EXPLOITING INSENSITIVITY IN STOCHASTIC SYSTEMS TO LEARN APPROXIMATELY OPTIMAL POLICIES

BY

JAMES C. DAVIDSON

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Doctoral Committee:

Professor Seth A. Hutchinson, Chair
Associate Professor Eyal Amir
Assistant Professor Maxim Raginsky
Assistant Professor Enlu Zhou

ABSTRACT

How does uncertainty affect a robot when attempting to generate a control policy to achieve some objective? How sensitive is the obtained control policy to perturbations? These are the central questions addressed in this dissertation. For most real-world robotic systems, the state of the system is observed only indirectly through limited sensor modalities. Since the actual state of the robot is not fully observable, partially observable information is all that is available to infer the state of the system. Further complicating matters, the system may be subject to disturbances that not only perturb the evolution of the system but also perturb the sensor data. Determining policies to effectively and efficiently govern the behavior of the system relative to a stated objective becomes computationally burdensome and, for many systems, impractical for the exact case. Thus, much research has been devoted to determining approximately optimal solutions for these partially observed Markov decision processes (POMDPs).

The techniques presented herein exploit the inherent insensitivity in POMDPs based on the notion that small changes in a policy have little impact on the quality of the solution except at a small set of critical points. First, a hierarchical method for determining nearly optimal policies is presented that achieves temporal and spatial abstraction though local approximations. Through a mixed simulation and analytic representation, a directed graph is generated to determine the underlying POMDP structure. The result is a multi-query method for generating the structural representation offline. The graph is generated by randomly sampling vertices. Local policies are then used to connect to the newly added vertices. A new edge is added if the local policy was successful. By continuing to extend the

graph at each iteration of the algorithm, a sparse representation is obtained. Theoretical and simulation-based results are provided to demonstrate the effectiveness of this approach. The second technique extends the methodology of the first technique to an anytime algorithm. Adaptive sampling is used to quickly and effective determine nearly optimal policies. Between exploitation and exploration sampling, the structural representation is expanded based on inductive bias on the past performance of the sampling algorithm in the neighborhood of a perspective sample. In this way, we are able to preferentially sample policies that are both more likely to result in better exploration and also more likely to increase the connectivity in a region of the space that has a lower cost. Finally, a perturbation analysis framework is developed. This serves two purposes. First, the derived analysis is used to support the hypothesis that POMDPs are often insensitive and to identify when they are not. Secondly, the perturbation analysis framework enables the chaining of forecasted evolutions together into a compact representation. This compact representation provides even greater temporal and spatial abstraction in an analytic representation.

To Amaya and Dotti

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Finding ways to more effectively cope with the uncertainty ubiquitous to real robotic systems has been a focus of robotics research for decades. The uncertainty in such robotic systems can be categorized as

- Process uncertainty, which models uncertainty in the robot's actions; and
- Measurement uncertainty, which models uncertainty in the robot's knowledge of its own state.[1]

By explicitly considering both types of uncertainty, we alleviate their effect on the robotic system.

Process uncertainty can be modeled by a Markov decision process (MDP), whereby the outcome of an action given the system is in some state is modeled by a random vector and associated probability function. Such a model, however, assumes that the state of the system is known after each action. This is often not the case in robotic systems. In such cases the state of the system is only partially observable due to limitations of the sensing capabilities of the robot. Further, it may be the case that the sensors produce nondeterministic measurements. Partial observability and nondeterministic sensor observations are captured by the measurement uncertainty model.

[1]Process and measurement uncertainty can be split into a third, dependent type of uncertainty known as environment uncertainty. Modeling environment uncertainty explicitly separates uncertainty inherent in the robotic platform from the uncertainty in the layout of the robot's environment. Addressing this type of uncertainty is the domain of simultaneous localization and mapping (SLAM) research (refer to [1] for an overview of SLAM research).

Partially observable Markov decision processes (POMDPs) provide a general and expressive formalism for representing both process and measurement uncertainty (both the execution of actions and observations from sensor data) with stochastic models. However, their use for planning in robotics problems has been limited due to problems of scalability and tractability. Evaluating the optimal policy can be computationally intractable with a worst case running time that is exponential in both the length of the time horizon and the number of observations for the exact solution [2].

Given the importance of finding optimal or nearly optimal solutions for systems subject to uncertainty, numerous researchers have developed approaches to approximate POMDP systems (refer to [3, Ch. 15 & 16] and [4] for surveys of such approaches). In recent years, a number of sampling-based approaches for generating policies have been proposed [5–11] to find efficient approximation methods. These methods have been used successfully to approximate the POMDP value function (i.e., the expected cost-to-go function). Several of these approaches have moved past synthetic experiments to motion planning with real robotic systems, such as [11, 12]. However, these approaches rely on either a coarse representation of state or simplified system dynamics and uncertainty models, which likely limit their capability of solving more difficult real world robotic tasks.

When described in terms of a system's state space, the evolution of a POMDP is governed by a set of transition probabilities that describe the effects of control actions, and an observation model that specifies uncertainty in the sensing process. If, instead, the system is described in terms of the belief space (i.e., the space of possible a posteriori probability functions on the state space), the evolution of the system can be modeled as an MDP. This corresponds to lifting the system description from a lower dimensional state space to a higher dimensional belief space. Most POMDP optimization algorithms operate, and approximate the system, at the level of the belief space.

We propose a qualitatively different approach, whereby the analysis is lifted from the belief space into the higher dimensional space of probability functions that can be defined

on the belief space. We refer to this lifted space as the *hyperbelief space.*[2] In the hyperbelief space, the system evolves deterministically, thus eliminating (in some sense) the explicit consideration of uncertainty during the planning process, at the expense of a much higher representational cost. Operating in the hyperbelief space produces challenges analogous to those encountered when operating in the belief space, e.g. exponential growth of reachable belief states. One of the challenges that manifests when operating in the hyperbelief space is that, in general, a POMDP system cannot reach arbitrary points in the space, regardless of the quality of control law. This occurs even though the hyperbelief evolution is deterministic. To account for this, we explicitly analyze the sensitivity of both cost and hyperbelief evolution functions with respect to perturbations in the hyperbelief state. This allows us to adapt a simulated cost and trajectory starting from one hyperbelief state to a cost and trajectory starting at a nearby hyperbelief, without re-simulation of the system. However, moving to a hyperbelief-based representation allows us to apply standard approximation techniques. For example, because the complexity of hyperbelief states is high, we employ particle filtering techniques to approximate the evolution of the system in the hyperbelief space. The formulation and analysis of the system's evolution in the hyperbelief space is provided in Chapter 4.

We will present two POMDP learning techniques that use the hyperbelief space and associated hyperfiltering techniques to forecast the behavior of POMDP systems via a discrete graph representation. The second method is derived from the first but deviates from the first in its intent and its approach. Both methods are inspired by the dual purpose of generating a structural representation while simultaneously generating approximately optimal policies for POMDPs with very large state spaces. The resulting methods achieve this through a combination of spatial and temporal abstraction. For both methods we construct

[2]The hyperbelief space refers to the space of probability functions over probability functions. *Hyperparameters* describe a distribution over the parameters of a prior probability function and are often used as conjugate priors. While both formulations have some similarities, the hyperbelief space and the related hyperfiltering extends beyond the prior to the uncertainty introduced by the unknown observations of the POMDP at each iteration.

a directed graph that represents the uncertain evolution of the system and can be used to find nearly-optimal policies for POMDP systems. The vertices of the directed graph represent hyperbeliefs and are generated by sampling. The edges are generated by simulation of the system for multiple stages, e.g. time steps, using simple local policies. The edges represent transitions from the hyperbelief represented by the source vertex to a region near the hyperbelief represented by the target vertex. We can translate a walk through the graph to a connected path through the hyperbelief space. These paths through the hyperbelief space correspond to feedback policies for the system. Retaining the sensitivity along edges of the graph also allows reuse of the data structure without re-simulation of the system. These spatial and temporal abstractions are key when planning for systems that have not only increasingly large state, action, and observation spaces, but also require long planning horizons to find sufficiently good policies. These properties give the presented algorithms advantages in terms of both scalability and practicality as demonstrated by experimental results and theoretical analysis.

The first method, the Sampling Hyperbelief Optimization Technique (SHOT), is a multi-query technique whereby a structural representation is built offline independent of the cost function. Then, when an objective is specified, the approximately optimal policy can be found quickly. During the offline process, SHOT generates the digraph and stores pertinent data about the evolution of the system. This includes a representation of the evolution of the system under the specified local policy as well as the distance to the target hyperbelief. Vertices are generated by randomly generating hyperbeliefs in the hyperbelief space. Edges to each newly generated hyperbelief are instantiated by simulating a local, greedy policy from a set of the neighboring vertices.

In this approach an optimization method adapted from standard graph optimization methods is used to determine the optimal policy in the generated graph. At the cost of exactness, this graph representation reduces the set of possibilities to be explored from a continuum in an infinite number of dimensions to a finite set. Moreover, optimizing over

the graph can be performed in worst case time complexity that is $O(|N|\,|E|)$, where $|N|$ is the number of vertices and $|E|$ is the number of edges in the digraph, using standard graph optimization algorithms.

An edge between a source hyperbelief sample and a target hyperbelief sample may be included in the graph even if the target cannot be reached from the source; a measurement of the distance to the target is included in the edge information. This distance measure is used during the graph optimization stage to bound the cost between edges. Lipschitz bounding functions are generated to estimate both the upper and lower bounds around each sample. These bounds provide a means to evaluate the performance of a local policy in the region around the starting hyperbelief sample. By generating an approximation for both the value and the evolution around the starting position for each edge in the graph, we are able to represent the performance beyond the finite set of sampled hyperbeliefs, thus achieving spatial abstraction. However, because the local policy may not attain the exact position of the target hyperbelief sample, a refinement algorithm is used to incrementally select a possible better policy and then simulate this policy to determine the actual cost of this policy and, thus, if it is better. The result is an algorithm, whereby the difference from the bounding value and the optimal value decreases with every iteration so that the method will find the optimal solution for the given graph in a finite amount of time (as there are a finite number of paths in the digraph). The details of this method are provided in Chapter 5.

The second learning method, Adaptive Exploration/exploitation Sampling-based Optimization for POMDPs (AESOP), is an evolution of the first approach. AESOP is an online, anytime algorithm. This anytime formulation is achieved at the cost of the multi-query representation. AESOP generates policies whose quality improves with increased planning effort and can be terminated at any point and provide the best policy found thus far. Moreover, AESOP expands the representation of policies from a single edge to multiple edges. This endows the technique with the ability to balance the computational burden of repre-

senting each vertex as a single belief (greater number of vertices) and the required number of vertices when retaining the complete evolution as a single vertex (greater number of edges).

Unlike SHOT, AESOP explicitly considers both exploration and exploitation. During an exploration phase, local learning methods are used to predict which of the local policies are most likely to expand the graph into an unexplored region of the hyperbelief space. After the local policies are simulated, those that were the most diverse–being the farthest away from the already explored space–are retained and added to the graph. Conversely, during an exploitation phase, the differential in the neighborhood of each vertex in the graph is evaluated to predict the most likely source vertex to make the greatest gain in value. The best source vertex is selected and the policy predicted to make the greatest gain is then simulated. If the result is similar to that predicted, then the resulting edges and vertices are added to the graph.

SHOT uses experimentally derived Lipschitz bounds to estimate the effect of a perturbation on the value and evolution at each vertex in the graph. AESOP exchanges the experimentally derived measure for an analytic representation of the perturbation analysis via a Taylor series approximation perturbation, which is presented in Chapter 6. This formulation enables AESOP to predict more precisely the effect of a perturbation on the value and evolution of the system. We utilize this information to create a composite of the policy edges in the graph. This representation enables us to predict the likelihood that the predicted outcome is representative of the actual outcome. We use this likelihood to inform both the learning methods used in the exploration and exploitation phases.

Both SHOT and AESOP attempt to achieve efficient optimization over the graph. The second method goes further by only updating the vertices and edges in the graph that are affected after the graph is expanded. By only updating the source of any edge if the value at the target vertex changes, a limited number of vertices is updated at each iteration. Moreover, this process more effectively back-propagates changes to the source vertex, speeding up convergence of the algorithm. The presentation and analysis of this method are provided

in Chapter 7.

Before presenting these techniques, background concepts are introduced and related research is explored in Chapter 2 and Chapter 3, respectively. In Chapter 4, hyperfiltering is provided to lay the necessary foundation for the proposed optimization technique. Then the optimization methods are presented: the offline multi-query POMDP optimization technique is developed in Chapter 5 and the sensitivity derived multi-edge technique is presented in Chapter 7. Motivation and analysis of the insensitivity of POMDP systems along with a methodology to chain forecasted evolutions is presented in Chapter 6. The dissertation concludes with some final remarks, comments, and a summary of the proposed future research in Chapter 8.

Chapter 2

PARTIALLY OBSERVED MARKOV DECISION PROCESSES

Relevant background material will be presented before hyperfiltering is formally introduced in Chapter 4. Hyperfiltering is a method for representing the effect of future uncertainty in POMDP systems. In particular, hyperfiltering is a means of propagating the complete representation of the uncertainty forward one future stage to the next. Uncertainty is present in many real-world robotic systems, from motion noise to sensor noise, and the presence of these uncertainties can plague the motion and sensing of many real-world robotic systems. By modeling the uncertainty and considering robotic systems as stochastic processes, it is possible to better design and simulate the evolution of these systems in an attempt to understand, alleviate, and/or anticipate the effect of noise in such systems. First, description of the specific class of system of interest is given in Section 2.1. This is followed by a discussion of the information space in Section 2.2.

2.1 Formulation

As models, POMDPs incorporate the possibility of incomplete and uncertain knowledge when mapping states to observations as well as uncertain knowledge as to evolution of the system from one stage to the next.

POMDPs include at least the following components:

- The state space representing the finite set of states of the world: $\mathcal{X}$.
- The finite set of control actions that can be executed: $\mathcal{U}$.

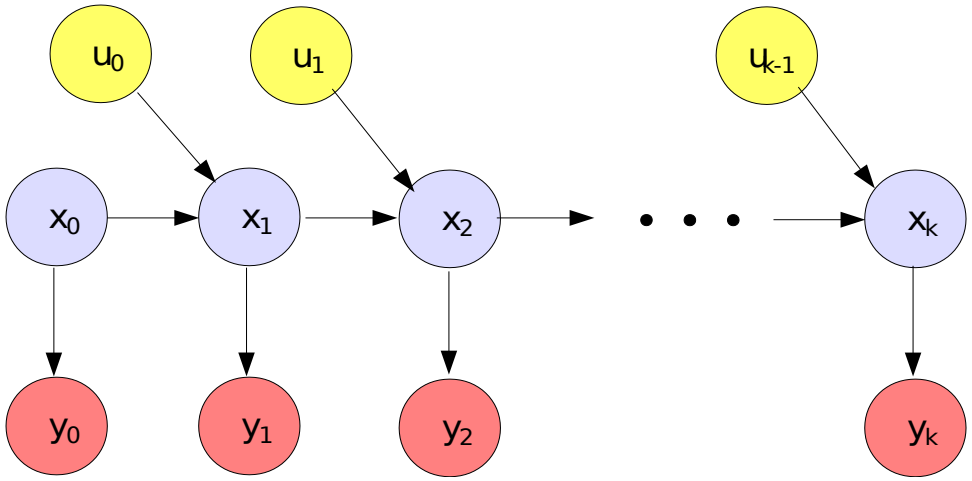


Figure 2.1: Dependency graph of a partially observed system defined by the observation probability function and the transition probability function

- The transition probability function: $p_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1}}$. This transition probability function represents the likelihood of the system being in one state and transferring into another state at stage k given the applied action at stage $k-1$.

- The set of all possible observations: $\mathcal{Y}$. The observations represent the information received by the sensors at each stage k.

- The observation probability function: $p_{\mathbf{y_k}|\mathbf{x}_k}$. The observation probability function represents the likelihood of a particular observation occurring given the system is in a specified state.

In addition, a POMDP may be specified with a cost function $c(\cdot)$, which defines the objective to be optimized for the POMDP.

2.2 The Information Space

As a system evolves from one stage to the next, it generates a sequence of random variables $\{\mathbf{x}_k\}_{k=0}^{K}$. Because the state is only indirectly observed through the observations, the system state is represented by a probability function conditioned on the set of observations and past

actions. A graphical representation of this process is illustrated in Figure 2.1.

Definition 2.1. *The information state, I_k, at stage k is the set of known information from the initial stage up to stage k. More precisely,*

$$I_k = \{y_0, u_o, y_1, u_1, y_2, u_2, y_3, \cdots, u_{k-1}, y_k\},$$

where $u_0, \cdots, u_{k-1}$ are the sequenced set of actions executed up to stage k, and $y_0, \ldots, y_k$ are the sequenced set of observations made. The total information known up to stage k is thus encapsulated in the information state.

The *information space* $\mathcal{I}_k$ is the set of all possible information states at stage k. The initial probability function over the state, as represented by $p_{\mathbf{x}_0}$, is assumed to be given and is often assumed that $p_{x_0}(\cdot|y_0) = p_{x_o}$.

At every stage k, $p_{\mathbf{x}_k|I_k}$ is the conditional probability function for the state given the information state I_k. The posterior probability function $p(x_{k+1}|I_{k+1})$ at stage $k+1$ is

$$p(x_{k+1}|I_{k+1}) = \frac{p(y_{k+1}|x_{k+1}) \sum_{x_k \in \mathcal{X}} p(x_{k+1}|x_k, u_k) p(x_k|I_k)}{\sum_{x_{k+1} \in \mathcal{X}} p(y_{k+1}|x_{k+1}) \sum_{x_k \in \mathcal{X}} p(x_{k+1}|x_k, u_k) p(x_k|I_k)}. \tag{2.1}$$

As a notational device, the concept of a *belief* is used to represent this conditional probability function.

