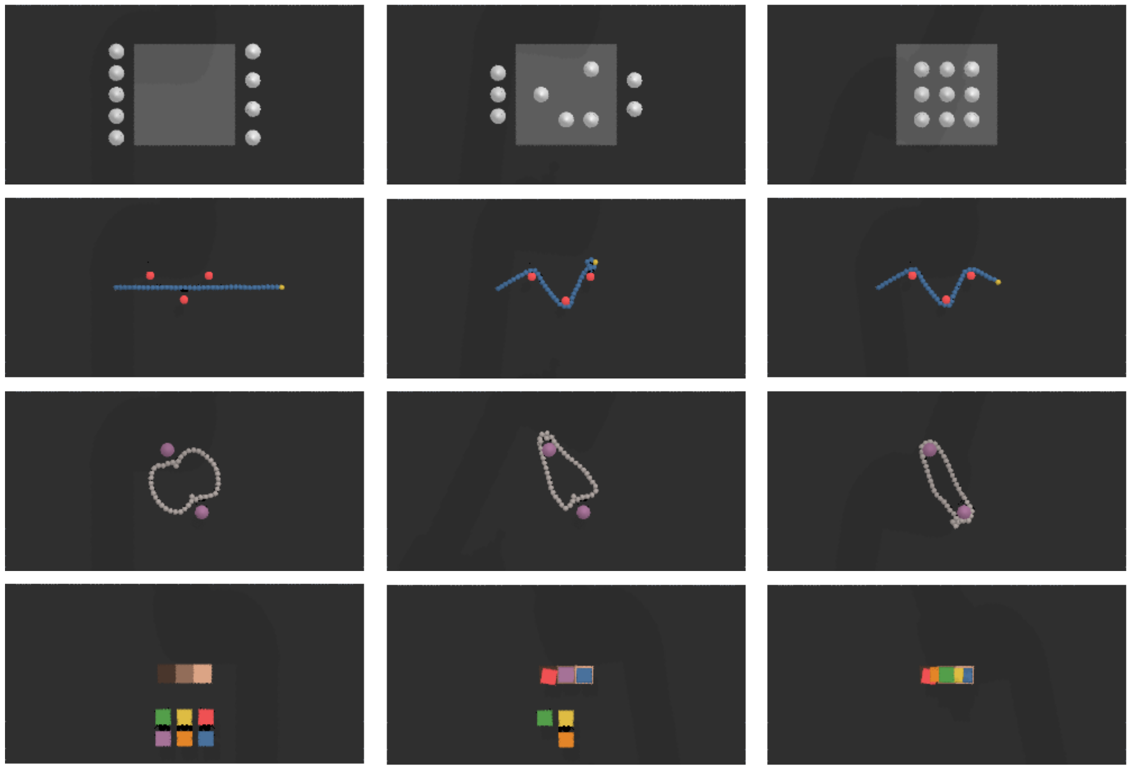


Figure E.3: **Visualization of the top-down image sequences of solving each task module**: *placing, routing, chaining*, and *stacking* (top to bottom).

The hyperparameters to be optimized are the weighting scheme and the maximum weight $w_{\max} \in [1.5, 5.0]$. The minimum weight is fixed to be 1 for all weighting schemes. The three weighting schemes tested are the following:

- **Linear:** Linearly increasing weighting scheme, where the weight of first step is 1.0 and the last step is $w_{\max}$, and the weights of other steps are interpolated linearly depending on the episode length.
- **Last:** Weighting scheme in which the weight of last step is $w_{\max}$ and all other steps are 1.0.
- **First-Last:** Weighting scheme in which the weight of first last steps are $w_{\max}$ and all other steps are 1.0.

We choose these weighting schemes since we often empirically observed that weighing the later steps of the demonstrations usually provided better learning for multi-task problems.

The inferred task complexity is determined by the first snapshot in which it achieves maximum evaluation performance.

Module	Inferred Complexity	Weighting Scheme [min. & max. weights]	Mean Completion	Mean Reward
placing	4K Steps	Linear: [1.0 ▁▂▃▅ 2.76]	100%	1.0
chaining	20K Steps	Last: [1.0 ▁▁▁▅ 1.92]	90%	0.95
routing	20K Steps	Linear: [1.0 ▁▂▃▅ 1.76]	25%	0.52
stacking	12K Steps	Linear: [1.0 ▁▂▃▅ 1.50]	90%	0.94

Table E.1: **Hyperparameters for SCTN weighted sampling and evaluation statistics for each task.**

E.4 Additional Experiment Results

We provide more detailed evaluation statistics for the 10 single-task and multi-task problems. Fig. E.4 contains learning curves showing the mean completion rates of different agents over 20×5 test-time episode evaluations as a function of training iterations for single-task problems, grouped by different numbers of demonstrations.

The agents are trained with trained for 20K $\times$ m iterations, where m is the number of task modules in a given problem, and one training iteration consists of one gradient update from data of batch size 1. This means that agents are trained for 20K training iterations for single-task problems, 40K for double-task problems, and 60K for triple-task problems. At each snapshot occurring every tenth of total training iterations, we evaluate the agents on 20 test episodes. These 20 episodes are generated using a held-out set of random seeds for testing, which does not overlap with random seeds used for generating demonstrations. We repeat this training procedure 5 times over 5 different TensorFlow training seeds in order to test consistency of our learning results, totaling $20 \times 5 = 100$ metrics for 10 snapshots of the model. Figs. E.5, E.6, E.7 and E.8 contain learning curves for multi-task problems.

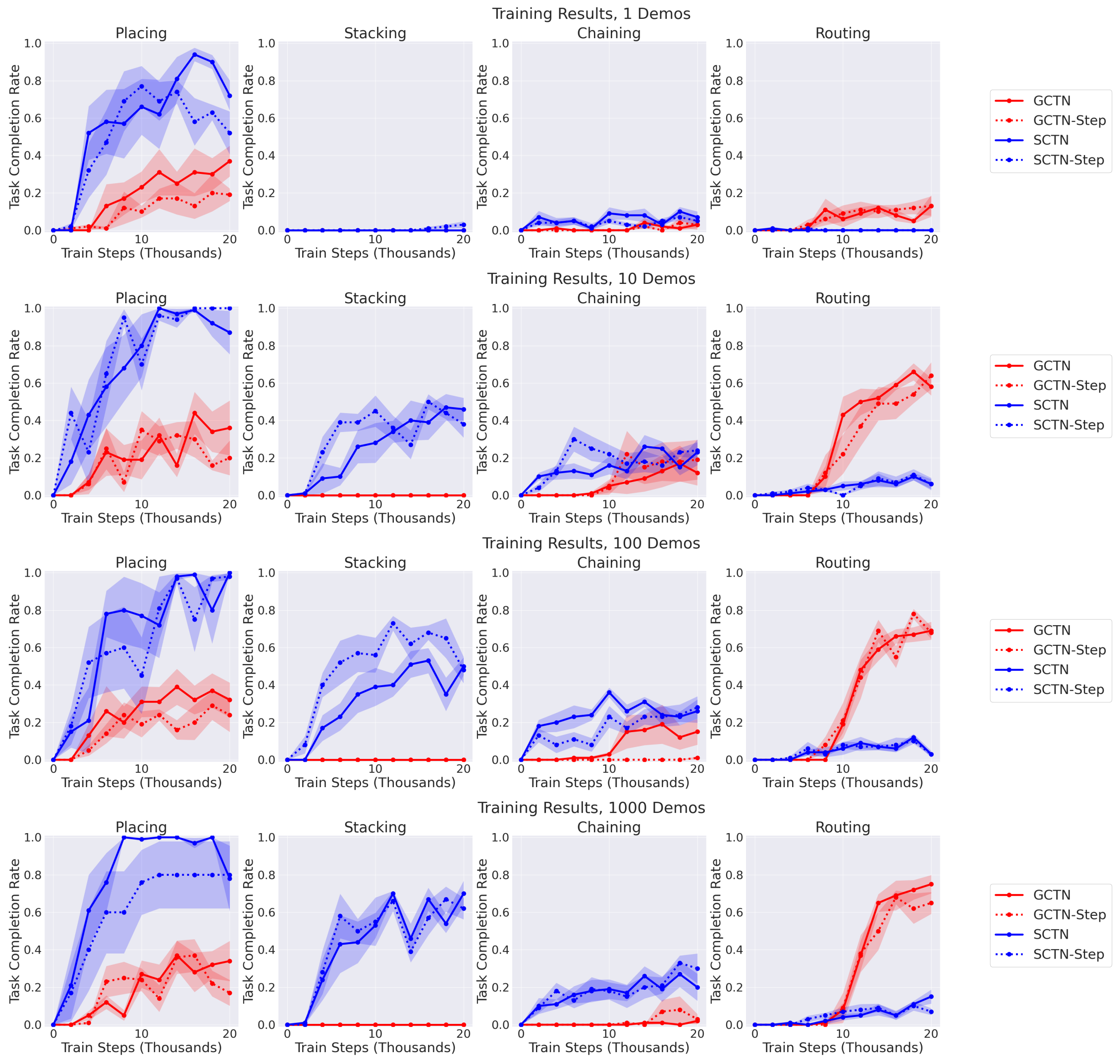


Figure E.4: **Results for models on the single-task problems**, grouped by numbers of demonstrations the agents are trained on. The mean task completion rate over 20×5 test-time episode evaluations is plotted against the number of training steps in the unit of thousands. All single-task problems are trained with 20K training iterations.

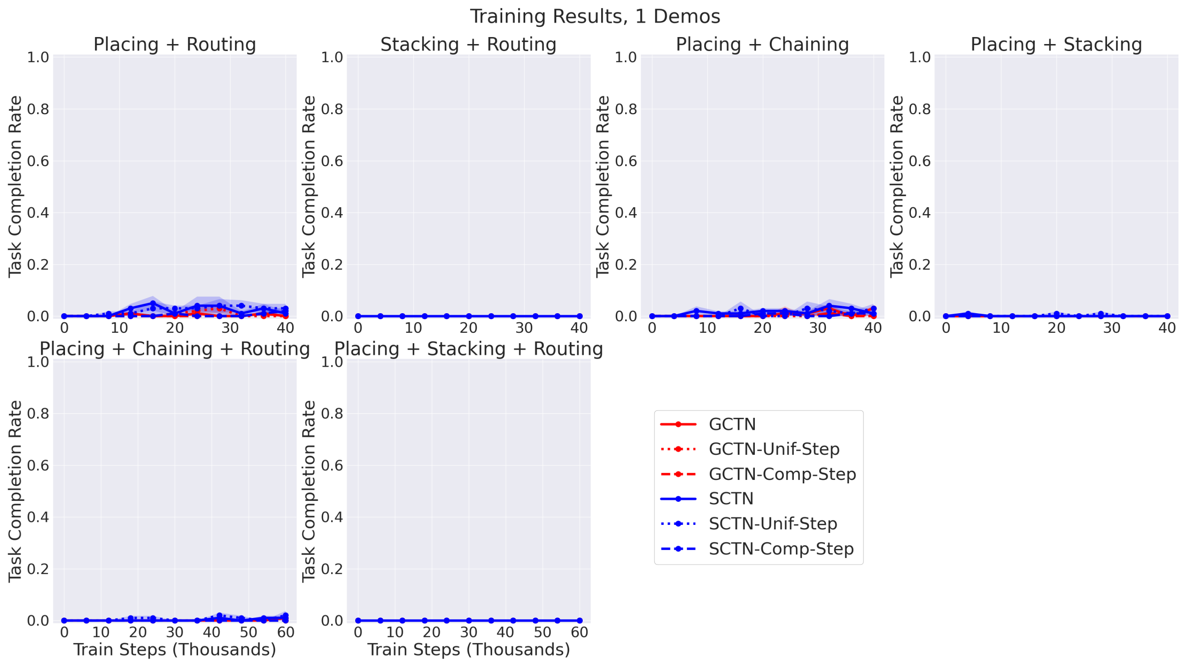


Figure E.5: **Results for models on the multi-task problems trained on 1 demonstration**. The plotting format follows Fig. E.4. All double-task problems are trained with 40K training iterations, and triple-task problems with 60K training iterations.