Definition 2.2. *The belief b_k at stage k is*

$$b_k \triangleq p_{\mathbf{x}_k|I_k}. \tag{2.2}$$

This notation is pervasive in the literature (refer to [3]) and is adopted because, later in Chapter 4, the formulation of the hyperfilter will be made more clear by using this notation.

The *belief space* $\mathcal{P}_b$ is the set of all possible beliefs for a given system; in particular, it is the set of all probability mass functions defined over the state space $\mathcal{X}$. For discrete state

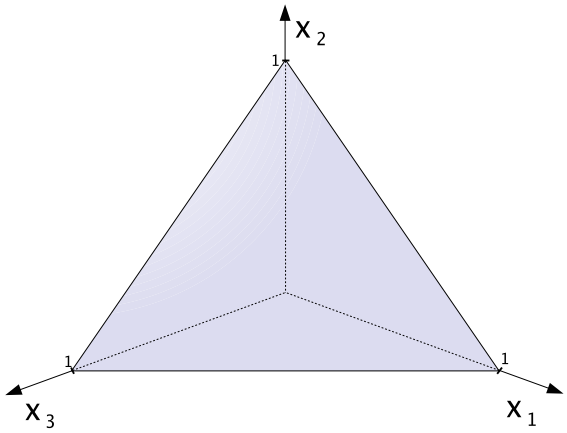


Figure 2.2: The 2-simplex

space systems, where the belief state has $|\mathcal{X}|$ states, the belief space can be represented by a $|\mathcal{X}| - 1$ dimensional simplex $\Delta^{|\mathcal{X}|-1}$.

Definition 2.3. *The n-simplex is the subset of* $\mathbb{R}^{n+1}$ *given by*

$$\Delta^n = \{(x_1, x_2, \ldots, x_{n+1}) \ : \ \sum_{i=1}^{n+1} x_i = 1 \ \textit{and} \ x_i \geq 0 \ \forall i\}.$$

The n-simplex is essentially a polytope that is an n-dimensional analogue of a triangle. The 2-simplex is illustrated in Figure 2.2.

The belief, being the conditional probability function over the state space given the information state, is a representation of the uncertainty of the state of the system. In fact, the belief is a sufficient statistic for the information vector, which was demonstrated in [13] (and subsequently in [14]).

A sufficient statistic, T satisfies the property that for the parameter θ, for which is an observation x is to be inferred from

$$p(x|T(x) = t, \theta) = p(x|T(x) = t).$$

In other words, the probability function over x is independent of θ when conditioned on T. Thus, all the information about the unknown parameter θ is captured in the sufficient

statistic T (refer to [15] for more on sufficient statistics).

The belief b_k is a sufficient statistic in that a control policy that depends on b_k will result in the same behavior as a control policy that depends on the information state for which the beliefs b_k were derived. Most optimization techniques rely directly on b_k because it encapsulates the information state and has a finite and constant dimensional representation, whereas the dimension of the information state grows at each stage. Because of the ability to sequentially evaluate the belief, b_k, for the current stage given b_{k-1} for the previous stage, the concept of the belief fits naturally into a dynamic programming (DP) framework (see [14] for an in-depth discussion about DP).

Filtering methods focus on estimating the current the belief, or an approximation of the belief, given the past set of observations and actions (e.g., [16–24]). By evaluating the belief at each stage as observations occur, filtering determines the behavior of the system as it evolves from the first stage to the current stage. However, filtering is applied as observations are received; filtering is not a method to predict the behavior into future stages for unknown future observations.

It is convenient to rely directly on b_k since it encapsulates the information state and has a finite and constant dimensional representation, whereas the dimension of the information state grows at each stage. Given the belief at stage k, the action applied at stage k and the observation at stage $k+1$, the belief b_{k+1} is given by the transition function

$$b_{k+1} = \phi(b_k, u_k, y_{k+1}). \tag{2.3}$$

In our case, this is a vector of dimension $|\mathcal{X}|$, and i^{th} component of b_{k+1} is given by

$$[\phi(b_k, u_k, y_{k+1})]_i = \frac{p(y_{k+1}|x^i) \sum_{x_k \in \mathcal{X}} p(x^i|x_k, u_k) b_k(x_k)}{\sum_{x_{k+1} \in \mathcal{X}} p(y_{k+1}|x_{k+1}) \sum_{x_k \in \mathcal{X}} p(x_{k+1}|x_k, u_k) b_k(x_k)}, \tag{2.4}$$

where x^i is the state corresponding to the i^{th} entry of the vector.

The transition function ϕ, given by (2.3), can be represented as a linear operator due to the finite cardinality of the state space. Define the matrix T_u for each $u \in \mathcal{U}$, such that $e_i^T T_u e_j = p_{\mathbf{x}'|\mathbf{x},u}(i \,|\, j, u)$, where e_i, e_j are elementary vectors for states i and j, respectively. Thus, the predicted belief $b_{k+1|k}$ from belief b_k under control u is

$$b_{k+1|k} = T_u b_k. \tag{2.5}$$

Similarly, define the matrix O_y, such that $O_y^{ii} = p_{\mathbf{y}|\mathbf{x}}(y \,|\, i)$, and $O_y^{ij} = 0$ for $i \neq j$. Then $O_y e_j = p_{\mathbf{y}|\mathbf{x}}(y \,|\, j)$. Thus, the updated belief b_{k+1} given $b_{k+1|k}$ for the observation y is

$$b_{k+1} = O_y b_{k+1|k}.$$

Notice that we can determine the probability of an observation y occurring given the belief b_k:

$$\begin{aligned} p(y_k|b_k) &= \sum_i p_{\mathbf{y}|\mathbf{x}}(y_k|i)_k b(i) \\ &= \mathbf{1}^T O_y b_k. \end{aligned} \tag{2.6}$$

Multiplying these matrices together yields the belief transition function

$$\phi(b_k, u_k, y_{k+1}) = \frac{O_{y_{k+1}} T_{u_k} b_k}{\mathbf{1}^T O_{y_{k+1}} T_{u_k} b_k} = \frac{\phi_{y_{k+1},u_k} b_k}{\eta_{y+1,u_k}^T b_k}, \tag{2.7}$$

where $\phi_{y_{k+1},u_k} = O_{y_{k+1}} T_{u_k}$ and $\eta_{y_{k+1},u_k}^T = \mathbf{1}^T \phi_{y_{k+1},u_k}$. The row vector η_{y+1,u_k}^T is a normalization factor that ensures $\phi(b_k, u_k, y_{k+1})$ maps to a belief on the simplex.

Often, in a POMDP or hidden Markov model (HMM) context, the objective is to compute, estimate, or approximate the current belief given current information state. Filtering methods, e.g. [16–24], focus on this problem. Refer to Appendix A for a detailed formulation of the filtering concept as well as a thorough review of approximate filtering techniques.

However, if we need to evaluate the cost of a set of control actions in future stages, we need to also consider the set of observations which are unknown a priori. The key distinction is that filtering is applied as observations are received, and is not a method to predict the behavior into future stages for unknown future observations.

Chapter 3

OPTIMIZATION METHODS FOR POMDPS

In this chapter we present an overview of reinforcement learning methods for POMDPs. First, we formulate the POMDP optimization problem in Section 3.1. We define the objective function and the primary methodologies of obtaining optimal policies. In Section 3.2 we catalog research relating to determining approximately optimal solutions to POMDPs. We frame the contributions of our work within the context of this related research.

3.1 Control Policies for POMDPs

Ultimately, the goal of much of robotics research is to engineer autonomous or nearly autonomous systems. Within the context robotics, control theory, and AI this concept of autonomy comes to fruition via the control policy $\pi(\cdot)$. A control policy maps input to control actions. If the control policy depends only on the stage, or $\pi : \{i\}_1^K \rightarrow \mathcal{U}$, it is referred to as an *open-loop policy* [14]. Alternately, a closed-loop/feedback policy takes information acquired as the system evolves, such as an estimate of the state, the previously applied actions, and so forth when determining which control action to apply. When the control policy takes as input the entire information state, or $\pi \ : \ \mathcal{I}_k \rightarrow \mathcal{U}$, it is referred to as an *information-feedback policy* [14].

The typical POMDP planning objective is to determine a policy that minimizes the

expected total cost[1] $V(b_0)$ of the system starting from the initial belief b_0,

$$V^*(b_0) = \min_{\pi \in \Pi} E\left[\sum_{k=0}^{K-1} c(b_k, \pi(b_k)) + c_K(b_K) \,|\, b_0\right],$$

where policy π in the class of feedback policies Π . This formulation is for finite-time horizon Bolza cost functions. Without any special structure, finding the optimal solution for a POMDP can be impractical, if not impossible, considering that the objective function can be a nonlinear function of the belief space. However, Smallwood and Sondik [25] established that finite-time horizon POMDPs have a special structure when the cost is the expected sum of rewards: the value function is piecewise linear and convex. POMDPs are still intractable even with this special structure.

We also consider discounted infinite horizon models where the total cost under π is

$$V^*(b_0) = \min_{\pi \in \Pi} E\left[\sum_{k=0}^{\infty} \gamma^k c(b_k, \pi(b_k))) \,|\, b_0\right] \tag{3.1}$$

and $\gamma \in (0, 1)$ is a discount factor. The derivation provided throughout this paper is applicable for such cost functions. The cost function $c : \mathcal{P}_b \times \mathcal{U} \to \mathbb{R}$ is belief dependent, enabling the evaluation risk-based cost functions. Risk-based cost functions encapsulate costs that are dependent on the uncertainty present in the system, which is ideal for localizing a robot at a goal position. The method presented in Chapter 7 is applicable to this type of cost function. However, the technique outlined in Chapter 5 considers a more general cost function that is hyperbelief dependent, which can also minimize this objective or other similarly motivated objectives.

The two canonical methods for finding optimal policies are value iteration, a dynamic programming (DP) approach, and policy iteration. Both of these techniques evaluate over the entire set of possibilities albeit in different spaces. Value iteration operates backwards

[1] Many POMDP planners are concerned with maximizing reward, but these two formulations are entirely equivalent.

from the terminal stage to the initial stage. At each stage, value iteration finds the optimal value from the current stage to the next stage and thereby the optimal policy for the current stage to the next stage. This repeats until the initial stage is reached and the optimal solution is found. Policy iteration, on the other hand, operates by starting with an initial policy. Then at each iteration the method searches for any change to the policy from the previous iteration that improves the performance. In either case, whether using policy iteration or value iteration, finding the exact solution is intractable, with a best known computational time complexity that is exponential in both the time horizon K (the number of stages into the future the process ends) and the number of observations $|\mathcal{Y}|$ [2]. However, finding the optimal policy for partially observed systems is desired for many real-world problems. Thus, many researchers have focused on finding efficient methods to solve POMDP models.

Because of strict discounting, it is possible to establish a bound on the approximation error for a given time horizon [2]. Many techniques have been developed to find more efficient exact solutions to the POMDP problem, such as [2, 26–31]. However, since each of these techniques is intractable, many researchers have turned their focus to finding approximate solutions. A taxonomy of approximation approaches is enumerated in section Section 3.2.

3.2 Approximation Methods

Finding the exact, optimal policy for a POMDP system is intractable [32], with a best known computational time complexity that is exponential in the time horizon K, i.e. the number of stages into the future that optimization considers. However, finding policies for partially observed systems is desired for many real-world problems, so researchers have focused on finding efficient approximation methods to solve POMDP models.

A variety of approaches exist that search for approximate solutions to POMDP systems. One popular approach is to reduce the dimensionality of the belief space. Foremost among techniques explored in the literature are compression [33–35], projection [36], and decompo-

sition [37,38] methods. Each of these techniques takes a unique approach to achieving the same goal: a reduced representation of the belief space to use for planning. By reducing the dimensionality of the belief space, one of the curses of dimensionality of POMDPs is abated.

Another popular approach to solving POMDPs focuses on approximating the value function. These methods typically sample points in belief space and use the Bellman equation to compute the value function over this subset of the belief space. Thrun in [5] proposed sampling based on Monte Carlo integration. Soon after, this method was augmented by retaining a linear function around each sampled belief (referred to as the α-vector) in [6]. Smith and Simmons [7,8] realized that the set of beliefs reached in a small number of stages typically generate the majority of the cost in discounted infinite-horizon problems, and thus biased sampling beliefs accordingly. However, in [11], Kurniawati et al. built on the work of Smith and Simmons by heuristically choosing beliefs that are reachable after many stages, selected for their potential impact on the value function. While these approaches have been successful in many aspects, they generate an approximation of the value function and not a representation of the structure of the POMDP system.

Our technique is inspired, in one part, by many of the sampling-based techniques developed in the robotics community. Sampling-based methods have become one of the dominant methods for planning in the robotics community (refer to [39, Ch. 7] for an overview and survey of sampling-based techniques). As described in [40], probabilistic sampling-based roadmap methods generate a series of samples in the configuration space and simple planners are used to link the samples together. In this way, a graph of the samples is created that, in some cases, eventually becomes a roadmap. The majority of these methods focus on finding feasible, but not necessarily optimal solutions. However, Kim et al. [41] have developed a technique to determine the optimal solution over the graph.

The concept of stochastic uncertainty in the process model was first introduced into sampling methods by Apaydin et al. [42]. This method creates a discrete approximation of a continuous space by creating a graph in the configuration space, augmented with likelihoods

for the graph edges. This simplified planning using this approximation of a continuous Markov decision process. While Apaydin et al. focused on determining feasible solutions, Alterovitz et al. [43] added the concept of optimizing over the graph with the Stochastic Motion Roadmap (SRM). SRM samples a set of points in the configuration space to generate the graph. To generate the probability of traversing edges, it samples the process model to generate a random set of resulting states for a given action.

Another method to build a sampling-based abstraction for partially observed systems was developed by Prentice and Roy in [44,45]. Prentice and Roy generate a sampling-based approach for nonlinear Gaussian systems, where beliefs are approximated using the extended Kalman filter (refer to [17]). In this approach, a set of mean samples is generated corresponding to points in the workspace. Next, a traditional probabilistic roadmap is used to generate a roadmap of the system assuming that the certainty equivalent controller is capable of placing the expected value of the robot's configuration at any point in the configuration space. They generate the transfer function of each edge and compute covariances along the most likely sample path. Using a standard breadth first search, the covariance is generated for each walk through the graph and is used to find an optimal graph path. Like the work by Prentice and Roy, Platt et al. in [46] develop a motion planning method that assumes Gaussian noise and relies on the maximum likelihood observation for planning. However, Platt's approach relies on linear quadratic regulation and a re-planning process to drive the agent to a goal belief state.

Simultaneously with our work, other methods such as that presented by Kurniawati et al. in [47] seek to plan specifically for POMDPs requiring long time horizons. Their approach samples points in the state-space as milestones which are used reduce the planning horizon. Another approach presented in [48] exploits domain knowledge to generate local-policies (or options). Unlike other approaches, the local policies are feed-back and not open-loop. He et al. [49] explore a macro-action approach, and establish ϵ-optimality of their technique, which outperforms SARSOP [11]–one of the leading state-of-the-art approaches. The multi-

resolution compression method developed in [37] achieves abstraction of the problem domain by grouping together similar portions of the state space to reduce the complexity of planning.

The other core inspiration for this work arose in the artificial intelligence community in parallel with the development of roadmap-based methods in robotics. Artificial intelligence researchers have developed the notion of temporal abstraction for optimizing options (policies) for Markov decision processes [50, 51]. Temporal abstraction is the process of representing actions as policies or multi-stage actions so that fewer stages need be optimized to determine effective policies. Some of the first work on temporal abstraction was performed by Sutton et al. [50] by formulating the MDP as a Semi-MDP process. Building on this work, several researchers applied hierarchical representations to POMDPs. Either the structure of the hierarchical representation is known a priori or it must be learned online. Planning with a predefined hierarchy of tasks has been explored in [52] and [53]. Other methods, such as [54] and [55], attempt to discover a hierarchy of tasks to use for planning.

Whether to learn a hierarchical representation or to optimize the system, in each of these approaches a finite state controller (FSC) representation is applied, where states in the FSC are abstract – only representative of transitioning from one state to the next. The transitions themselves are triggered by observations. Thus, the representation is essentially evolving in the observation space. The drawback to such a representation, which also motivates the necessity of a hierarchical approach, is that the number of possible observation strings increases exponentially as the time horizon increases. Our approach builds hierarchical structural representation in the hyperbelief space of the system. We simultaneously learn the structure and optimize what has been learned.

The core difference between these methods and our work is the space in which the algorithms operate. While the above methods operate in the configuration (or equivalently state) space, which is augmented in various ways to handle uncertainty, our method operates in the hyperbelief space. This means we explicitly consider multiple observations and sample paths in the evolution of the system. Since the hyperbelief space has not been extensively

explored in the literature, in Section 4.1 we discuss the evolution of the system in this space, and practical approaches for approximating trajectories.

Chapter 4

HYPERFILTERING

In this chapter we will present a method for the forward sequential simulation of POMDP systems into future stages (Section 4.2). A preliminary version of this technique first appeared in [56]. We refer to this process as hyperfiltering, and we present a specific computational approach to hyperfiltering that we call hyper-particle filtering in Section 4.3. We will use hyperfiltering in the forward-based, hierarchical method developed in Chapter 5. Before doing so, we will explore some insights into the evolution of POMDPs provided by hyperfiltering to garner an understanding into the behavior of such systems in Section 4.4.

4.1 Hyperbelief Space, Evolution, and Approximations

The sample path evolution of a POMDP can be completely captured by a single trajectory through the belief space. When the system is in operation it follows a sample path, but the particular sample path will only be known a posteriori. This is because the next observation and subsequent controls are random variables. The prediction step, $T_{u_k} b_k$, is deterministic in the belief space since u_k is known, but the update step will be uncertain, due the unknown future observation y_{k+1}. Therefore, the next stage belief is also a random variable: $\mathbf{b}_{k+1} = \phi(\mathbf{b}_k, u_k, \mathbf{y}_{k+1})$. If a closed-loop policy is used to generate actions, subsequent controls will become random variables that that depend on $\mathbf{b}_{k+1}$. To analyze the complete behavior of a POMDP in the future, e.g. to evaluate the possible effect of a control policy, we must consider all sample paths that have nonzero probability.

To characterize this complete behavior, we analyze the behavior of the system in the

space of probability functions over the belief space, referred to as the *hyperbelief space*. In the hyperbelief space, the system evolves deterministically, thus eliminating (in some sense) the explicit consideration of uncertainty during the planning process, at the expense of a much higher representational cost.

The probability measure that captures the probability distribution over the belief space is referred to as a *hyperbelief*. Specifically, the hyperbelief β_k at stage k is a functional $\beta_k : \mathcal{P}_b \to \mathbb{R}_+$ such that $\int_{\mathcal{P}_b} \beta_k(b_k) db_k = 1$. The hyperbelief space $\mathcal{P}_\beta$ is the space of possible hyperbeliefs, defined as the set of all probability measures on $\mathbb{B}(\mathcal{P}_b)$, the Borel σ-algebra defined over the belief space $\mathcal{P}_b$. For discrete state space POMDP systems, the belief space $\mathcal{P}_b$ is represented as the simplex $\Delta^{|\mathcal{X}|-1}$. The Borel σ-algebra $\mathbb{B}(\Delta^{|\mathcal{X}|-1})$ exists and, thus, the hyperbelief space $\mathcal{P}_\beta$ is well defined.

We adopt the convention that β_k, the hyperbelief at stage k, is defined as a conditional probability density, given an initial hyperbelief β_0 and an information-feedback control policy π:

$$\beta_k \triangleq p_{\mathbf{b}_k|\beta_0,\pi}(\cdot|\beta_0, \pi), \tag{4.1}$$

where $\beta_k(b_k)$ is the conditional probability density value assigned to b_k given β_0 and policy π.

The rest of this section is devoted to discussing the sequential evolution and approximation of β_k. In Section 4.2, we present the hyperbelief transition function and demonstrate that compositions of this operator are well-defined. Thus, we are able to sequentially compute the next stage hyperbelief and evaluate probabilistic system behavior stage-wise. We refer to this procedure as *hyperfiltering*.

Of course, we cannot compute this quantity exactly without succumbing to the same exponential explosion of support points in the belief space from which the general POMDP planning problem suffers. However, moving to a hyperbelief-based representation allows us to use standard, principled approximation techniques to approximate the POMDP's evolu-

tion. In Section 4.3, we present a technique called *hyper-particle filtering* that sequentially approximates the hyperbelief.

4.2 Hyperbelief Evolution

In the hyperbelief space, the transition from β_k to β_{k+1} is deterministic. For a given policy $\pi \in \Pi$, the hyperbelief transition function, $\Phi : \mathcal{P}_\beta \times \Pi \to \mathcal{P}_\beta$, maps the hyperbelief at stage k to a hyperbelief at stage $k+1$, with $\beta_{k+1} = \Phi(\beta_k, \pi)$. Since β_{k+1} represents a probability density function (pdf) over the simplex $\mathcal{P}_b$, we use $\Phi(\beta_k, \pi)(b_{k+1})$ to denote the pdf value assigned to the specific belief b_{k+1} by the hyperbelief β_{k+1}.

Theorem 4.1. *The hyperbelief transition function, $\Phi : \mathcal{P}_\beta \times \Pi \to \mathcal{P}_\beta$ that maps the hyperbelief at stage k to a hyperbelief at stage $k+1$, with $\beta_{k+1} = \Phi(\beta_k, \pi)$ is given by*

$$\Phi(\beta_k, \pi)(b_{k+1}) = \int_{\mathcal{P}_b} \left(\sum_{y_{k+1} \in Y^*} \sum_{x_{k+1}} p(y_{k+1}|x_{k+1}, \pi(b_k)) \sum_{x_k} p(x_{k+1}|x_k, \pi(b_k)) b_k(x_k) \right) \beta_k(b_k) db_k \tag{4.2}$$

in which $Y^* = \{y_{k+1} \mid b_{k+1} = \phi(b_k, u_k, y_{k+1})\}$.

The transition equation (4.2) can be derived by applying the definition of the hyperbelief, and marginalizing appropriately. For a proof of Theorem 4.1, refer to Section 4.5. The hyperbelief transition equation (4.2) can be expressed more compactly as

$$\Phi(\beta_k, \pi)(b_{k+1}) = p_{\mathbf{b}_{k+1}|\beta_0,\pi}(b_{k+1}|\beta_0, \pi) \ = \int_{\mathcal{P}_b} p(b_{k+1}|b_k, \pi) \beta_k(b_k) db_k, \tag{4.3}$$

where $p(b_{k+1}|b_k, \pi)$ is known as the belief transition function [3].

Defining Π to be the set of all information-feedback policies that depend on the state and defining $\mathcal{M}(\mathcal{P}_b)$ as the set of all bounded $\mathbb{B}(\mathcal{P}_b)$-measurable functions defined over $\mathcal{P}_b$, it is possible to establish a sequential formulation of the hyperbelief.

Theorem 4.2. *For a system modeled as a POMDP with a discrete state space and with a given control policy $\pi \in \Pi$, the hyperbelief $\beta_k \in \mathcal{P}_\beta$ at stage k given the initial hyperbelief $\beta_0 \in \mathcal{P}_\beta$ can be evaluated via the recursive application of the belief transition probability function from stage k to the initial stage. This holds if the belief transition function is defined such that $p_{b_{k+1}|b_k,u_k}(\cdot|b_k,u_k) \in \mathcal{P}_\beta$ for all $b_k \in \mathcal{P}_b$, $u_k \in \mathcal{U}$ and $p(\mathbf{b}_{k+1}|\cdot,u_k) \in \mathcal{M}(\mathcal{P}_b)$ (is measurable over the belief space) for all $b_{k+1} \in \mathcal{P}_b$, $u_k \in \mathcal{U}$.*

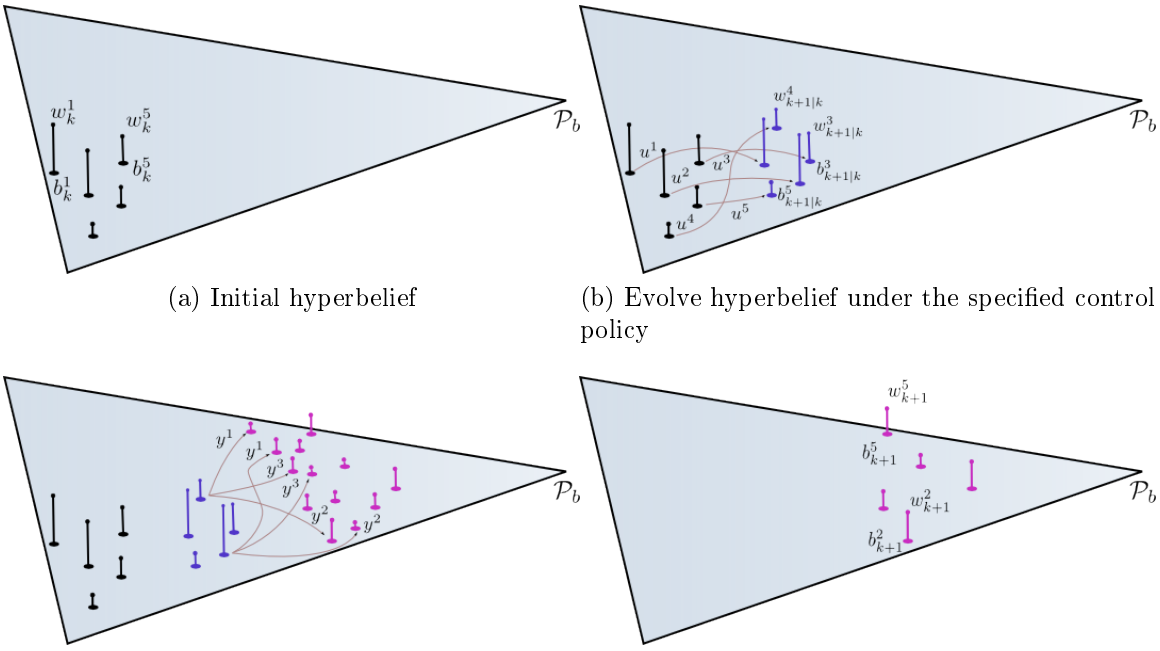
The proof follows by induction on the application of the belief transition function. Also, by elementary properties of integrable functions, the hyperbelief can be evaluated and is a probability function defined over the belief space. Refer to Section 4.5 for the full proof. As a result, the hyperbelief is well-defined over an arbitrary number of stages and the hyperfiltering procedure can be carried out indefinitely.

4.3 Hyper-Particle Filtering

Because of the highly nonlinear behavior of the hyperbelief transition function and the exponential growth in the number of feasible beliefs from one stage to the next, we approximate the hyperbelief transition function using a method analogous to particle filtering (e.g, [57–69]). As stated earlier, this process is referred to as hyper-particle filtering. A hyperbelief β_k is represented by a set $\tilde{\beta}_k$ of hyper-particles, each of which includes a sample belief b_k^i and an associated nonnegative scalar weight, w_k^i. Analogous to particles, each hyper-particle represents a point in the belief space at which a nonzero probability mass is concentrated. Thus, $\tilde{\beta}_k$ provides a discrete approximation to the hyper-belief β_k

$$\tilde{\beta}_k = \left\{(w_k^i, b_k^i)\right\}_{i=1}^R . \tag{4.4}$$

The hyper-particle filtering algorithm takes as input a set $\tilde{\beta}_k$ of hyper-particles and a control policy π, and outputs a new set of hyper-particles $\tilde{\beta}_{k+1}$, which is obtained using



(a) Initial hyperbelief

(b) Evolve hyperbelief under the specified control policy

(c) Generate Posterior for the observations

(d) Resample updated hyper-particle set

Figure 4.1: Hyper-particle filtering process

the hyperbelief transition probability function given by (4.2). The algorithm works by sampling a set of beliefs $\{b_{k+1}^l\}$ using particle filtering methods, and then adjusting the weights associated to these beliefs to generate the approximated hyperbelief $\tilde{\beta}_{k+1}$. This process is illustrated in Figure 4.1.

Directly sampling from the (4.2) is not practical. Instead, we use the transition function T_u to first generate sample beliefs $b_{k+1|k}$ (prediction step). From this set of predicted beliefs, we then sample observations, propagate the set of beliefs through the observation model $O_{y,u}$, and adjust the weights assigned to each hyper-particle (update step).

These two steps are implemented as follows: At stage k, a traditional particle filtering algorithm is applied to each hyper-particle $b_k^i = \{w_k^i, b_k^i\} \in \tilde{\beta}_k$, with R hyper-particles, to generate a new belief $b_{k+1|k}^j = T_{u_k} b_k$. Each new hyper-particle is assigned a weight $w_{k+1|k}^j = w_k^i$. Once the predicted set of R hyper-particles is generated, for each belief $b_{k+1|k}^j \in \tilde{\beta}_{k+1|k}$, we generate at most S sample beliefs. This is done by randomly sampling from some importance sampling function $q_{b_{k+1}|b_{k+1|k}}$ to create a new set of hyper-particle

samples: $\tilde{\beta}_{k+1} = \{w_{k+1}^l, b_{k+1}^l\}_{l=1}^{RS}$ at stage $k+1$. The importance sampling function can be any function that generates belief samples, such as randomly sampling an observation and the selecting the updated belief as the random sample.

In addition to updating each predicted belief, we must also determine the weight of the updated belief. The weight of the predicted belief must be distributed to the set of updated beliefs. Ideally, we would sample each observation for each belief to generate the updated hyper-particle set. The weight of each updated belief is proportional to the product of the previous weight and the probability of the new belief for such a technique. Unfortunately, this causes an exponential growth in the number of hyper-particles as the number of stages increases. To address this we choose to use an importance sampling function to generate a subset of the possible updated beliefs instead. Unfortunately, unwanted properties may be introduced as an artifact of the importance sampling function, which may create a bias in set of samples generated. This bias needs to be considered when evaluating the weight of the sampled beliefs, otherwise the result can quickly become erroneous.

By dividing $p_{\mathbf{b}_{k+1}|\mathbf{b}_{k+1|k}}$ by $q_{\mathbf{b}_{k+1}|\mathbf{b}_{k+1|k}}$, we can attenuate the bias. As observed when performing Monte Carlo integration, for some function $c(\cdot)$,

$$E[c(b)] = \sum_{b \in \mathcal{P}_b} c(b)p(b) = \sum_{b \in \mathcal{P}_b} c(b)\frac{p(b)}{q(b)}q(b).$$

The expectation of a random vector with a probability function $p(b)$ can be represented as the expectation of another random vector with the probability function $q(b)$ by weighting $c(b)$ by the ratio of $p(b)$ and $q(b)$. This reduces or eliminates the the bias of the importance sampling on the expected value.

Considering the effect of the bias, we can update the weight w_{k+1}^l for each hyper-particle b_{k+1}^l by

$$p(b_{k+1}^l|\tilde{\beta}_k, \pi) \approx \eta_{k+1}\frac{p(b_{k+1}^l|b_{k+1|k}^j)}{q(b_{k+1}^l|b_{k+1|k}^j)}w_{k+1|k}^j = w_{k+1}^l, \tag{4.5}$$

where η_{k+1} is a normalizing constant. This follows from the form of the exact update in (4.3):

$$p(b^l_{k+1}|\tilde{\beta}_k, \pi) = \int_{b_k \in \mathcal{P}_b} p(b^l_{k+1}|b_k, \pi(b_k)) p(b_k|\tilde{\beta}_k, \pi) db_k \tag{4.6}$$

$$= \int_{b_k \in \mathcal{P}_b} p(b^l_{k+1}|T_{\pi(b_k)} b_k) \tilde{\beta}_k(b_k) db_k, \tag{4.7}$$

which we obtain by marginalizing $p(b^l_{k+1}|\beta_k, \pi)$ on b_k. We obtain the result in (4.7) because $p(b_{k+1}|b_k, u_k) = p(b_{k+1}|T_{u_k} b_k)$.

At each stage, the hyper-particle set $\tilde{\beta}_k$ comprises a finite set of samples. Thus, (4.7) reduces to the summation over all the belief samples in $\tilde{\beta}_k$, such that

$$\int_{b_k \in \mathcal{P}_b} p(b^l_{k+1}|T_{\pi(b_k)} b_k) \tilde{\beta}_k(b_k) db_k = \sum_{i=1}^{R} p(b^l_{k+1}|T_{\pi(b^i_k)} b^i_k) w^i_k. \tag{4.8}$$

The set $\tilde{\beta}_{k+1|k}$ is generated to approximate the output of $T_{\pi(b^i_k)} b^i_k$ for each sample in $\tilde{\beta}_k$, so substituting $\tilde{\beta}_{k+1|k}$ into (4.8), where $w^i_k = w^j_{k+1|k}$, gives

$$\sum_{i=1}^{R} p(b^l_{k+1}|T_{\pi(b^i_k)} b^i_k) w^i_k \approx \sum_{j=1}^{R} p(b^l_{k+1}|b_{k+1|k}) w^j_{k+1|k}. \tag{4.9}$$

Finally, by replacing $p(b^l_{k+1}|b_{k+1|k})$ by $\frac{p(b^l_{k+1}|b^j_{k+1|k})}{q(b^l_{k+1}|b^j_{k+1|k})}$ to reduce the bias, we arrive at (4.5). The normalizing constant η_{k+1} is given by $\frac{1}{\eta_{k+1}} = \sum_{l=1}^{RT} w^l_{k+1}$.

Unfortunately, sampling more than one observation per hyper-particle causes an exponential growth in the number of hyper-particles as the number of stages increases. We therefore resample and normalize the set of hyper-particles to ensure that there are at most S hyper-particles and the weights of those hyper-particles sum to one. The algorithm describing computation of one entire stage is described in Algorithm 1. The algorithm for *HPF_sample* is given in Algorithm 2. The resampling algorithm performs sampling without replacement by generating a value uniformly and then raising the sampled value by the

weight of each hyper-particle sample. The lowest values are then selected to represent the new hyperbelief. The key to this particular resampling method is that the hyper-particles are randomly sampled according to their probability of occurring.

Algorithm 1: Hyper-particle filter

Input: $\tilde{\beta}_k$, where $\tilde{\beta}_k = \{w_k^i, b_k^i\}_{i=1}^R$: hyper-particle set,
n: number of output hyper-particles samples
T: number of intermediate hyper-particle samples,
π: a control policy
Output: $\tilde{\beta}_{k+1|k} = \{\bar{w}_{k+1|k}^i, \bar{b}_{k+1|k}^i\}_{i=1}^n$

$l \leftarrow 1$;
for $j \leftarrow 1 : R$ **do**
  predict $\hat{b}_{k+1|k}^j$ using the particle filtering prediction ;
  $\hat{w}^j \leftarrow w^j$;
  for $t \leftarrow 1 : S$ **do**
    sample $\hat{b}_{k+1}^l$ from the distribution $q(\cdot|\hat{b}^j)$;
    $l \leftarrow l + 1$;
for $l \leftarrow 1 : RS$ **do**
  $\hat{w}_{k+1}^l \leftarrow \frac{p(\hat{b}_{k+1}^l|\hat{b}_{k+1}^j)}{q(\hat{b}_{k+1}^l|\hat{b}_{k+1}^j)} \hat{w}_{k+1|k}^j$;
$w_{tot} \leftarrow \sum_{l-1}^{RS} \hat{w}_{k+1}^l$;
normalize each $\hat{w}_{k+1}^l$ by w_{tot} ;
$\tilde{\beta}_{k+1} \leftarrow \{\hat{w}_{k+1}^l, \hat{b}_{k+1}^l\}_{l=1}^{RS}$;
$\tilde{\beta}_{k+1} \leftarrow$ HPFsample($\tilde{\beta}_{k+1}$,n) ;
return $\tilde{\beta}_{k+1}$

It is important to note that the hyper-particle filtering procedure is agnostic to belief representation. Thus, it may also be used as a two-tiered approximation approach where the exact belief b_k is replaced by a parameterized or particle-filtered approximation. Furthermore, transition and sampling functions may also be approximated. Each additional approximation added to the algorithm will typically decrease the quality of the overall method.