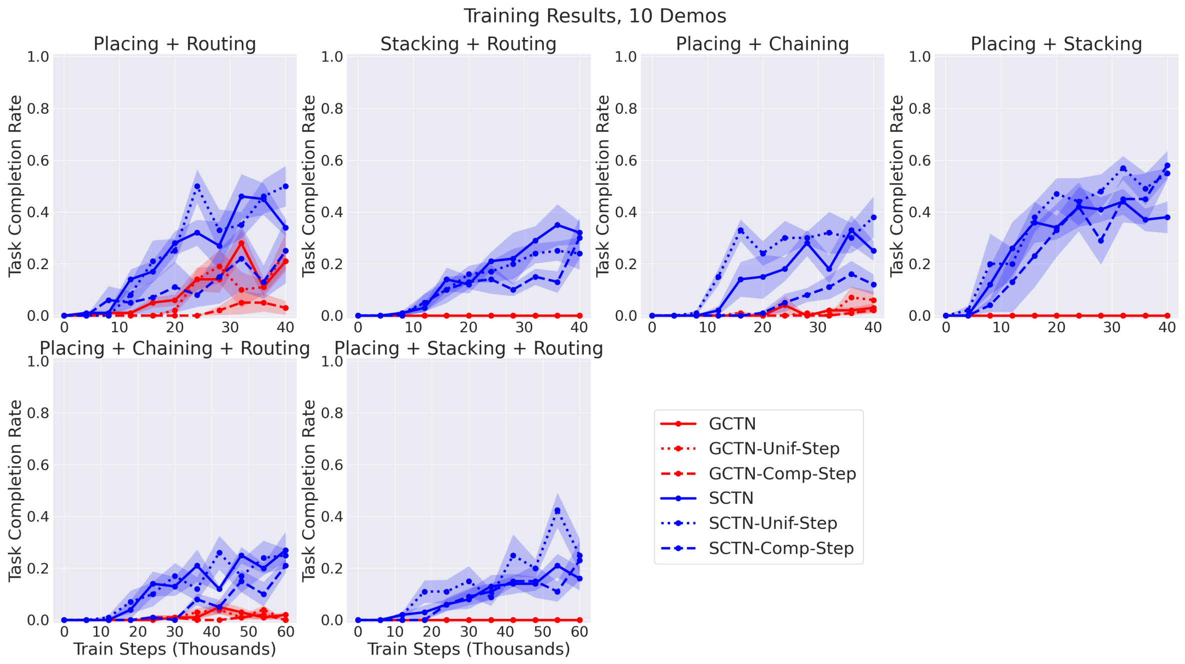


Figure E.6: **Results for models on the multi-task problems trained on 10 demonstrations**. The plotting format follows Fig. E.5.

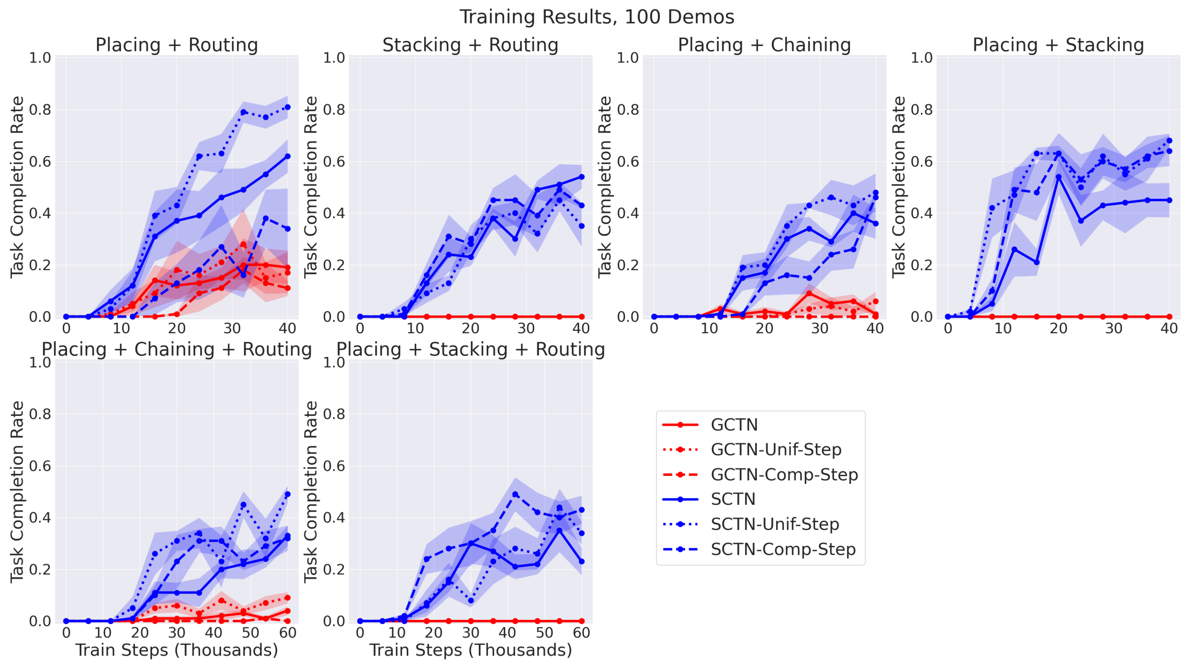


Figure E.7: **Results for models on the multi-task problems trained on 100 demonstrations**. The plotting format follows Fig. E.5.

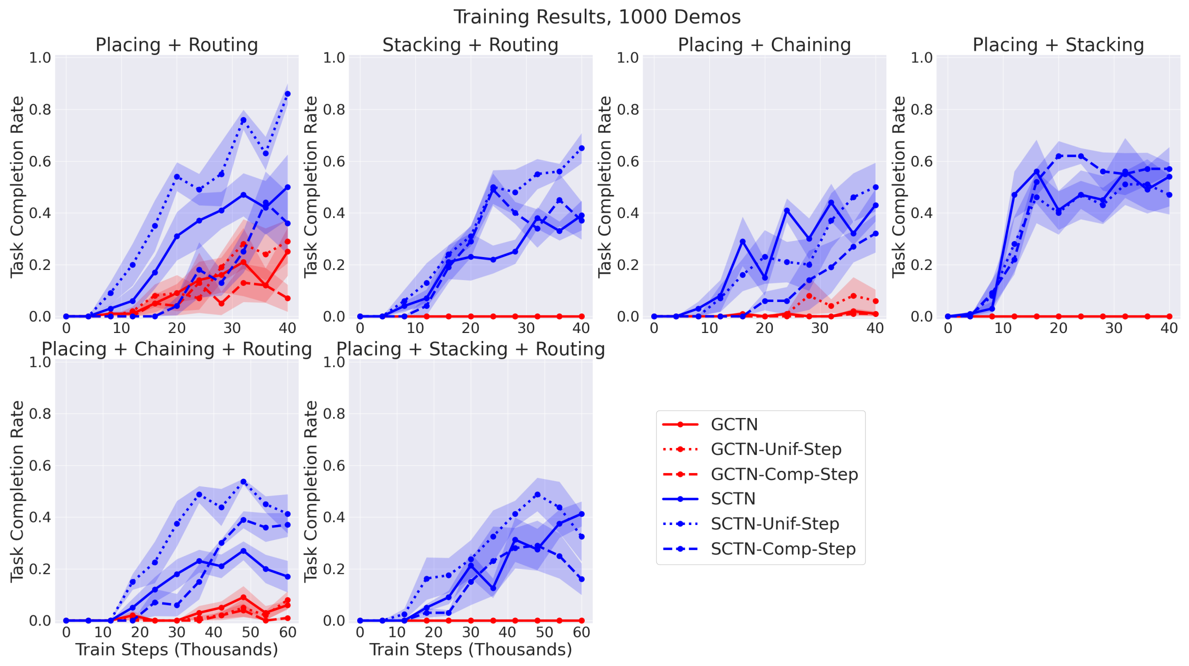


Figure E.8: **Results for models on the multi-task problems trained on 1000 demonstrations**. The plotting format follows Fig. E.5.

Appendix F

Details for Navigation Between Initial and Desired Community States Using Shortcuts

F.1 Supporting Information

This section provides the supporting figures and tables for Chapter 12.

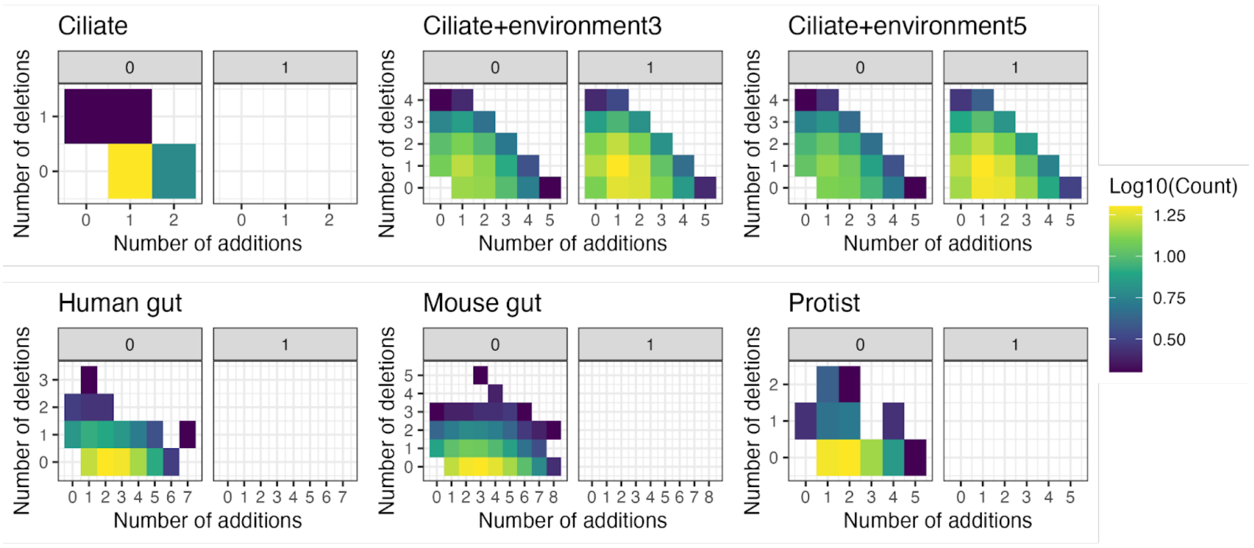


Figure F.1: **State Diagram Actions Distribution.** Distribution of actions within edges for the state diagrams shown in Fig. 12.2. Panels indicate the number of ϵ-additions (x-axis), ϵ-deletions (y-axis), and environmental change actions (facets) comprising each edge in the state diagram.

Sp.	**Labels**	**Taxon Name**	**Sp.**	**Labels**	**Taxon Name**
Ciliate (Maynard)			**Mouse Gut (Bucci)**		
1	CO	*Colpidium striatum*	1	Barne	*Barnesiella*
2	DE	*Dexiostoma campylum*	2	und. Lachn	*und. Lachnospiraceae*
3	LO	*Loxocephalus sp.*	3	uncl. Lachn	*uncl. Lachnospiraceae*
4	PA	*Paramecium caudatum*	4	Other	*Other*
5	SP	*Spirostomum teres*	5	Blaut	*Blautia*
Human Gut (Venturelli)			6	und. uncl. Molli	*und. uncl. Mollicutes*
1	BH	*Blautia hydrogenotrophica*	7	Akker	*Akkermansia*
2	CA	*Collinsella aerofaciens*	8	Copro	*Coprobacillus*
3	BU	*Bacteroides uniformis*	9	Clost diffi	*Clostridium difficile*
4	PC	*Prevotella copri*	10	Enter	*Enterococcus*
5	BO	*Bacteroides ovatus*	11	und. Enter	*und. Enterobacteriaceae*
6	BV	*Bacteroides vulgatus*	**Protist (Carrara)**		
7	BT	*Bacteroides thetaiotaomicron*	1	Chi	*Chilomonas sp.*
8	EL	*Eggerthella lenta*	2	Cyc	*Cyclidium sp.*
9	FP	*Faecalibacterium prausnitzii*	3	Tet	*Tetrahymena sp.*
10	CH	*Clostridium hiranonis*	4	Dex	*Dexiostoma sp.*
11	DP	*Desulfovibrio piger*	5	Col	*Colpidium sp.*
12	ER	*Eubacterium rectale*	6	Pau	*Paramecium aurelia*
			7	Cep	*Cephalodella sp.*
			8	Spi	*Spirostomum sp.*
			9	Eug	*Euglena gracilis*
			10	Eup	*Euplotes aediculatus*
			11	Pbu	*Paramecium bursaria*

Table F.1: **Ecological Navigation Empirical Dataset Species Detail.** Taxon names within each dataset. Column 'Sp.' indicates the numeric species abbreviation used in the code repository. Column 'Labels' indicates the taxon abbreviation in code. Column 'Taxon Name' indicates the scientific name of the taxon.