Algorithm 2: HPFsample (Hyper-particle filter resampling)

Input: $\tilde{\beta}$: where $\tilde{\beta} = \{w^i, \hat{b}^i\}_{i=1}^{RS}$ a hyper-particle set,
n: number of output hyper-particles samples
Output: $\tilde{\beta}_{upd} = \{w^i, \hat{b}^i\}_{i=1}^{R}$

create a set s taking a value v and a hyperparticle $\{w, \hat{b}\}$;
for $j = 1 : RS$ **do**
    $r \leftarrow$ uniform random value between $(0, 1]$;
    $v^j \leftarrow r^{w^j}$;
    insert $\{v^j, \{w^j, \hat{b}^j\}\}$ into s ;
sort s on v ;
$\tilde{\beta}_{upd} \leftarrow R$ lowest valued hyper-particles in s ;
normalize $\tilde{\beta}_{upd}$;
return $\tilde{\beta}_{upd}$

4.4 Discussion

At the most basic level, hyper-particle filtering is similar to sample path simulation. However, with the additional ability to selectively sample and resample future beliefs, a more efficient representation of the future evolution may be obtained. As an example, under sample path simulation, if a sampled path is extremely unlikely–a result of sampling unlikely observations–resources are wasted computing the future evolution even further. With hyper-particle filtering, resampling occurs at each stage and the set of possible beliefs are selectively sampled to (typically) avoid devoting resources to unlikely paths. Moreover, hyper-particle filtering enables sampling and resampling based on the forecasted hyperbelief at each stage, whereas sample path methods rely on a single belief. Because of this, many of the benefits, but also the drawbacks, of particle filtering are transfered to hyper-particle filtering [57]. A particle representation enables us to forecast the future evolution using a compressed representation. As with particle filtering, however, care must be taken to avoid particle impoverishment. The particle representation will diverge from the actual hyperbelief if this occurs.

Insight into the impetus behind hyper-particle filtering can be realized through a simple example. Consider a POMDP with three states, three observations, and two actions using

policy π, where the objective is to localize the system in 9 stages. Localization in this context is defined as minimizing the uncertainty in the system's position. The evolution of this system is illustrated in Figure 4.2 by projecting the support points of the hyperbelief onto the belief space. Thus, the simplex represents the belief space. Each line segment in the plot corresponds to a belief, with the probability of that belief being proportional to the height of the line segment. Starting at stage 4 in Figure 4.2a, there are a total of 27 beliefs with nonzero probability. In Figure 4.2b, there are 81 feasible beliefs for stage 5, and at stage 6 there are 244 feasible beliefs as shown in Figure 4.2c. Although, the number of feasible beliefs grows exponentially in the number of stages, this example demonstrates that only a few of the beliefs that have appreciable probability, and that support in the belief space tends to cluster. The hyper-particles we sample will tend to be the support points with higher likelihood (refer to Section 4.3), and represent beliefs from the different clusters. This enables a reasonable representation of the evolution of the system to be maintained while not dealing with the exponential explosion of support points.

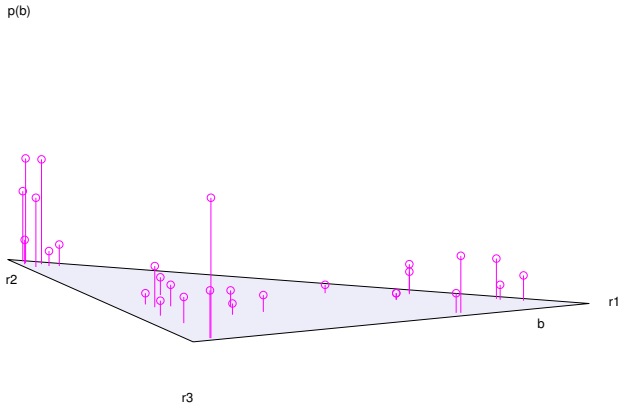
The computational complexity of hyper-particle filtering varies, depending on the choice of performance parameters (e.g., the number of particles and hyper-particles), from $O(KR(QL+M))$ to $O(R^K)$, where K is the time horizon, R is the number of hyper-particles, Q is the number of particles approximating the belief (via particle filtering), $O(L)$ is the computational time complexity of the particle filtering sampling, and $O(M)$ is the computational complexity of performing the hyper-particle sampling.

4.5 Hyperfiltering Proofs

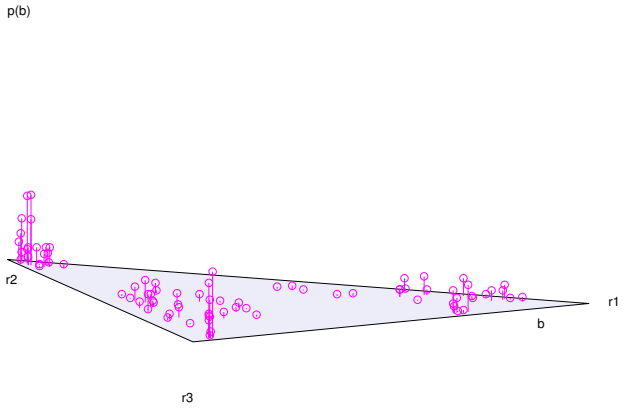
The following proof validates Theorem 4.1.

Proof. [Theorem 4.1] By definition, the hyperbelief transfer function Φ evaluated at belief b_{k+1} is given by

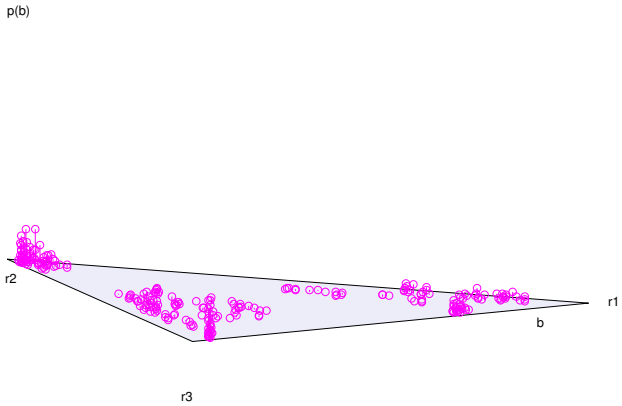
$$\Phi(\beta_k, \pi)(b_{k+1}) = p_{\mathbf{b}_{k+1}|\beta_0,\pi}(b_{k+1}|\beta_0, \pi).$$



(a) Stage k=4



(b) Stage k=5



(c) Stage k=6

Figure 4.2: Evolution of the hyperbelief for a simple example

To obtain the hyperbelief value at stage $k+1$, we marginalize over the stage k beliefs

$$p_{\mathbf{b}_{k+1}|\beta_0,\pi}(b_{k+1}|\beta_0,\pi) = \int_{\mathcal{P}_b} p(b_{k+1}|b_k,\beta_0,\pi)p(b_k|\beta_0,\pi)db_k \tag{4.10}$$

$$= \int_{\mathcal{P}_b} p(b_{k+1}|b_k,\pi)\beta_k(b_k)db_k. \tag{4.11}$$

We obtain (4.11) from the fact that the belief at stage $k+1$ is conditionally independent of the initial hyperbelief β_0 given the belief at stage k, and by applying the definition $\beta_k(b_k) = p(b_k|\beta_0,\pi)$. This provides a recursive formulation for β_{k+1}, given in terms of β_k and $p(b_{k+1}|b_k,\pi)$. Thus, we need only find an expression for $p(b_{k+1}|b_k,\pi)$. The term $p(b_{k+1}|b_k,\pi(b_k))$ is typically referred to as the belief transition probability function. Many approaches in the POMDP optimization literature either explicitly or implicitly use the belief transition probability function (refer to [3]). The belief transition probability function can be simplified using a marginalization integral over the observation space at stage $k+1$ (i.e., the set of possible observations at the future stage $k+1$):

$$p(b_{k+1}|b_k,\pi) = \int_{\mathcal{Y}} p(b_{k+1}|y_{k+1},b_k,\pi)p(y_{k+1}|b_k,\pi)dy_{k+1}. \tag{4.12}$$

Consider the term $p(y_{k+1}|b_k,\pi)$ in (4.12). By marginalizing first over the state at stage $k+1$, we obtain

$$p(y_{k+1}|b_k,\pi) = \sum_{x_{k+1}} p(y_{k+1}|x_{k+1},b_k,\pi(b_k))p(x_{k+1}|b_k,\pi(b_k)) \tag{4.13}$$

$$= \sum_{x_{k+1}} p(y_{k+1}|x_{k+1}) \sum_{x_k} p(x_{k+1}|x_k,\pi(b_k))b_k(x_k). \tag{4.14}$$

We obtain (4.14) by using the fact that the observation y_{k+1} is conditionally independent of past beliefs and actions given the state x_{k+1}, by marginalizing the second term with respect to the state at stage k, and applying the definition of $b_k(x_k)$.

For the remaining term, $p(b_{k+1}|y_{k+1},b_k,\pi)$, using (2.3), we have $b_{k+1} = \phi(b_k,u_k,y_{k+1})$ for

a given b_k, $\pi(b_k) = u_k$, and y_{k+1} Therefore,

$$p(b_{k+1}|y_{k+1}, b_k, u_k) = \begin{cases} 1 & \text{if } b_{k+1} = \phi(b_k, u_k, y_{k+1}) \\ 0 & \text{otherwise.} \end{cases} \tag{4.15}$$

Because of this, the integral (4.12) need be evaluated only at the specific values y_{k+1} that satisfy (2.3) for the given b_k and u_k. We define this set as $Y^* = \{y_{k+1} \mid b_{k+1} = \phi(b_k, u_k, y_{k+1})\}$. Combining the above results yields (4.2). □

The following proof establishes Theorem 4.2.

Proof. [Theorem 4.2] The proof follows by induction on the application of the belief transition probability function. First note that the belief transition function, as a function of a random belief and a random observation, is Markovian; the future probability of a belief depends only on the previous belief, and the policy, which depends on the previous belief. At the second stage, $k = 2$, the hyperbelief can be formulated by marginalizing the hyperbelief β_2 on b_1 and substituting $u_1 = \pi(b_1)$ to obtain

$$\begin{aligned} \beta_2(b_2) = p(b_2|\beta_1, \pi) &= \int_{b_1 \in \mathcal{P}_b} p(b_2|b_1, \beta_1, \pi(b_1)) p(b_1|\beta_1) db_1 \\ &= \int_{b_1 \in \mathcal{P}_b} p(b_2|b_1, \pi(b_1)) \beta_1(b_1) db_1. \end{aligned} \tag{4.16}$$

Because the state space is finite, the belief space is represented as a finite-dimensional simplex. The integration therefore is performed over the simplex. In the second equation, $p(b_2|b_1, \beta_1, \pi(b_1))$ is conditionally independent of β_1 when conditioned on b_1. Because $p(b_1|\beta_1)$ is the probability of b_1 conditioned on the hyperbelief β_1, by definition it is equivalent to $\beta_1(b_1)$. As β_1 is the initial hyperbelief, it is assumed to be given. The result is $p(b_2|b_1, \pi(b_1))$, which is just the belief transition probability function. The assumption that $\beta_1 \in \mathcal{P}_\beta$ implies that β_1 is an integrable function. Moreover, by the assumption that $p(b_{k+1}|\cdot, u_k) \in \mathcal{M}(\mathcal{P}_b)$ for all $b_{k+1} \in \mathcal{P}_b$, $u_k \in \mathcal{U}$, implies (4.16) is integrable as the product of two integrable func-

tions is also integrable. Moreover, because $p_{b_{k+1}|b_k,u_k}(\cdot|\ b_k, u_k) \in \mathcal{P}_\beta$ for all $b_k \in \mathcal{P}_b$, $u_k \in \mathcal{U}$ is integrable and satisfies the properties to be a probability measure; then, by definition of $\mathcal{P}_\beta$, the hyperbelief β_2 belongs to $\mathcal{P}_\beta$.

Assuming that $\beta_{k-1} \in \mathcal{P}_\beta$ and that β_{k-1} can be evaluated by integrating over the belief transition probability function the hyperbelief at stage $k-2$, the hyperbelief β_k at stage k is expressed as

$$\begin{aligned} \beta_k(b_k) = p(b_k|\beta_1, \pi) &= \int_{b_{k-1}\in\mathcal{P}_b} p(b_k|b_{k-1}, \beta_1, \pi(b_{k-1}))p(b_{k-1}|\beta_1, \pi)db_{k-1} \\ &= \int_{b_{k-1}\in\mathcal{P}_b} p(b_k|b_{k-1}, \pi(b_{k-1}))\beta_{k-1}(b_{k-1})db_{k-1}. \end{aligned} \tag{4.17}$$

The first equation follows from the marginalization of the probability of $p(b_k|\beta_1, \pi)$ on b_{k-1}. The second equation follows from the belief transition probability function at stage k being conditionally independent of the initial hyperbelief β_1 given the belief b_{k-1} at stage $k-1$ and by substituting $u_{k-1} = \pi(b_{k-1})$ into the belief transition probability function. Again, because of the assumed form of the belief transition probability function and that $\beta_{k-1} \in \mathcal{P}_\beta$, (4.17) is integrable and $\beta_k \in \mathcal{P}_\beta$. □

Chapter 5

A SAMPLING HYPERBELIEF OPTIMIZATION TECHNIQUE FOR STOCHASTIC SYSTEMS

In this chapter we propose an anytime algorithm for determining nearly optimal policies for both finite-time horizon and infinite-time horizon POMDPs using a sampling-based approach. The proposed technique, sampling hyperbelief optimization technique (SHOT), attempts to exploit the notion that small changes in a policy have little impact on the quality of the policy except at a small set of regions. The result is a technique to represent POMDPs independent of the initial conditions and the particular cost function. This allows us to vary the initial conditions and the cost function without having to re-perform the majority of the computational analysis.

We present our technique as an anytime algorithm and verify our results based on standard benchmark problems from the POMDP literature. The proposed method will be developed and analyzed in Section 5.2. However, first we provide background information in Section 5.1. Examples are provided in Section 5.5. We conclude, in Section 5.6, with some future directions and final remarks.

5.1 Introduction

A central theme of almost all approximation techniques is to reduce the set of possibilities to be evaluated, whether simplifying the representation of the belief or by simplifying the cost function. Drawing on insights offered in [70] about why belief sampling-based techniques (such as [5–11]) are so effective, we develop an alternative method that is inspired by graph-based sampling methods (e.g., [40]). In Chapter 4, we introduced the notion of hyperfiltering,

which evolves forward into future stages the probability function over the belief, or the hyperbelief. Because the evolution of a system over the hyperbelief space is deterministic, we can find the optimal plan in the hyperbelief space using an approach derived from standard search techniques.

To do so, we generate a discrete approximation of the structure of the hyperbelief space via a digraph representation. A set of sample points in the hyperbelief space is randomly generated, which correspond to the vertices in the digraph. Edge weights between ordered pairs of samples are then generated using a local planner. The local planner is used to predict the evolution of the system from one sampled hyperbelief to another. Because of the stochastic, partially observed nature of the problem, hyperfiltering is used to estimate the future hyperbelief, under a given policy, from one stage to the next. Instead of requiring that each source hyperbelief sample reach the target hyperbelief sample, the distance between the hyperbelief sample and the target sample is included with the edge information between each pair of samples. This distance is later used to determine the sensitivity of a edge to variations in the source hyperbelief.

To understand and quantify when our proposed algorithm is effective, we characterize the time complexity to approximation error trade-off of our approach. This trade-off is expressed through conditions necessary to achieve exponential convergence of the probability of determining an approximately optimal value at the initial configuration. The convergence rate relative to the time complexity is investigated to determine when a polynomial time solution is attainable. The proposed algorithm exploits both spatial and temporal abstraction; we consider the impact of both on the time complexity of finding an approximately optimal value. A preliminary version of this research was presented in [71].

5.2 Methodology: Generating the Digraph

A central theme of almost all POMDP approximation techniques is to reduce the set of possibilities to be evaluated, whether by simplifying the representation of the belief or by simplifying the cost function. Drawing on insights offered in [70] about why belief sampling-based techniques (such as [5–11]) are so effective, we develop an alternative method that is inspired by graph-based sampling methods (e.g., [40]) and, in particular, the iterative Rapidly-exploring Random Tree method [72] developed by Kuffner and Lavalle.

At a high level, RRT methods create a tree representation of the connectivity of the configuration space of a system. Starting with a single root configuration, this is achieved by iteratively adding edges to expand the tree at each iteration. Edges are created at each iteration by first generating a random sample in the configuration space of a robot. A local planner is then used to traverse some distance toward the sampled configuration from the nearest neighbor of the sampled configuration already in the existing tree. The resulting configuration is inserted into the tree with an edge from the nearest neighbor to the resulting configuration. This process then repeats until a feasible path from the initial configuration to goal configuration is found.

When the process terminates, the RRT provides a representation of the connectivity of the configuration space of the robot starting from some initial configuration. A path through the configuration space can be constructed by traversing the edges in the graph. A path is then the composition of each of the path segments represented by an edge.

Sampling-based approaches including RRTs rely on deterministic evolution of the robotic system to generate paths in the configuration space. POMDPs are stochastic systems; however, the future evolution of the system is stochastic and, as such, sampling-based approaches are not directly transferable. Several attempts have been made to rectify this issue, which were discussed in Chapter 3. These approaches vary from sample path simulation for MDPs [50] to an analytic approximation of the future evolution of a linear Gaussian system

using maximum likelihood observations [44].

Our approach is different from sampling-based approaches including RRTs in that we re-frame the POMDP optimization problem as a deterministic process by lifting the analysis into the hyperbelief space. In Chapter 4, we introduced the notion of hyperfiltering, which forecasts the future behavior of a POMDP by evolving the probability function over the belief into into future stages. Because the evolution of a system over the hyperbelief space is deterministic, we can find the optimal plan in the hyperbelief space using an approach derived from RRT techniques.

Many aspects of our approach are analogous to RRT techniques. In particular, the configuration space of the RRT corresponds to the hyperbelief space in our approach. As with RRTs, our approach is iterative: we expand the graph at each stage. Instead of sampling a configuration, we randomly sample a hyperbelief. We then use a local planner to connect hyperbelief samples. Our local planner essentially constructs a sequence of intermediate hyperbeliefs, via hyperfiltering, which we term a path segment. The resulting path segments are added to the graph as edges.

There are two significant ways in which our approach differs from standard RRT techniques. First, an extra step occurs at each iteration whereby an attempt is made to connect neighbors of the newly created hyperbelief to the graph. Because of this our method generates a graph (with multiple edges per vertex) not a tree. Secondly, our approach is focused on optimality and not feasibility. At the end of each iteration we perform graph optimization to determine the currently best policy from the initial hyperbelief over the graph. Each iteration of the algorithm seeks to improve this policy by adding edges to the graph.

A summary of our approach is as follows. Our algorithm iteratively builds a graph $G =< N, E >$, in which nodes correspond to samples in the hyperbelief space and edges correspond to path segments in the hyperbelief space. At each iteration a new node is generated using an RRT-style expansion step. This requires generating a target hyperbelief sample, and constructing a local path segment in the hyperbelief space. Following this

expansion, our algorithm attempts to connect the new hyperbelief sample to existing nodes in the graph. Finally, the approximate to the value function is updated over the new graph. We now describe the graph generation steps in more detail. Th next section (Section 5.3) will detail the optimization procedure.

5.2.1 Generating vertices: sampling a random hyperbelief

The first phase of the graph expansion proceeds by generating a candidate hyperbelief to add to the graph. Our approach relies on random sampling of the hyperbelief space to generate new candidates. Effective random sampling allows us to explore the hyperbelief space and grants us the ability to quickly and easily generate candidate samples. Unfortunately, effective sampling is a difficult problem, and an enormous amount of research has gone into sampling for probabilistic roadmap methods and RRTs [39, Ch. 7]. Sampling from the hyperbelief space adds new difficulties that are a consequence of the fact that we need to sample from an infinite dimensional space.

Similar to RRT techniques, our approach generates a random hyperbelief sample that is used to expand the graph into the frontier of the hyperbelief space. We begin by randomly generating a sample hyperbelief β^s. For the experimental results presented in Section 5.5, we use a simple sampling procedure that generates a impulse hyperbelief. An impulse hyperbelief comprises a single belief that occurs with probability one. So the sampling procedure reduces to generating a single belief, which we obtain by randomly sampling a set of particles and weights.

With the sample hyperbelief generated, we determine the nearest neighbor β^{near} in the current graph to the sampled hyperbelief β^s. A local policy is used to move β^{near} toward β^s for k stages until the local policy can no longer make any progress. The hyperbelief β_k^{near}, which represents the final hyperbelief achieved by the local planner, is then selected as the new candidate vertex to add to the graph provided it is not too near any existing $n \in N$.

If the candidate is too close, it is rejected and the above process repeats until an adequate candidate vertex is found, at which point the candidate vertex β^{new} is added to the graph. A more detailed description of local planning procedure and how to determine distances in the hyperbelief space are described in 5.2.2.

An illustration of the sampling process is demonstrated in Figure 5.1, where the bottom-right side of the image depicts the belief space $\mathcal{P}_b$, the bottom-left depicts the hyperbelief space $\mathcal{P}_\beta$, and the top is a visualization of the graph vertices in G. The sampling we described ultimately generates hyper-particle sets, which include both a set of beliefs as well as associated weights. However, the method starts by sampling a single impulse hyperbelief β^s. The sampled impulse hyperbelief is illustrated in the right side of the figure. Next, the nearest neighbor β^6 is selected and used as the starting hyperbelief. The correspondence betweenβ^6 and its and its hyper-particle representation is shown in the figure. A local policy is executed, which progresses for three stages. The final simulated hyperbelief (β_3^6) is selected as the new hyperbelief vertex $\beta^9 = \beta^{new}$ to add to the graph.

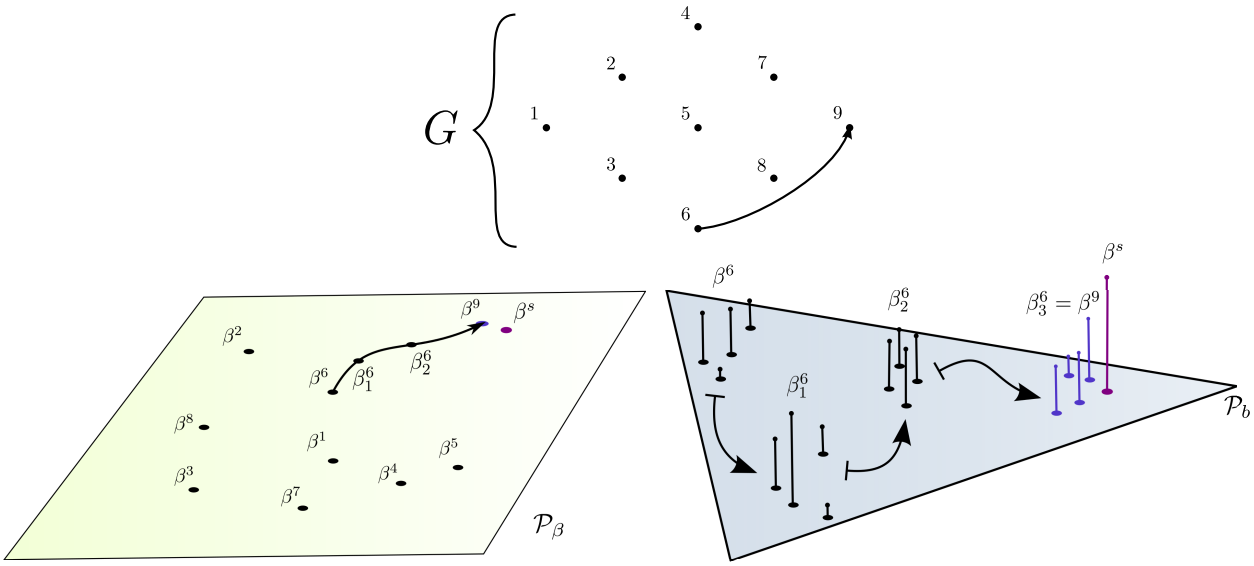


Figure 5.1: Random set of hyperbelief samples in the hyperbelief space $\mathcal{P}_\beta$ and corresponding vertices in digraph G

The motivation for our sampling scheme is that the initial condition becomes less relevant as a system evolves into future stages where the effect of the process and observation

uncertainty will dominate the form of the hyperbelief. Moreover, this sampling approach has the benefit that each sampled hyperbelief will always reside in the reachable space. We will demonstrate later in Section 5.5 that this simple sampling method performs well for the problems that we have considered. However, this sampling approach is expensive and generates numerous similar samples which suggests that better sampling methods are needed to scale to even larger systems.

As is the case with most sampling-based methods, we anticipate that the sampling function will play a crucial role in the convergence properties of our method. One reason the choice of a random sampling function is important relates to the fact that a hyperbelief sample may not be reachable–meaning there is no path from the initial hyperbelief to the hyperbelief sample. Moreover, the performance of the algorithm will be dictated by how quickly and effectively a representative subset of the hyperbelief space is explored.

Our sampling scheme is just one of a many possible schemes. Methods for sampling the hyperbelief space may be as simple as representing the hyperbelief as an impulse function of a single belief, which is sampled uniformly from the belief space–as we have done. Or more complicated methods can be employed. For example, it may be possible to construct hyperbeliefs by sampling from a set of basis functions.

The method developed by Roy and Gordon in [34] is an example of a basis function approach that may be adaptable to sampling the hyperbelief space. In their approach, a basis function set for representing beliefs is learned based on simulation of the system. They project the belief space onto set of exponential functions to find a more efficient representation of the belief space by essentially reducing the dimension of the belief space. Adapting such a method would involve simulating the system in the hyperbelief space and then selecting a limited basis set to represent the possible set of hyperbeliefs. A hyperbelief sample may then be generated taking the sum of the basis functions weighted by randomly sampling the coefficients.

There are a number of alternative sampling techniques used for deterministic systems

that may be adaptable for sampling in hyperbelief space. These include techniques for sampling based on improving the connectivity of the graph [73], on biasing sampling based on information gain [74], finding the equivalent representation of the medial axis [75], or the like. Effective sampling in the hyperbelief space remains as a key area for further investigation.

5.2.2 Planning between hyperbelief samples

In Section 5.2.1 we introduced the notion of generating a path segment from β^{near} to β^{new}. In this section we describe this process of generating path segments between hyperbelief samples in greater detail. The local planner relies on a hyperbelief distance measure to determine the distance between hyperbelief samples. Various hyperbelief measures that may be used in our approach will be briefly introduced as well.

Local planning in our approach is considerably more involved than standard RRT planning methods. Planning is more difficult because we are evolving in the hyperbelief space, which is a functional space. Practical POMDP systems are uncontrollable in the hyperbelief space.[1] RRT methods are concerned with finding feasible paths from a specific initial configuration and a goal configuration. We are attempting to find the optimal path relative to some cost function. Our approach, therefore, needs to consider the cost along the path segment.

Our approach, which is hierarchical, relies on local policies to generate the path segments between edges. Local policies are intended to be only locally optimal or even greedy. This way the planning process is split into two levels, so at the lower level, the local policy is planning is myopic–optimizing simple objectives. This reduces the complexity of planning at the lower level in exchange for having to optimize at the upper level, which selects the sequence of local policies to execute. One key requirement of the local policies is that they are

[1] In this context controllability implies that any hyperbelief cannot be driven to an arbitrary point in the hyperbelief space.

independent from one another, which allows composition of neighboring edges in the graph, which represent a combination of path segments. By composing a series local policies, our method seeks to obtain global behavior.

In our framework local policies use simple cost functions to plan between hyperbelief samples. To determine the control for an intermediate hyperbelief, we employ a policy that minimizes the cost of some function $c_l : \mathcal{P}_\beta \times K \to \mathbb{R}$. For instance, the cost can be the distance to the target hyperbelief. Such a cost function is greedy and seeks to move toward the target hyperbelief at each stage. Alternately, limited time-horizon optimization can be performed to choose a locally optimal path segment between vertices.

Local policies that minimize the distance to a target hyperbelief require definition of an appropriate distance function $d : \mathcal{P}_\beta \times \mathcal{P}_\beta \to \mathbb{R}^+$ in the hyperbelief space. Defining useful hyperbelief metrics still remains an open area of investigation. Fortunately, there are various measures that determine the distance between beliefs such as the Jensen-Shannon divergence [76], earth mover's distance [77], or the like (see [76] for a catalog of probability distance functions). We can impose a hyperbelief distance function by applying ensemble metrics such as the Lukaszyk-Karmowski metric [78] or earth mover's distance by having these methods incorporate a distance measure over the belief space as their weighting function.

Assuming we want to simulate a path segment from β^i to β^j, henceforth $i \to j$. We first select the policy $\pi_{i\to j}$. For distance minimizing functions this is equivalent to setting the goal hyperbelief to β^j. The source hyperbelief vertex β^i is selected and hyper-particle filtering is performed to forecast the system forward one stage into the future using a hyper-particle filtering approximation of Theorem 4.1:

$$\beta_k^{i\to j} \quad = \quad \Phi(\beta_{k-1}^{i\to j}, \pi_{_{i\to j}}),$$

where we denote $\beta_0^{i\to j} = \beta^i$. This process repeats, generating a sequence of intermediate hyperbeliefs $\{\beta_0^{i\to j}, \ldots \beta_{K-1}^{i\to j}, \beta_K^{i\to j}\}$ until a maximum number of iterations is exceeded or until

the greedy policy can no longer make any progress towards the target hyperbelief vertex. The number of stages a local policy executes, K, varies for each path segment but will not exceed some defined maximum value.

Two sets of data are retained from the local policy simulation:

- The sequence of intermediate hyperbeliefs $\{\beta_0^{i\to j}, \ldots \beta_{K-1}^{i\to j}, \beta_K^{i\to j}\}$ that represent the evolution of the system at each instance of time along the path segment representing the edge. This information will be used to generate the cost along a path segment.
- The distance from the terminal hyperbelief along the path segment to the target hyperbelief: $d^{i\to j} = d(\beta_K^{i\to j}, \beta^j)$. This distance is used later to generate value and distance bounding functions.

The optimization procedure in Section 5.3 will utilize this information to compute cost and bounding functions along the path segment.

5.2.3 Adding edges to the graph

The result of the vertex sampling in Section 5.2.1 is a new vertex β^{new} and an edge from β^{near} to β^{new}. We are not content to build a tree like RRT methods. Instead, we generate a graph representation because the cost along an edge is a factor that impacts the overall optimality of a path a graph. Additional edges are generated by selecting existing vertices in the graph to join to β^{new}. Unlike β^{new} which is reachable from β^{near}, it is unlikely that any other hyperbelief vertex β^i in the graph G will be able to reach the new vertex β^{new}. This is a consequence of the uncontrollability of the POMDP system. Our hypothesis is that reachability is not necessary; merely getting close to β^{new} is sufficient for planning purposes. We use the results of the simulated path segments to determine bounding functions (Section 5.3.3) that compensate for this limitation.

We begin by randomly selecting a set of hyperbelief vertices in the graph N_{set}. The sampling method is likely to have a significant impact on the overall effectiveness of our

algorithm. This issue has not been addressed as we have relied on uniform sampling to generate the set. To compensate we choose a large sample size, which likely adds unnecessary computational overhead to our method.

For each β^i in N_{set}, we use the local planning algorithm from Section 5.2.2 to generate a path segment from β^i towards the new vertex β^{new}. The results of the simulated policy including the sequence of intermediate hyperbeliefs $\{\beta_k^{i\to new}\}_{k=0}^{K}$ and the terminal distance $d^{i\to new}$ are retained and the edge $i \to new$ is added to the graph. This method is then repeated where the objective is now to generate a path from β^{new} to each vertex in N_{set}. The result is new edges both to and from the new hyperbelief vertex β^{new}. This process is illustrated in Figure 5.2. In this example, the first part of this process is illustrated. We are attempting to connect each of the existing eight vertices to the newly created vertex β^9. In

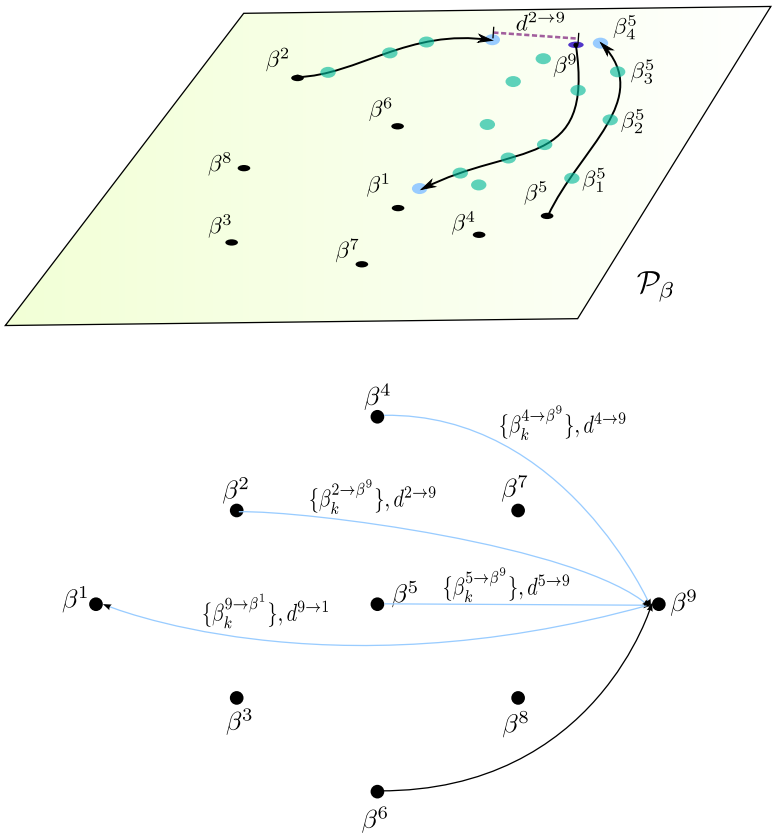


Figure 5.2: Graph G and associated edge information directed to vertex β^9

this example, β^4 proceeds to plan into the future for five stages before the greedy policy is

unable to make any additional progress towards β^9. The distance from the final hyperbelief to the target is represented by $d^{4\to 9}$. Once the local policy terminates, the distance $d^{4\to 9}$ and each of the intermediate hyperbeliefs $\{\beta_k^{4\to 9}\}_{k=0}^{5}$ are added to the edge information of $4 \to 9$. The algorithm for the digraph generation (including both the hyperbelief sampling) is described in Algorithm 3.