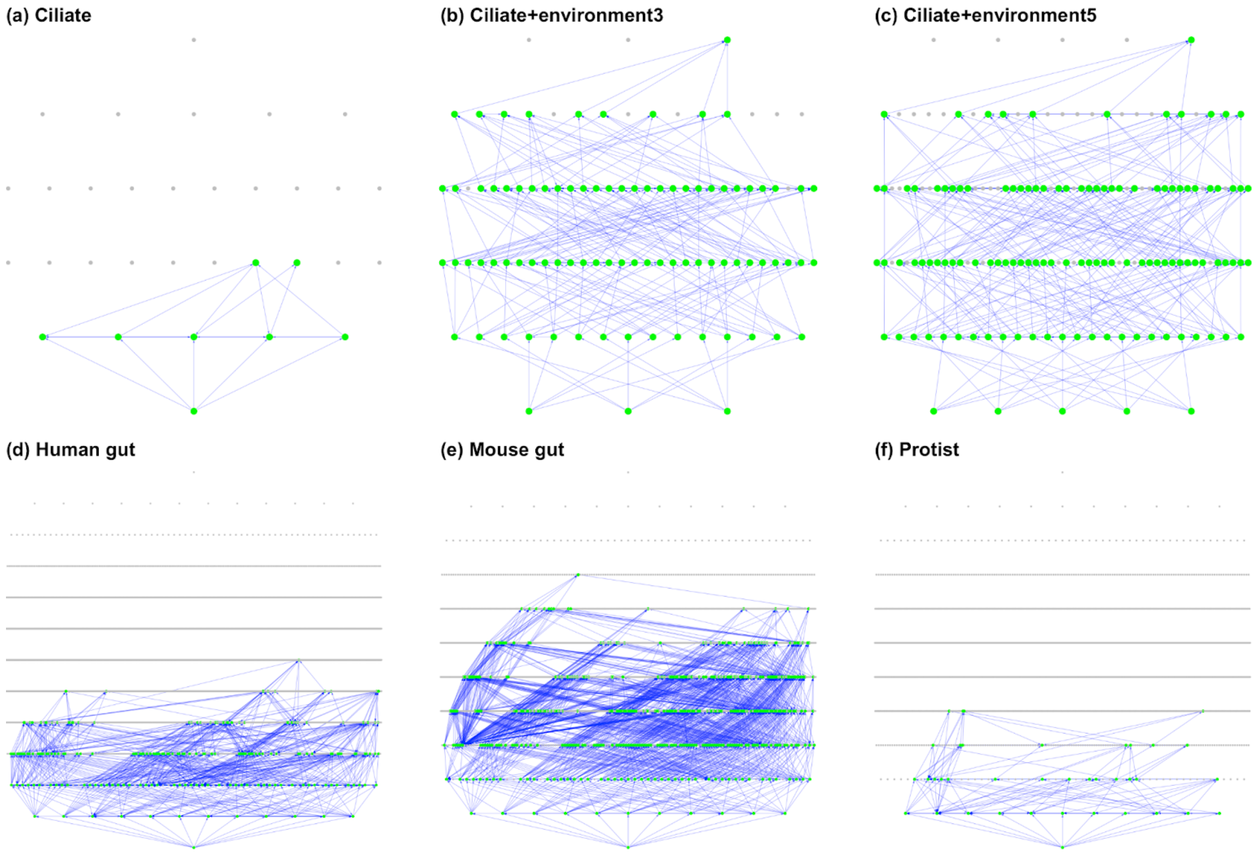


Figure F.2: **Invasion Graphs.** Invasion graphs for all datasets. Notation and parameters are the same as in Fig. 12.2, except that transient states are not colored orange.

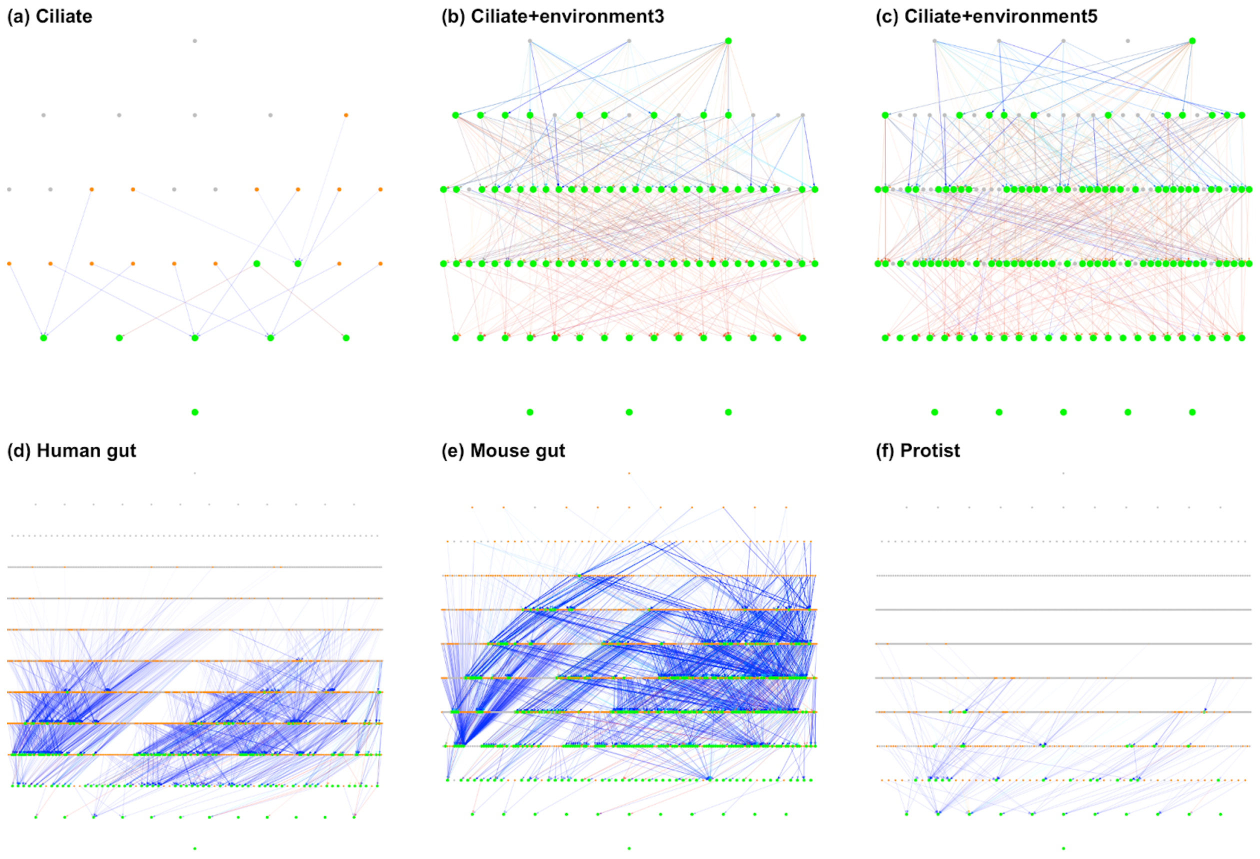


Figure F.3: **Un-invasion graphs.** Un-invasion graphs for all datasets. Notation and parameters are the same as in Fig. 12.2.