Algorithm 3: Generate hyperbelief sample digraph

Input: $\mathcal{P}_\beta$: hyperbelief space, n : number of vertices, K : maximum number of iterations, π : greedy policy
Output: $G =< N, E >$: digraph with edge information

$N\{\beta^i\}_{i=0}^{n} \leftarrow$ Randomly generate samples from $\mathcal{P}_\beta$;
foreach $i \in N$ **do**
  foreach $j \in N$ **do**
    $\beta_0^{i\to j} \leftarrow \beta^i$;
    $k \leftarrow 1$;
    $dist \leftarrow -\infty$;
    while $k \leq K$ *and* $dist \leq min\_dist$ **do**
      $\beta_k^{i\to j} \leftarrow \Phi(\beta_{k-1}^{i\to j}, \pi)$;
      dist $\leftarrow$ distance between $\beta_k^{i\to j}$ and β^j ;
      if $dist \leq min\_dist$ **then**
        add $\beta_k^{i\to j}$ to $E^{i\to j}$ edge information ;
      $k \leftarrow k + 1$;
    add k and min_dist to edge information $E^{i\to j}$;
return G

5.3 Methodology: Attaining the Approximately Optimal Policy

During each iteration, after the new vertex and edges are added, we update the value function to determine the current best policy over the graph. We approximate the value function by optimizing our graph representation of the system. Upper and lower bounding functions are used to estimate the bounds for the cost-to-go for each vertex. The best bounded path in the graph represents the approximately optimal policy starting from the initial hyperbelief. To reduce the uncertainty bounds, we perform a refinement algorithm. The refinement

recursively selects and them simulates the best candidate policy. Each iteration of the refinement algorithm seeks to reduce the bounds on the value function.

Constructing the nearly optimal policy using our approach is a hierarchical process. Starting from the initial hyperbelief node, the optimal edge with the initial hyperbelief as the source is selected. The local policy associated with this edge, i.e. with the goal being the target of the edge, is then executed. When the local policy can no longer make any progress towards the target hyperbelief, the next edge is selected from the optimal path in the graph and the next local policy is executed. As an example, assume an optimal path from the initial hyperbelief is $0 \to i \to j \to l$. The optimal policy would initiate, starting at β^0, by executing $\pi_{0\to i}$, which attempts to transition from β^0 to β^i. Once the local policy terminates, the next policy is selected: $\pi_{i\to j}$. This policy executes until the system can no longer make progress towards β^j. Finally, local policy $\pi_{j\to l}$ is executed. It is important to note that our method does not guarantee optimality for any configuration–only the policy from the initial configuration is nearly optimal. This is a consequence of representing samples as forecasted hyperbeliefs instead of belief samples.

5.3.1 Cost function

Before we can detail the optimization procedure, we must first define the class of cost functions that can be optimized using our approach. As previously described, one of the beneficial aspects of hyperfiltering is that the evolution of the hyperbelief from one *future* stage to the next is deterministic. This implies that the cost function can be expressed as a deterministic function of the hyperbelief. We therefore choose to optimize hyperbelief cost functions of the form $c : \mathcal{P}_\beta \times \Pi \times \mathbb{N} \to \mathbb{R}$, (where Π is the set of all belief and stage feedback policies) with terminal cost $c_K : \mathcal{P}_\beta \to \mathbb{R}$. The cost function, however, must be a separable function in the stage number. Thus, it must be either additive, $c(\beta_k, \pi_k, k) = c^1(k)c^2(\beta_k, \pi_k))$, or multiplicative, $c(\beta_k, \pi_k, k) = c^1(k) + c^2(\beta_k, \pi_k))$, or a combination of the two. For a finite-time

horizon problem, the value function for a given initial hyperbelief β_0 is defined as

$$V(\beta_0) = \min_{\pi \in \Pi} \left[\sum_{k=0}^{K-1} c(\beta_k, \pi_k, k) + c_K(\beta_K) \right]. \tag{5.1}$$

Many practical cost functions are expressed by this cost function representation. Often, the objective is a weighted sum of independent objectives, which may include minimizing how long it takes to complete a task. Discounted cost functions are also in this class of cost functions. The dependence of this cost function on the hyperbelief instead of just the belief enables a richer class of objectives to be represented. For instance, a cost function that depends on the hyperbelief itself may penalize deviation from a desired trajectory in $\mathcal{P}_\beta$. Or the cost function could be used to minimize the amount of uncertainty present in the system as it evolves. Any nonlinear function, not just the linear expectation, over the belief space may be optimized. However, our approach does have the limitation that we can only enforce an upper bound on the number of stages executed. Maintaining strict time horizons is not possible with this technique because the number of stages executed along an edge may vary.

5.3.2 Generating edge costs

We begin the optimization process by generating the cost for each new edge that was added to the graph based on the intermediate hyperbeliefs. The cost for a given edge $i \to j$ is then computed as

$$v^{i \to j} = \sum_{k=0}^{K} c(\beta_k^{i \to j}, \pi_{i \to j}, k),$$

where K is the number of intermediate hyperbeliefs and where $\beta_k^{i \to j}$ is the the intermediate hyperbelief sample at stage k. The result, $v^{i \to j}$, is the cost along the path segment represented by the edge.

For discounted cost functions, the discount applied to the cost function for each edge in G is initialized to start at stage 0. As the evaluation from edge to edge proceeds, the

value of each edge is multiplied by the discount factor raised to the total number of stages performed before reaching the said edge. Thus, if the edge $i \to j$ is reached at stage s, then

$$\gamma^s v^{i\to j} = \gamma^s \sum_{k=0}^{K^{i\to j}} \gamma^k c(\beta_k^{i\to j}, \pi_{i\to j}, k) = \sum_{k=0}^{K^{i\to j}} \gamma^{k+s} c(\beta_k^{i\to j}, \pi_{i\to j}, k).$$

This way the proper discount is applied at each stage. Similar adjustments can be made for other separable time or additive time cost functions of the form specified in Section 5.3.1.

5.3.3 Generating bounding functions

We are going to approximate the cost along an edge $v^{i\to j}$ by deriving an upper and lower cost bounding function $\bar{v}^{i\to j}(\cdot)$ and $\underline{\mathrm{d}}^{i\to j}(\cdot)$, respectively, as a function of the distance from the source hyperbelief β^i. To do this we will need to determine how the value changes as a function of distance. Moreover, we need to consider issues that arise when we join multiple path segments together because a path in the graph is the composition of multiple path segments. To determine how a variation along a path propagates along path segments, we will determine how the distance from the source hyperbelief along a path segment affects the ability of the local policy to reach the target hyperbelief sample. Like with the cost function, we generate upper and lower distance bounding functions $\bar{d}^{i\to j}(\cdot)$ and $\underline{\mathrm{d}}^{i\to j}(\cdot)$, respectively. By back-chaining the effect of these perturbations, we will be able to bound the variation caused at any path segment based on a perturbation on the initial hyperbelief in the path. Combining this with the value bounding functions, we will be able to generate bounds for the value along a path.

The distance bounds can be determined by using the edge information for the digraph. We generate both a lower bound $\underline{\mathrm{d}}^{i\to j}(\cdot)$ and an upper bound $\bar{d}^{i\to j}(\cdot)$ for each edge $i \to j$ in the graph G. Together the lower and upper bounding functions indicate a range for how close a hyperbelief sample may get to the target hyperbelief based on the starting distance to the nominal source hyperbelief (such that $\underline{\mathrm{d}} : \mathbb{R} \to \mathbb{R}$ and $\bar{d} : \mathbb{R} \to \mathbb{R}$). This notion is

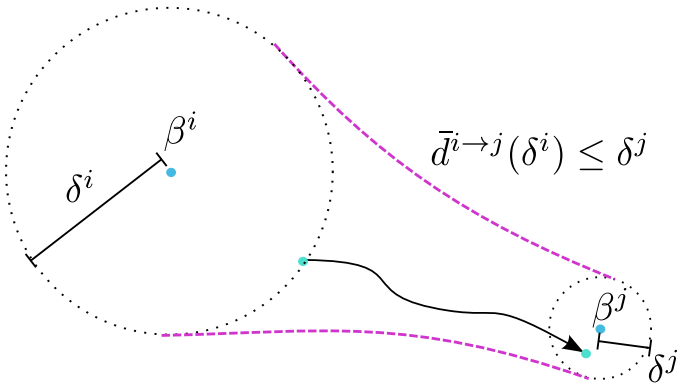


Figure 5.3: Sensitivity of distance δ^j to the target hyperbelief β^jas a function of a perturbation in the distance δ^i from the source hyperbelief β^i

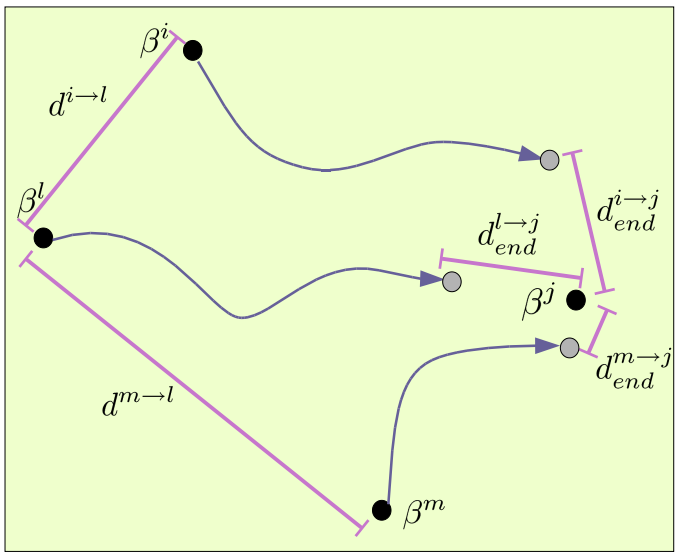


Figure 5.4: Sensitivity of source hyperbelief sample attaining target hyperbelief sample

illustrated in Figure 5.3, where $\bar{d}^{i\to j}(\delta^i) =\leq \delta^j$.

One method to determine the sensitivity of the hyperbelief space around some source hyperbelief sample is to use a set of neighboring samples around the source hyperbelief to determine the ability of the hyperbeliefs in the neighborhood around the source hyperbelief to reach the target hyperbelief sample. This concept is illustrated in Figure 5.4, where two sample hyperbeliefs $\beta^l$and β^m are examined to determine the sensitivity of β^i at reaching β^j. Then a set of linear functions can be fit around the set of samples to generate both the upper and lower distance bounding functions.

Distance information is important because when we derive the optimal solution *over the digraph*, we will have to take into consideration the fact that each source hyperbelief sample may not be able to reach each target hyperbelief sample. To address this issue we will bound, from above and below, the value as a function of distance from the source hyperbelief sample. Thus, we assume that in addition to the value function $v^{i\to j}(\cdot)$, we derive value bounding functions that are a function of the distance from the nominal source hyperbelief. We require both a function giving an upper-bound $\bar{v}^{i\to j}(\cdot)$, and a lower-bound function $\underline{v}^{i\to j}(\cdot)$, so that $\underline{v}^{i\to j}(d) \leq v^{i\to j}(d) \leq \bar{v}^{i\to j}(d),\ \forall d \in \mathbb{R}^+$. In practice, a method substantially similar to the technique used generate distance bounding functions is used to generate the value bounding functions. Likewise, when a stage dependent cost function is used, e.g. discounted cost, the the same method is used to determine upper and lower bounding functions for the number of stages. With the bounding information determined, we can perform graph optimization to determine the nearly optimal policy.

5.3.4 Digraph optimization

As described in Section 5.2.2, local policies are used to plan between vertex hyperbeliefs; we now establish what is the optimal choice of hyperbelief samples to visit. In other words, we need to determine the optimal cost-to-go for each hyperbelief sample in the graph. The nearly optimal policy for the POMDP will then be represented as the optimal path in the graph starting from the initial hyperbelief. To do this, we optimize the value function over the digraph. However, as discussed in Section 5.2.2, each source hyperbelief sample may not be able to attain a target hyperbelief sample using the applied local policy. To account for this we use both the distance bounding functions and the value bounding functions to generate the bounds for a path composed of a plurality of path segments. We express the perturbation in distance from i to l as the distance from the terminal hyperbelief along $i \to j$ to the source hyperbelief β^l: $\delta^{j\to l} = d(\beta_K^{i\to j}, \beta_0^{j\to l})$. The bounds on the value between path

$i \to j \to l$, are then given as

$$\underline{\mathrm{v}}^{i\to j}(0) + \underline{\mathrm{v}}^{j\to l}(\delta^{j\to l}) \leq v^{i\to j\to l} \leq \bar{v}^{i\to j}(0) + \bar{v}^{j\to l}(\delta^{j\to l}).$$

Because there is no perturbation from β^i, $\underline{\mathrm{v}}^{i\to j}(0) = v^{i\to j}$and $\bar{v}^{i\to j}(0) = v^{i\to j}$.

When there is more than two path segments $\beta^{n_1} \to \beta^{n_2} \to \ldots \to \beta^{n_m}$, the upper bound on the cost along the path becomes

$$\bar{v}^{n_1\to n_m} = v^{n_1\to n_2} + \bar{v}^{n_2\to n_3}(\delta^{n_2\to n_3}) + \sum_{r=4}^{m} \bar{v}^{n_{r-1}\to n_r}(\delta_{max}^{n_{r-1}\to n_r}), \tag{5.2}$$

where

$$\delta_{max}^{n_{r-1}\to n_r} = \bar{d}^{n_{r-1}\to n_r} \circ \ldots \circ \bar{d}^{n_2\to n_3}(\delta^{n_2\to n_3}). \tag{5.3}$$

Because we assume linear bounds, the max distance and maximum cost are always a product of the maximum functions. The lower bound distance can be defined by substituting max with min in (5.3) and $\bar{d}$ with $\underline{\mathrm{d}}$. Similarly, a lower bound on the cost can be found by substituting max for min and $\bar{v}$ for $\underline{\mathrm{v}}$ into (5.2). The bound on the distance is determined at each stage and then passed along the path to the next vertex to be used to estimate the bounds on the distance at the next stage. These distance bounds are then used to determine both the upper and lower bounds on the cost for the specified path.

The graph optimization method is based on Dijkstra's algorithm (refer to [79, Ch. 24]). However, if negative weight cycles are allowed (i.e., the robot receives a reward for traversing the cycle), then we replace Dijkstra's algorithm with the Bellman-Ford algorithm [79, Ch. 24], which can handle negative weight cycles but only with an increased computational cost. The basic idea is to start with each vertex $\beta^i \in N$. Update the minimal value for that vertex as the cost $c_K(\beta^i)$. Then for each edge $i \to j \in E$, evaluate the minimum upper bound and the minimum lower-bound on the cost from vertex i to adjacent vertex j and update the cost of the edge. We then repeat this step for the number of vertices. This method

has a computational complexity of $O(|N|^2 + |N|\,|E|)$ (Bellman-Ford has a computational complexity of $O(|N|^2\,|E|)$).

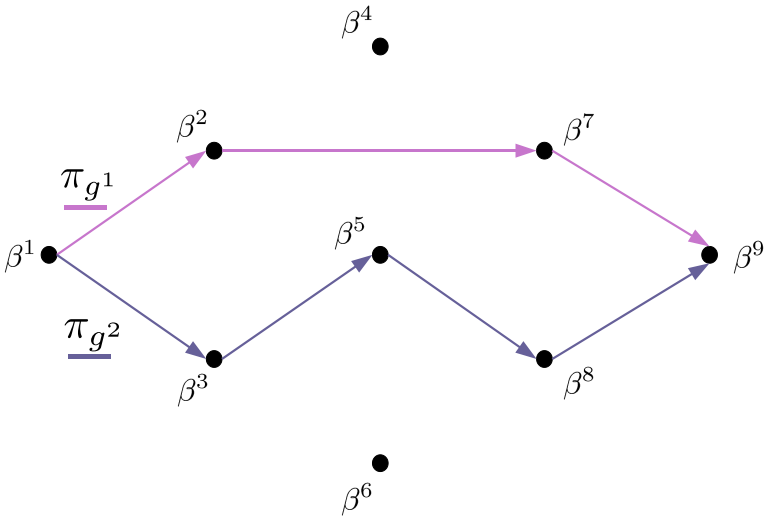
This process is performed to determine the policy that has the least least upper-bound value. The least upper-bound value corresponds to the worst possible performance of the optimal policy. In conjunction we determine the least lower bound value, which corresponds to the policy the best possible performance. We then simulate the system using the least upper-bounded path to determine the actual cost of the perceived optimal policy. With the actual value of the selected optimal policy and the absolute best case value for the current graph, we can provide the range in which the optimal solution may reside, which is the difference between the actual value and the least lower-bound value.

One of the deficiencies of this approach is that the resulting policy, and the the associated value function, is not necessarily optimal for the specified digraph due to perturbation along the path. What is achieved is a bound on the optimal policy and value. To overcome this shortcoming we present an iterative refinement algorithm, which will find the exact optimal solution for a given digraph (but not the POMDP) in a finite amount of time and that each iteration of the method will only improve the quality of the solution.

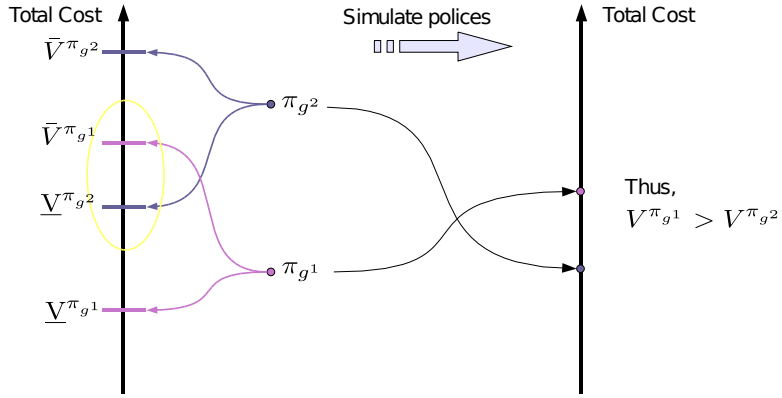
5.3.5 Refinement

The quality of the bounding functions (both the upper and lower bounds) may play a significant role in producing a suitable solution in the previously outlined optimization technique. We present a refinement algorithm to mitigate the effects of the bounding function that also guarantees the optimal solution for a given digraph will be found if enough time is given.