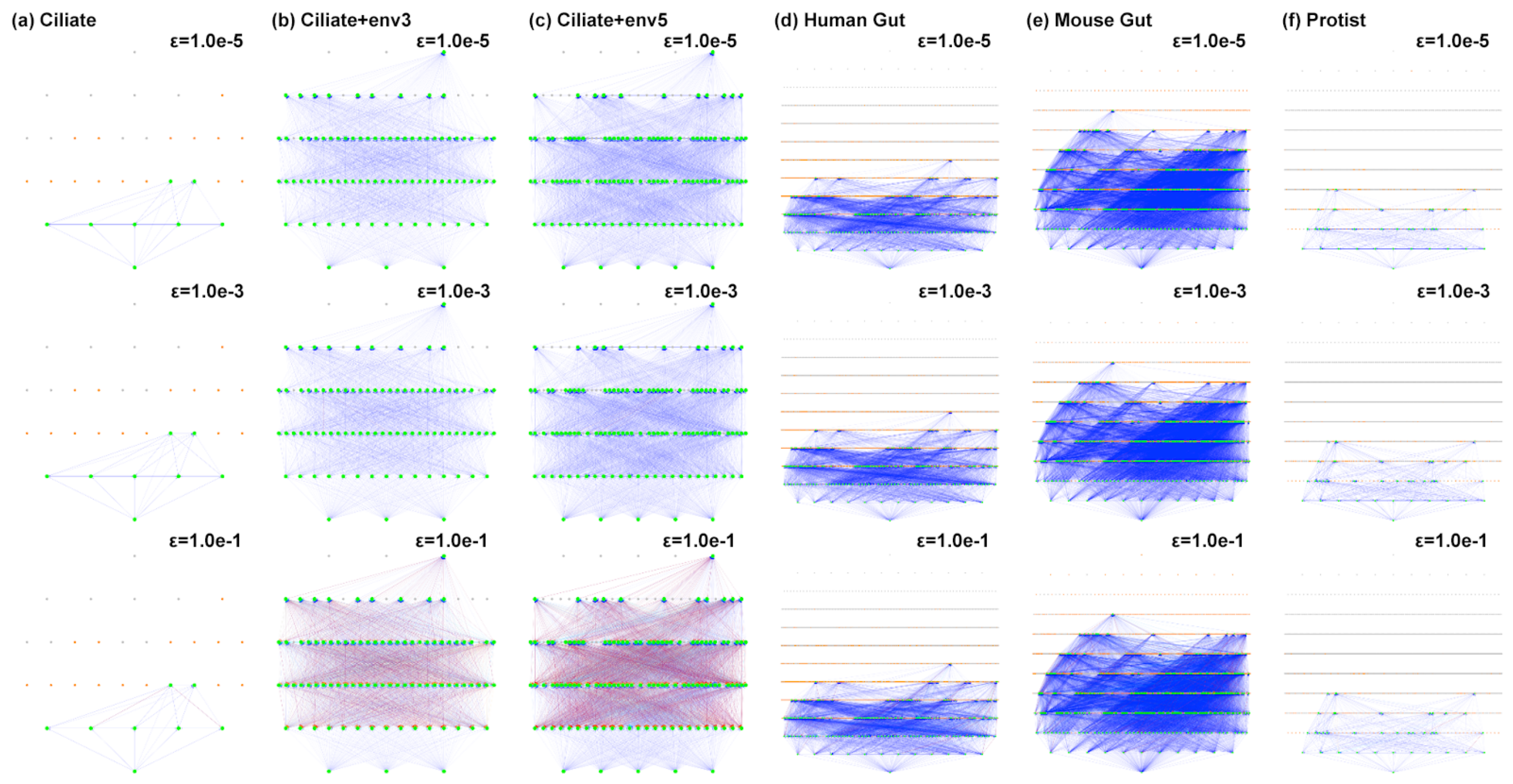


Figure F.4: **State Diagram Properties.** State diagram properties vary with ϵ. Columns indicate datasets; rows indicate low to high values of ϵ. All labels are as in Fig. 12.2.

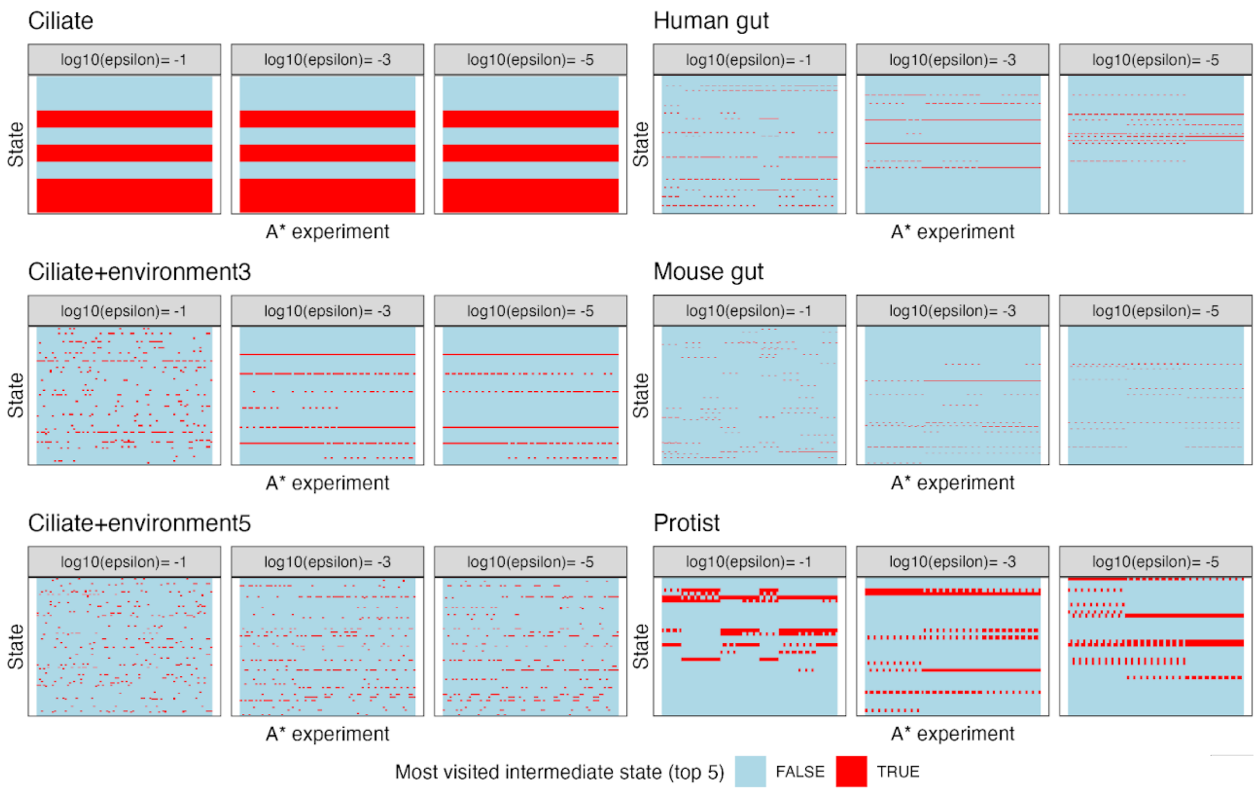


Figure F.5: **Most-visited Intermediate States.** For each dataset and A* experiment we identified the intermediate states that were most commonly visited in navigation among all state pairs. We then rank-ordered these states by their prevalence within each dataset and A* experiment. Panels show states on the y-axis and experiments on the x-axis. Top-5 common states are colored in red; all others in blue. Panels are faceted by ϵ. Columns within each panel indicate different experiments, i.e. different combinations of costs C_i. The prevalence of apparent horizontal red lines within each each facet indicates that the identity of most-visited intermediate states is sometimes not strongly dependent on C_q.