The refinement algorithm recursively selects the current policy least-upper bound policy and simulates it to determine whether the actual value for the policy is lower than the bounds for any other policy. If so, then the optimal policy has been found. Otherwise, the process repeats with the next least upper-bound policy being selected. This process repeats until



(a) Example least lower-bound and least upper-bound optimal switching policy



(b) Possible configuration of the cost for every possible policy

Figure 5.5: Example least lower-bound and least upper-bound optimal switching policy

the optimal policy over the graph is found or until some approximation threshold is reached. The process is described in greater detail below.

Refinement begins after the optimization technique described above is performed to obtain the least upper-bound path with its associated upper-bound $\bar{V}^{\pi_g^1}$ and lower-bound $\underline{V}^{\pi_g^1}$ costs. An example least upper-bound path is depicted in Figure 5.5a as π_g^1.

Then the least upper-bound path is simulated to obtain the actual value $V^{\pi_g^1}$. Then the refinement begins by determining the second least upper-bound path, e.g., π_g^2 in Figure 5.5a. The policy induced by the path π_g^2 is then simulated to find the actual value $V^{\pi_g^2}$ of π_g^2. The

resulting value is compared to the second least upper-bounded value. If the lower bound $\underline{V}^{\pi_g^2}$ of the second-to-lowest value is above the actual value of the already simulated value, then the next least-upper bound path is determined. However, if the actual value $V^{\pi_g^1}$ for π_g^1 resides within the bound of this second to least upper-bound path, then the second to least upper-bounded path is simulated. The cost $V^{\pi_g^2}$ of the resulting simulation is then compared to the previous lowest value. If the newly simulated cost is the lowest, then it is selected as the minimum. In Figure 5.5b, after simulation of π_g^2, it is determined that the value $V^{\pi_g^1}$ is greater than $\underline{V}^{\pi_g^2}$. Thus π_g^2 is simulated and it is discovered that π_g^2 achieves a lower value than π_g^1. The process then repeats with the third least upper-bounded path being checked. This process continues until a maximum number of iterations (less than or equal to the number of possible paths in the digraph) or minimum threshold on the bound of the value is met. At any stage of this process the bound on the value is evaluated as the minimum actual simulated value to the least lower-bounded value of the remaining (non-simulated) paths. The resulting minimum path comprises a set of path segments. The local policies associated with the path segments as well as the ordering of path segments define the approximately optimal policy. The algorithm describing this process is outlined in Algorithm 4.

5.4 Analysis: Rate of Convergence

In this section we will establish the rate of convergence of the SHOT algorithm. We experimentally observed in Section 5.5 that our proposed method performs well for the systems evaluated. However, the experimental results do not illuminate the conditions necessary for our method to perform satisfactorily. The fundamental assumption on which we base our analysis is that the POMDP systems are locally insensitive with respect to both their optimal policy and value function. We attempt to capture this notion and its impact on the performance of the SHOT algorithm. Using this framework we establish conditions for

Algorithm 4: Refinement method

Input: optimal_cost: the optimal cost found so far,
optimal_policy: the current best policy,
verified_policies: the list of polices evaluated so far,
k : refine iteration
Output: optimal_cost,
optimal_policy,
verified_policies

//Initial values generated optimal_policy ← verified_policies ← Dijkstra(G, source_event) ;
check_path ← verified_policies.end ;
while *check_path is not empty* **do**
  i ← Remove and return lowest cost vertex from check_path ;
  foreach *neighbors j of i* **do**
    value_list[j] ← $v^i.end + \bar{v}(i,j)$;
  value, index $leftarrow$ find kth lowest value in value_list ;
  append v^i ← value ;
  append $E \leftarrow i \rightarrow j$;
new_policy ← policy generated from source_event with the kth lowest value ;
append verified_policies ← new_policy ;
new_cost ← simulation result with new_policy ;
if *new_cost < verified_cost* **then**
  optimal_cost ← new_cost ;
  optimal_policy ← new_policy ;
least_bound ← optimal_cost − least lower bound cost //which is generated by searching the digraph and determining $\underline{v}$ against the verified_policies ;
if *$k <$ maximum refine iterations and least_bound $\geq$ threshold* **then**
  $k \leftarrow k + 1$;
  optimal_cost, optimal_policy, verified_policies ← refine(optimal_cost, cost_graph, verified_policies, G, source_event, k);
return *optimal_cost, optimal_policy, verified_policies*

exponential convergence of the probability of determining an approximately optimal value at the initial configuration. This implies that, after each iteration, the chances of our algorithm having obtained an approximately optimal solution increase exponentially if some assumptions are met.

The bounds provided below, however, are both loose and difficult to establish for specific POMDP systems. Tightening the bounds, providing easier to establish conditions, and developing adaptive sampling techniques to enhance the convergence rate are topics of future

research.

Our analysis is inspired from research from both deterministic sampling-based and point-based stochastic research. Much of the framework we present is derived from the analysis of Lavalle and Kuffner in [72], which establishes the convergence of RRTs. Our method extends their approach of feasibility in deterministic systems to finding nearly optimal policies for POMDPs and develops analysis for the computational gains achieved from both spatial and temporal abstraction. In [70] Hsu et al. derive theoretical analysis for conditions sufficient for determining an approximately optimal solution to a POMDP in polynomial time. Their approach, which is developed for point-based algorithms, determines the relationship between the covering number of the optimal reachable belief space (essentially the number of beliefs required to obtain a specified approximation error) and obtaining a polynomial time solution. Their analysis measures the impact of approximating the value function by utilizing the piecewise-linear and convex representation of the value function and the discounting factor for infinite-horizon POMDP systems. Like Hsu et al.'s analysis, we rely on insensitivity properties of POMDP systems in the neighborhood around sampled points. Hsu relies on insensitivity around belief points, we rely on insensitivity around hyperbeliefs.

5.4.1 Exact solution computational complexity

We begin by developing the analysis framework. Then, we demonstrate the computational complexity of the exact solution using this framework. To understand the gains of our method over the exact, exhaustive optimization method, we will need to first determine the lower bound on the computation cost for the exact solution. Exact solutions (refer to [2]) have a computational complexity that is exponential in the time horizon. Papadimitriou and Tsitsiklis [32] prove that POMDPs are PSPACE-Hard and, thus, intractable. To ensure appropriate comparison, we derive a proof of the exhaustive optimization algorithm in an anytime framework.

To begin this analysis we must first know the number of reachable beliefs that can exist when forecasting K stages into the future. For the duration of our analysis we will assume that a POMDP system has $|\mathcal{X}|$ states, $|\mathcal{U}|$ actions, and $|\mathcal{Y}|$ observations. It is further assumed that each belief in the reachable space is unique.

Theorem 5.1. *The exhaustive, anytime algorithm for determining the optimal solution to a K stage POMDP for a given initial belief has time complexity of $O(|\mathcal{X}|^2\,(|\mathcal{U}||\mathcal{Y}|)^K)$.*

What we observe from Theorem 5.1 is that the cost of optimization is trivial relative to the cost of expanding the belief tree. Moreover, the computational complexity grows exponentially in the time horizon.

5.4.2 SHOT computational complexity

Now that we have established the computational complexity of the exact solution, we will move to the time complexity and convergence rate of the SHOT algorithm. We start by establishing that when seeking the exact solution, SHOT will perform better than exhaustive evaluation under general conditions. The presentation of the proofs is as follows:

1. We establish the assumptions required for our framework. Using the stated assumptions, we determine the exact time complexity to discover the optimal solution using the SHOT technique. Along the way, we establish the expected number of iterations required to discover the optimal solution as well as the convergence rate.

2. With both the time complexity of SHOT and of the exhaustive solution, we derive the conditions required for SHOT to be more efficient than the exhaustive solution.

Like with the exhaustive solution derived above, the following proofs are concerned with finding the exact optimal solution. We will relax this requirement in Section 5.4.3–where we derive the computational gains in approximating the exact solution.

The basis of our analysis relies on the existence and size of a set of an optimal sequence of local goals. Let there exist a finite sequence of optimal goal subsets $\mathcal{O}^g = \{\mathcal{O}_0^g, \mathcal{O}_1^g, \ldots, \mathcal{O}_K^g\}$. For each goal set $\mathcal{O}_k^g$, let there also exist a subset $\mathcal{O}_k^b$, which we denote as a basin subset. Let these subsets be defined such that for all k, $0 \leq k \leq K$:

1. Each of the subsets has measure, with $\mathcal{O}_k^g \subseteq \mathcal{P}_\beta$ and $\mathcal{O}_k^b \subseteq \mathcal{P}_\beta$;

2. The initial hyperbelief is contained within $\mathcal{O}_0^b$, or $\beta_0 \in \mathcal{O}_0^b$;

3. The optimal goal set$\mathcal{O}_k^g$ is a subset of the basin set $\mathcal{O}_{k+1}^b$;

4. Each local policy π with the target hyperbelief $\beta \in \mathcal{O}_k^g$ that starts from $\beta^{start} \in \mathcal{O}_k^b$ will terminate with the final hyperbelief in $\mathcal{O}_k^g$;

5. Each local policy π with goal hyperbelief $\beta \in \mathcal{O}_k^g$ is invariant for any starting hyperbelief in $\mathcal{O}_k^b$: the exact same sequence of actions will be executed for each information vector;

6. If there exists a vertex β in the graph $\mathcal{G}$, whereby $\beta \in \mathcal{O}_{k-1}^g$, and a target hyperbelief β^{target} is sampled from $\mathcal{O}_k^g$, then the SHOT algorithm will select β as the source vertex to generate an edge between β and β^{target}.

The first three assumptions guarantee that there is measurable overlap connecting each goal set to the previous stage's goal set and to the initial hyperbelief. The fourth assumption ensures that the local policy is sufficient to link the sequence of goal sets together. Assumption 5 ensures the optimal policy will not change within the set. Assumption 6 ensures that the the method to expand the graph will select the correct vertex when a target hyperbelief is sampled.

We could instead make more restrictive assumptions regarding selection of source vertices that are based on the metric or local value function as an analog to the RRT analysis in [72]. RRT methods typically select the nearest neighbor to expand the graph. We, however, make no assumption that the nearest neighbor (or only the nearest neighbor) is selected when

expanding the graph. For this reason we wish to distill the graph expansion assumption to the essence of what is required for the algorithm to find the optimal policy.

Because we use random sampling in the hyperbelief space, we must determine the likelihood that we eventually sample all of the optimal goal sets to discover the optimal policy.

Definition 5.1. *For some measure μ, let p_β be defined as*

$$p_\beta \equiv \min_{0 \leq k \leq K} \frac{\mu(\mathcal{O}_k^g)}{\mu(\mathcal{P}_\beta^r)},$$

where $\mathcal{P}_\beta^r \subseteq \mathcal{P}_\beta$ is the reachable portion of the hyperbelief space starting from the initial hyperbelief β_0.

If a uniform sampling function is used, the probability of sampling a hyperbelief within any of the optimal goal sets is greater than or equal to p_β. Any POMDP system satisfying the second assumption will have a nonzero probability of sampling each goal set. Because each goal set has a non-zero chance of being sampled, the SHOT algorithm's probability of successfully finding the optimal solution will increase at each iteration.

Theorem 5.2. *Given a POMDP system with $p_\beta > 0$, the probability that the SHOT algorithm finds the optimal solution after n iterations is $1 - \frac{1}{2}e^{-2(np_\beta - K + 1)^2/n}$, where $n \geq Kp_\beta$.*

Corollary 5.1. *The expected number of iterations required for the SHOT algorithm to find the optimal solution is K/p_β.*

We have established the exponential convergence of the probability of sampling the optimal policy as well as the expected number of stages required to find the optimal solution. The rate of convergence only tells a portion of the story, however. To fully quantify the SHOT algorithm we need to establish the time complexity of finding the optimal solution and the conditions necessary for the SHOT algorithm to require less computational effort than the exact solution.

Lemma 5.1. *The SHOT algorithm executing for n iterations on a K stage finite-horizon optimization of a POMDP system with a uniform sampling method over the hyperbelief space has a time complexity of*

$$O(n|\mathcal{X}|^2|\mathcal{Y}|^K)$$

if only a single vertex needs to be expanded at each iteration and, at each of the n iterations, a new policy edge is added to the graph. It is further assumed that the time complexity of selecting the vertex is negligible.

Again, as with the case of the exact optimization, the complexity of the expansion is the primary contributor to the algorithmic complexity. However, if the number of edges in the graph increases so that $n > |\mathcal{X}||\mathcal{Y}|^K$, the complexity of optimization will outweigh the complexity of expansion. This scenario is unlikely for any practical system, and, as we will show in Section 5.5, the SHOT algorithm often converges to nearly optimal solutions for relatively small values of n.

Several difficult-to-establish assumptions are made, primary of which is the size of the goal sets. We also neglect the cost of selecting neighbors when expanding the graph. However, if we assume that a low-dimensional parameterization of the hyperbelief can be constructed, then an efficient K-d tree implementation can be used adding a minimal impact to the algorithm. This analysis is all-or-nothing and the convergence does not take into account the impact of nearly-optimal policies or samples in the convergence of the algorithm; instead, only the exact optimal policy is considered and the probability of sampling that policy is analyzed.

We have established the time complexity of the SHOT algorithm for a given number of iterations, but it still remains to determine under what conditions the SHOT algorithm will outperform the exact, exhaustive solution.

Theorem 5.3. *For SHOT to have a lower time complexity than the exact solution, the minimal probability of sampling each goal set must be: $p_\beta \geq \frac{K}{|\mathcal{U}|^K}$.*

One interesting observation about the relationship between p_β and the number of actions is that as the number of possible actions increases, the likelihood of the randomized SHOT method performing better increases exponentially. Additionally, as the number of stages increases, the randomized method will outperform the exact method.

5.4.3 Computational complexity gains from approximation

While randomized sampling reduces the time complexity to some degree, finding the optimal solution is still expected to take exponential time as a function of the the time horizon. Furthermore, the specification of the optimal goal sets has obfuscated some of the difficulties in sampling in the hyperbelief space. We will now present an approximation method that uses spatial abstraction to reduce the time complexity (with an example that reduces the complexity to polynomial time if the approximation error is large enough). The presentation of the proofs is as follows:

1. The assumptions we used for the exact case are modified to broaden the framework to include approximation error.

2. Using the modified assumptions, we determine the computational requirements to achieve a specified approximation error using the exact expansion of the hyperbelief, which has the potential to grow exponentially in the time horizon.

3. To reduce the computational complexity further, the error introduced by using a reduced representation of the hyperbelief via a fixed number of samples is investigated.

4. We then identify the approximation error to time complexity trade-off using the reduced hyperbelief approximation with the SHOT technique.

To extend our analysis to approximate solutions, we will need to add additional assumptions. The requirement that a policy starts in one goal set and terminates in the next goal set restricts the size of the goal sets themselves. By reducing the requirement to sample

intermediate hyperbeliefs that are on optimal path to sampling hyperbeliefs in the neighborhood around optimal path, we will effectively broaden the region around the goal set to increase the probability of successfully finding the optimal path at each stage. Imposing dissipative assumptions on the POMDP system's evolution under the local policy will reduce the restrictions on the evolution of the system from one goal set to the next. Exploiting the increased capabilities of the local policy, we can reduce the time complexity of expanding the graph by approximating the hyperbelief using hyper-particle filtering by using a limited subset of beliefs. The error introduced by approximating the hyperbelief is offset by the dissipative property of the system under the local policy. The combination of these two factors, increased probability of finding the optimal solution at each stage and reduced time complexity when expanding the graph, can potentially lead to a significant reduction in the total time complexity required to find a nearly optimal solution.

To analyze the gains from approximation, we will assume there exists an ϵ-goal set $\mathcal{O}_k^{g,\epsilon}$ around each goal set $\mathcal{O}_k^g$, such that the distance between any set of hyperbeliefs $\beta^1 \in \mathcal{O}_k^g$ and $\beta^2 \in \mathcal{O}_k^{g,\epsilon}$ is less than ϵ: $d(\beta^1, \beta^2) \leq \epsilon$. For each ϵ-goal set $\mathcal{O}_k^{g,\epsilon}$we define an ϵ-basin set $\mathcal{O}_k^{b,\epsilon}$ all $0 \leq k \leq K$. The ϵ-basin set is analogous to the original basin set but now it defines the region for which the local policy will terminate in the ϵ-goal set. Let ϵ_{min} represent the minimum distance of all ϵ-goal sets $\mathcal{O}_k^{g,\epsilon}$ for $0 \leq k \leq K$. The updated assumptions are as follows for all $0 \leq k \leq K$:

1. The ϵ-goal set$\mathcal{O}_k^{g,\epsilon}$ is a subset of the ϵ-basin set $\mathcal{O}_{k+1}^{b,\epsilon}$;

2. Each local policy π with the target hyperbelief $\beta \in \mathcal{O}_k^{g,\epsilon}$ that starts from $\beta^{start} \in \mathcal{O}_k^{b,\epsilon}$ will terminate with the final hyperbelief in $\mathcal{O}_k^{g,\epsilon}$;

3. For any target hyperbelief $\beta \in \mathcal{O}_k^{g,\epsilon}$, the POMDP system under the local policy π is dissipative: if the minimum distance from $\beta_k \in \mathcal{O}_k^{b,\epsilon}$ to $\mathcal{O}_{k-1}^{g,\epsilon}$ is d_k and the minimum distance from β_{k+1} to $\mathcal{O}_k^{g,\epsilon}$ is d_{k+1}, then $d_k \leq d_{k+1}$.