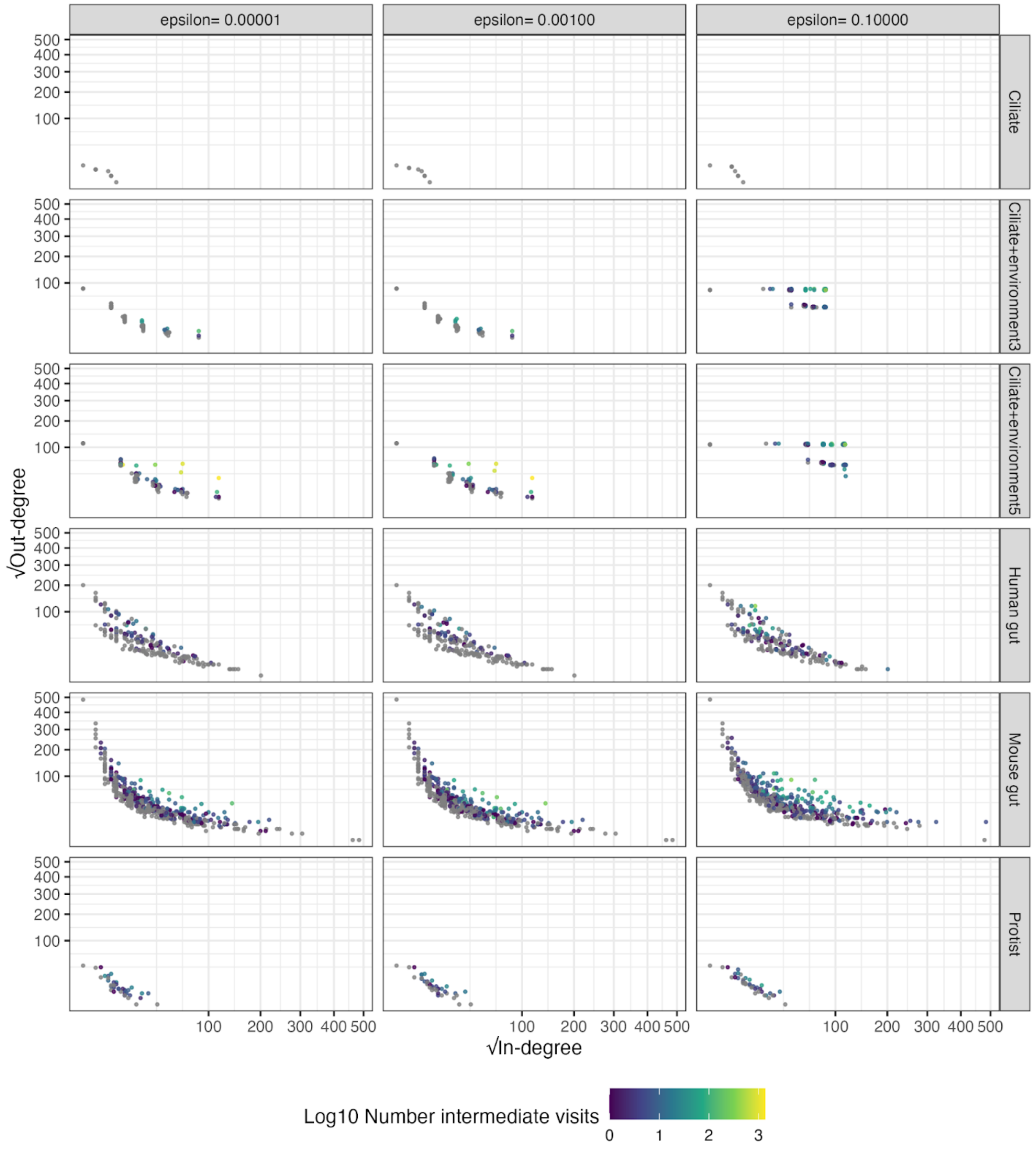


Figure F.6: **Network Toplogy Summary.** Summary of network topology for state diagrams for different datasets (rows) and ϵ values (columns). Facet points indicate in- and out-degree for each state, and are colored by the number of intermediate visits. Gray points indicate states that are not reachable by non-brute-force navigation. Data are shown for a case where all $C_q = 1$ for all action types; results do not vary strongly with C_q (not shown).

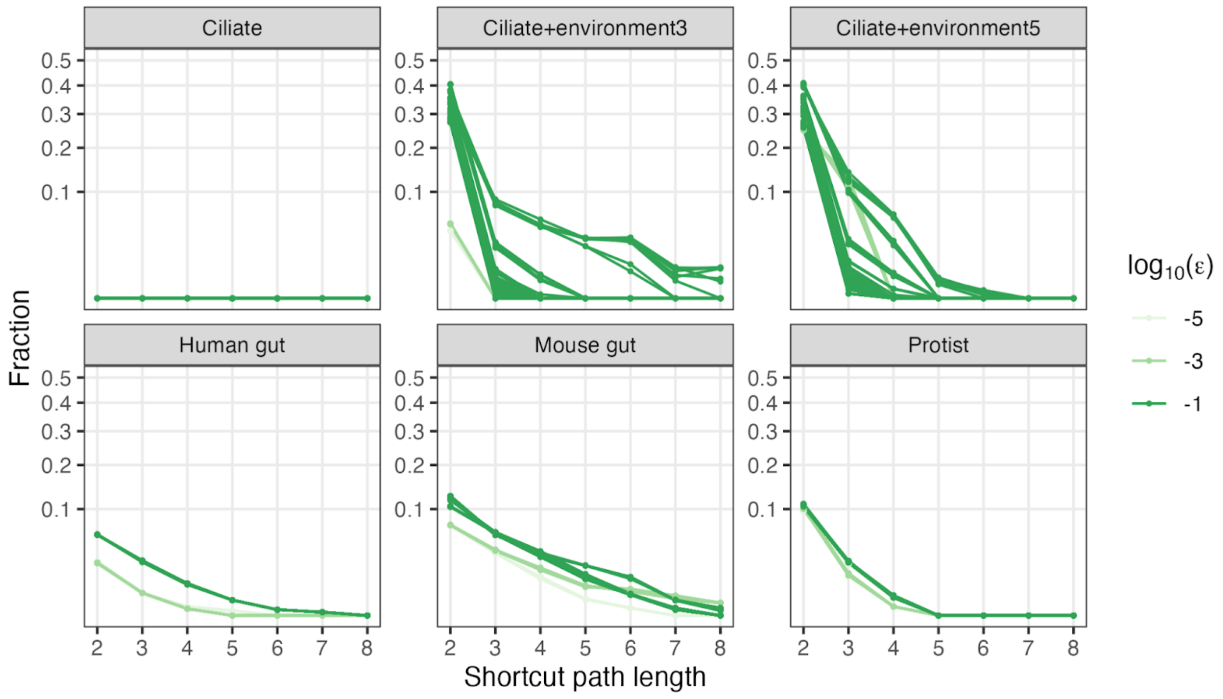


Figure F.7: **Distribution of path lengths among shortcuts.** Panels are faceted by dataset; line color indicates ϵ; multiple lines represent variation due to assumed costs C_q.

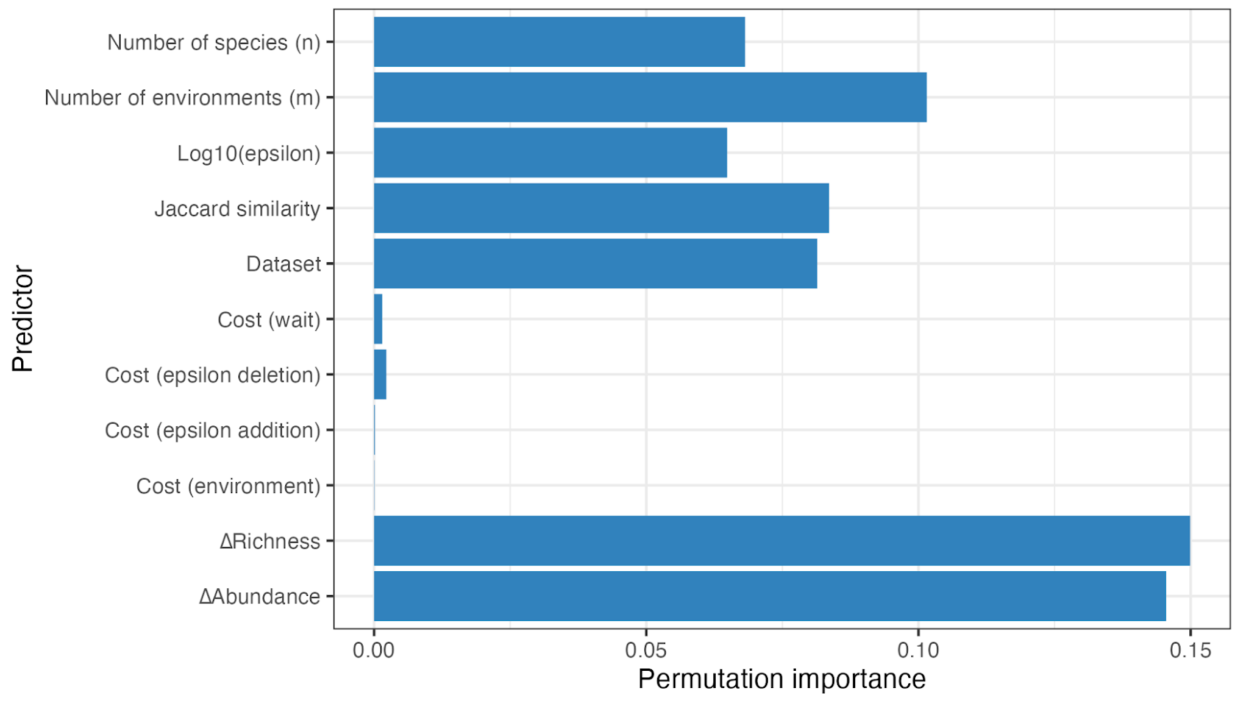


Figure F.8: **Random Forest Variable Importance.** Variable importance plot for the random forest model shown in Fig. 12.5.

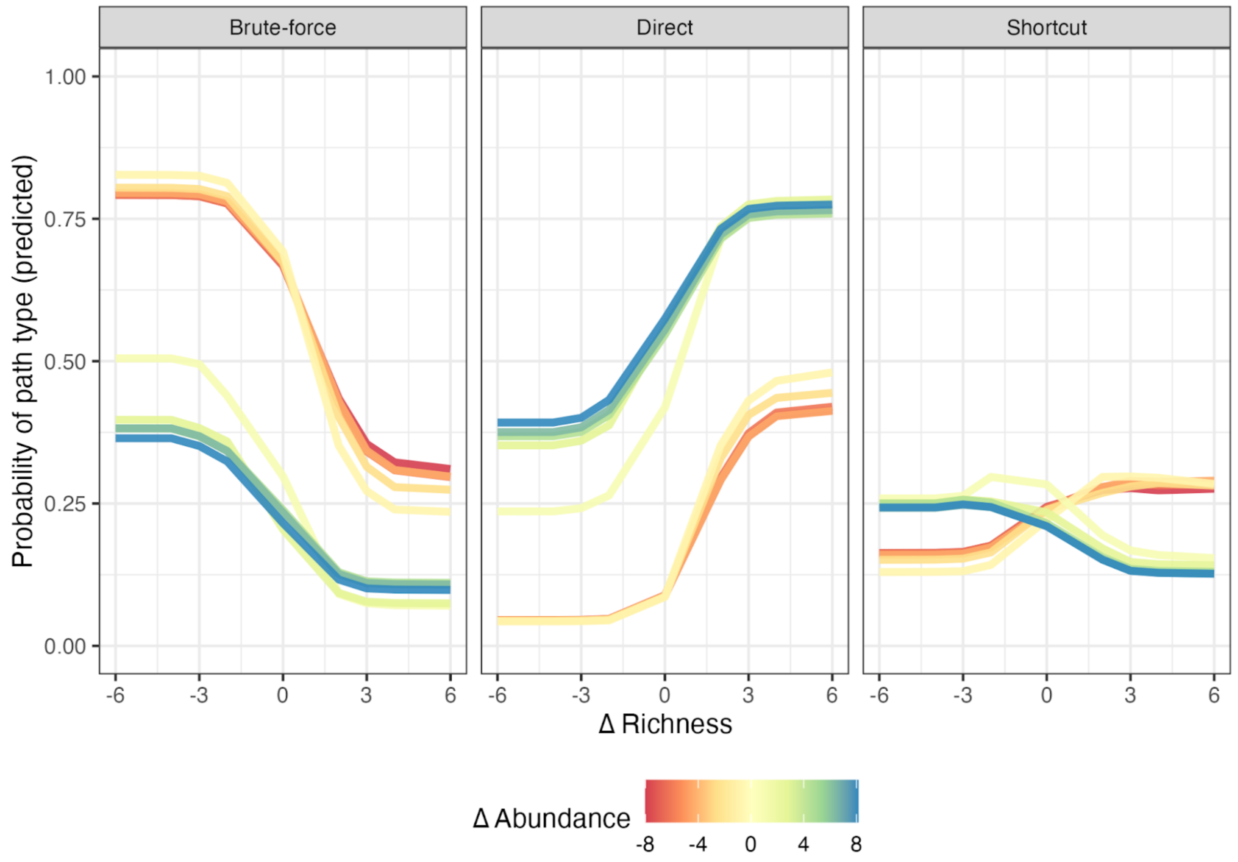


Figure F.9: **Random Forest Partial Dependence.** Partial dependence plots indicate the effect of ΔAbundance and ΔRichness on the probability of navigation yielding no path (brute-force solution), a direct path, or a shortcut path. Δs are defined as desired state values minus initial state values.