4. If there exists a vertex β in the graph $\mathcal{G}$, whereby $\beta \in \mathcal{O}_{k-1}^{g,\epsilon}$, and a target hyperbelief β^{target} is sampled from $\mathcal{O}_k^{g,\epsilon}$, then the SHOT algorithm will select β as the source vertex to generate an edge between β and β^{target}.

5. Each ϵ-goal set $\mathcal{O}_k^{g,\epsilon}$is strictly larger than the goal set $\mathcal{O}_k^g$: $\epsilon_{min} > 0$.

The third assumption represents the largest deviation from the previous assumptions. Now we no longer restrict the policy to be invariant; instead we empower the local policy to reject small disturbances. By increasing the capabilities of the local policy we can broaden the region around a goal set to become the ϵ-goal set. The larger the region, the faster the SHOT algorithm will converge. This comes at the cost of exactness: increasing the size the ϵ-goal sets increases the approximation error. The maximum value of ϵ that can be used and still satisfy the above conditions will depend strongly on the POMDP system and the local policy.

As discussed in Section 5.3.4, the SHOT algorithm builds both upper and lower bounds of the value function at each vertex in the graph: we assume that the value function along each edge is Lipschitz bounded. For the duration of the analysis we will assume that there exists an ξ, such that the value along edge $i \to j$ for any hyperbelief $\tilde{\beta}^i \in \mathcal{P}_\beta$ from hyperbelief β^i is $|\Delta V_{i\to j}| \leq \xi d(\tilde{\beta}^i, \beta^i)$ for any edge $i \to j$ in the graph G. If the cost function is Lipschitz bounded by ξ, then under the assumptions above, the value along an edge is necessarily Lipschitz bounded by ξ.

Definition 5.2. *For some measure μ consistent with the assumptions above, let p_β^ϵ be defined as*

$$p_\beta^\epsilon \equiv \min_{0\leq k\leq K} \frac{\mu(\mathcal{O}_k^{g,\epsilon})}{\mu(\mathcal{P}_\beta^r)}$$

where $\mathcal{P}_\beta^r \subseteq \mathcal{P}_\beta$ is the reachable portion of the hyperbelief space starting from the initial hyperbelief β_0.

Since p_β has measure, we know for a sample bound ϵ_{samp} that $p_\beta^\epsilon \geq \epsilon_{samp}/d_{max}$,where the maximum distance between any two hyperbeliefs in $\mathcal{P}_\beta$ is d_{max} , which we assume to be

finite.

Theorem 5.4. *For the SHOT technique to achieve a nearly optimal policy with error* $0 < V_{err} \leq \epsilon_{min}/d_{max}$ *using exact expansion of the hyperbelief, the total expected time complexity is of the order of*

$$O(\frac{\xi K}{V_{err}}|\mathcal{X}|^2|\mathcal{Y}|^K).$$

It is apparent that the time complexity savings for a specified error V_{err} is strongly dependent on the time horizon and the Lipschitz bound.

Reducing the expected number of iterations required to find the optimal solution has a large impact in the overall time complexity of the SHOT algorithm. However, a major contributor to the time complexity persists as the term that is exponential in the time horizon: $|\mathcal{Y}|^K$. This term results from the computational cost to expand the complete hyperbelief at each iteration. We propose to approximate the complete hyperbelief using a hyper-particle approximation. By limiting the number of beliefs within the hyperbelief at each stage we will restrain the growth of the hyperbelief well below the full exponential representation.

We demonstrated in Section 4.4 that hyper-particle filtering with m beliefs has a time complexity of $m|\mathcal{X}|^2|\mathcal{Y}|$, if the exact belief is propagated. If $m \ll |\mathcal{Y}|^{K-1}$ then hyper-particle filtering achieves substantial computational savings.

Theorem 5.5. *The time complexity of the SHOT algorithm having executed for* n *iterations and using hyper-particle filtering with* m *beliefs is of the order* $O(nm|\mathcal{X}|^2|\mathcal{Y}| + n^2)$.

We use (5.6) instead of the final result in Lemma 5.1 because if m is small enough n^2 becomes a contributing factor in the overall time complexity.

The error induced from approximating a hyperbelief with a hyper-particle representation needs to be considered when evaluating the impact on the optimization result. The total approximation is the combination of the hyper-particle filtering and the sampling approximation bounds. This combination is still restricted to be less than ϵ_{min}. If the bound exceeds

ϵ_{min}, the requirement that any hyperbelief in one ϵ-goal set will terminate in the next ϵ-goal set will be violated.

Theorem 5.6. *Let the hyper-particle filtering approximation be ϵ_{hpf}, then the time complexity of generating a policy with the approximated value error V_{err} is*

$$O\left(K^2(\frac{K\xi}{V_{err}} - \frac{1}{\epsilon_{hpf}})^2 d_{max}^2 m|\mathcal{X}||\mathcal{Y}|\right),$$

if the sampling approximation error ϵ_{samp} satisfies $\epsilon_{samp} \leq \epsilon_{min} - \epsilon_{hpf}$.

The result implies that there is a coupling between the hyper-particle filtering approximation and sampling approximation. As a consequence there is a direct trade-off between hyper-particle filtering error and the sampling error. Both can increase to a point, but after that point an increase in one source of error will require a decrease in the other. The net effect on the overall time complexity is still dependent on two variables: V_{err} and ϵ_{hpf}.

We can eliminate ϵ_{hpf} by considering the convergence rate of the hyper-particle filtering technique. Le Gland and Oudjane [80] demonstrate that, under some mixing assumptions, particle filtering has a convergence rate that is inversely proportional to the number of samples:

$$E\left[\left(\varphi(\tilde{b}_k^m) - \varphi(b_k)\right)^2\right] \leq \frac{c||\varphi||}{m},$$

where c is some constant that depends on the system and $\varphi \in B(\mathbb{R}^{|\mathcal{X}|})$, where $||\varphi|| \equiv \sup_{x \in |\mathcal{X}|} |\varphi(x)|$. Experimental results of hyper-particle filtering [56] support this bound. Thus, we can consider the hyper-particle filtering approximation error in terms of the number of samples.

Corollary 5.2. *Let a POMDP system have a hyper-particle filtering convergence rate*

$$\epsilon_{hpf} \leq \frac{C_{sys}}{m},$$

where C_{sys} is a system dependent and non-negligible constant term. The total time complexity

of the SHOT algorithm relative to the number of hyper particle samples m and the desired error $0 < V_{err} \leq \epsilon_{max}/d_{max}$, cost function Lipschitz bound ξ and time horizon K is

$$O\left(K^2(\frac{K\xi}{V_{err}} - \frac{m}{C_{sys}})^2 d_{max}^2 m|\mathcal{X}||\mathcal{Y}|\right).$$

Under the above convergence rate assumption, the SHOT algorithm is quartic time in the number of stages K, cubic time in the number of hyperbelief samples m, and linear time in both the size of the state-space $|\mathcal{X}|$ and the observation-space $|\mathcal{Y}|$. The result is polynomial in all terms but the error term V_{err}. The inverse relationship is highly nonlinear and quickly overwhelms all other terms as the error term diminishes. However, as the error term increases the value stabilizes to near constant value. Thus, for reasonable approximation errors, the SHOT method will achieve polynomial time complexity.

5.4.4 Temporal abstraction

We conclude with an analysis by establishing the benefits of temporal abstraction on the overall time complexity of the SHOT method. Besides providing spatial abstraction, our method exploits the fact that a local policy may execute for a plurality of stages. This enables us to achieve temporal abstraction via the potential to reduce the number of stages the system optimizes for each policy and the number of iterations required to find the optimal solution.

Theorem 5.7. *Let the minimal temporal abstraction be t stages, so that instead of K optimal goal sets there are $\lceil K/t \rceil$. Let all the other assumptions still hold. The expected number of stages to find the optimal solution reduces to $\frac{K}{tp_\beta}$ and the total expected time complexity reduces to*

$$O\left(\frac{K}{tp_\beta}|\mathcal{X}|^2|\mathcal{Y}|^K + \left(\frac{K}{tp_\beta}\right)^2\right). \tag{5.4}$$

It is immediately clear that expansion–right-hand side of (5.7)–does not benefit from the

temporal abstraction. This makes intuitive sense because each policy that is expanded has to be simulated for each stage. Optimization, conversely, is reduced by the inverse square of the temporal abstraction. If there is a way reduce the computational burden when simulating the policy, then more tangible computational savings would be observed.

5.5 Results

To verify the proposed technique, we applied it to several of the benchmark problems found in the literature: maze20, hallway2, CIT, and Fourth (from [2]). We selected this set of benchmark systems because they vary in size and represent a variety of systems. These systems range from small to moderate in size, from low noise to moderate noise, and from minimal observability to near full observability.

Each of these examples is an expected state cost system. This class of cost function imparts a simple and elegant structure on the value function in the belief space; namely, the optimal value function is piecewise-linear and concave over the belief space (it is convex when considering a reward function instead of a cost function). Traditional value-iteration methods exploit this structure when determining approximately optimal solutions. This piecewise-linear and concave form in the value function arises from the fact that the cost function is linear in the belief space and that the backup from one stage to the next is a linear function for each action. The optimization function is then the minimum over the set of linear functions in the belief space, which is a piecewise linear function. Such systems are ideal for analysis as POMDPs with an expected state cost have been studied extensively.

For these examples, we generate upper bounds experimentally by measuring the sensitivity of the cost associated with the starting distance of each neighboring edge. For example, given edge $i \rightarrow j$, we determine the relationship of the cost for $v^{l \rightarrow j}$ relative to the distance of edge l to i for each l such that edge $l \rightarrow j$ exists. We then generate a linear upper and lower bound for the cost. In a similar manner, bounds for the distance from one edge to the

next were determined as well as the bounds on the number of elapsed iterations.

During the expansion phase, each sample hyperbelief is generated as an impulse hyperbelief, which comprises a single belief. The belief is represented as a set of particles and weights. To generate the belief, we first randomly choose the number of particles, n, by sampling a Poisson distribution with mean on the order of $\sqrt{|\mathcal{X}|}$. A set of s particles are then uniformly sampled from the state space $\mathcal{X}$. Next, each particle is randomly weighted. The weight for the first state is selected by sampling a uniform distribution. The next state is weighted as one minus the weight of the first sample. This repeats until all states are weighted. Finally, the weights are normalized to ensure the total weight sums to 1.

The Lukaszyk-Karmowski distance measure [78] was selected as the hyperbelief distance function. This pseudo-metric is essentially a measure of the expected distance between each pair of beliefs between two hyperbeliefs, where the probability of each pair is just the product of the probability of both beliefs. Jensen-Shannon divergence was used as the belief distance function. Jensen-Shannon divergence is a symmetric version of relative-entropy, or Kullback-Leibler divergence. In each of the examples below, the local policy function sought to minimize this hyperbelief distance function.

The selected set of examples are discounted infinite horizon expected state reward systems. The discounting factor plays an important role in the value function and, thus, the resulting policy of the system. The SHOT method as described was derived for finite-time horizon problems. Similar to the technique we describe above to generate a bound for the cost, we generate a bound on the number of iterations that elapse while traversing edges of the digraph to account for the effect of the discounting on cost along an edge. This way we can propagate the discount from one event to the next via the number of iterations that elapse between events, which enables to generate an estimate of the discounted cost along multiple edges.

The results of the the simulations are presented in Table 5.1. The proposed method is compared against SARSOP [11]. SARSOP is not a temporal/spatial abstraction method.

Instead it is a point-based anytime algorithm. However, it is a leading POMDP approximation method for the benchmark problems presented. For the purposes of our analysis, we ran our method using a set number of hyperbelief samples: 20 samples for 4x4 and Maze20, 100 samples for Hallway2, 200 samples for CIT; and 300 samples for Fourth. For each of the examples, SARSOP was executed for the same time duration as SHOT. Each test was conducted by running both techniques between 10 to16 times to establish the average performance. By performing analysis in this way, we attempted to normalize the effectiveness of the results to obtain an equitable comparison.

Table 5.1: Verification via comparison to benchmark problems

				Expected Total Cost	
	$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{Y}\rvert$	$\lvert\mathcal{U}\rvert$	SARSOP	SHOT
4x4	16	2	4	-3.750	-3.705
Maze20	20	8	6	-38.788	-31.000
Hallway2	92	17	5	-0.547	-0.503
CIT	284	28	4	-0.834	-0.850
Fourth	1024	28	4	-0.594	-0.814

As is typical with infinite horizon problems, there is a reset state that transports the system back to the initial configuration once the system enters the goal state. This artificial structure imposed on the problem to coax the system to be amenable to discounted cost/reward problems can impart a periodic nature to the control policy. In such cases there may be correlated policies between system resets, which reduces the burden on the learner/optimizer. However, the representation is often not as simple as a single period repetition; only the fraction of the probability that is in one of the goal states is redistributed to the initial configuration. The system may go through several resets before it has a similar distribution over the state-space as it has encountered before. As a roadmap method, SHOT may implicitly benefit from this aspect present in all of the benchmark problems that we

evaluate. A definitive evaluation of this effect was not performed.

The smallest and simplest example, 4x4, represents a system with no observational discrimination except at the goal state, in which a reward is obtained. The system incurs a small amount of process noise for each action it takes. The optimal solution is a greedy policy, which directs the robot directly to the goal state. By evaluating our approach against this trivial example, we are able to baseline our result and help quantify the inherit loss due to the abstraction. In this case, we see for our naive sampling procedure that the optimal is obtained by SARSOP is -3.750, whereas SHOT obtains -3.705. The difference is likely due to a limitation of the metric used by the local policy or a consequence of the sparse representation hyperbelief space. Regardless, we observe a minimal impact on the value obtained using SHOT.

The Maze20 example, while only consisting of 20 states and 8 observations, has a complex control policy. The system is unobserved unless certain actions are taken, which either give a reading of the presence of a wall to the north/south or east/west of the robot. The other actions enable movement in the four cardinal directions. The robot initially starts with the possibility of being in three of corners of the maze. Because of the initial uncertainty, the MDP solution is severely sub-optimal. The discount factor is 0.90, which forces a myopic policy (discount factor is 10% after just 20 iterations). Such strong discounting entices the system to reach the goal state quickly. The consequence of this effect is that the control policy must balance between localization of the robot and goal seeking behavior. The disparate starting states also create disparate sample paths, which require a relatively complex hyperbelief representation (large number of belief samples) to forecast the behavior of the system. For this example we see that SARSOP's solution outperforms SHOT by a respectable margin of -7.8 or a 25% improvement. The poor performance of SHOT is likely due to the complexity of the control policy in comparison to the number of samples use to generate the policy.

The Hallway2 example makes observations at every iteration and the discount factor is

significantly higher at 0.95 (higher implies lower discounting per iteration). This increase allows the control policy to be less myopic and consider less short term gain. The starting configuration for Hallway2 is a uniform distribution over all states. The control actions include rotation and forward and backward movement. The nonholomic constraints add additional complexity to the planning process. We can see from Table 5.1 that this is the first example that is non-trivial that SHOT becomes competitive with SARSOP, achieving a reward within 8%.

The next set of examples represent a system similar to Hallway2: a robot system in a structured space that is attempting to reach a goal state. Unlike Hallway2, CIT and Fourth start in a known initial condition (with probability 1) and have a discount of 0.99, which requires 230 iterations to reach a discount factor of 10%. Essentially, CIT and Fourth are just scaled versions of the same problem. We see the benefit of temporal and spatial abstraction begin with these two examples: SHOT achieves a gain of 2% over SARSOP for the CIT example and a substantial gain of 37% for the Fourth example.

The experimental results are buttressed by the theoretical results presented in Section 5.4, and establish the validity of the SHOT framework and the potential of the approach. From the experimental analysis, we see that the potential value of SHOT becomes apparent as the size of the system increases. The sampling method we used is naive and there is great potential in selecting better or biased sampling techniques that converge much more quickly, which will enable SHOT to expand to even larger systems.

Using a technique derived from SHOT, Candido et al. in [48] explore using local policies (or micro-options) that are constructed from domain knowledge applied to multi-agent POMDP systems. The problem of interest in their approach is to extinguish a fire using a plurality of firefighters. The exhaustive set of policies for this class of systems grows exponentially with the number of agents, the number of observations, and the size of the state-space. While no absolute evaluation of effectiveness of the results of their method is possible, convergence and the development of non-greedy policies was observed. Inter-

estingly, their learner developed non-intuitive behavior that succeeded in extinguishing the fire where user defined policies failed. Their work demonstrates the ability of the SHOT framework to extend to both large and multi-agent systems.

5.6 Conclusion

A method for finding nearly optimal policies for POMDPs with total cost or finite time horizons was presented. The proposed method is a sampling-based technique using a hierarchical planner. The lower level planner executes local, greedy feedback policies, and the higher level planner coordinates the order of hyperbelief samples that are visited. This method attempts to capture the connectivity of the POMDP system, while simultaneously learning the nearly optimal policy for the stated objective function.

Analysis was performed to determine loose upper-bounds on the convergence rate of the method and to establish several requirements for the method to outperform the exact optimal method. The computational-to-optimal trade-off was examined to identify conditions and scenarios for which dramatic computational savings can be had with only a minor impact on the quality of the optimal solution. Experimental results not only support the analysis but exceed the analysis capabilities

Future research includes evaluating alternate sampling schemes, such as generating event samples from the observation space. Sampling from the observation space has the potential to alleviate some of the issues that arise from attempting to sample targets from the belief space directly. Biased sampling techniques may also hasten the convergence of the proposed algorithm. Another avenue being investigated is sensitivity analysis, which will enable us to analyze the effect of a perturbation on the performance of a local policy. This should enable us to field biased sampling techniques more readily and to represent the effect of a perturbation more precisely than the Lipschitz bounds currently used.

5.7 SHOT Proofs and Analysis

5.7.1 Exact solution time complexity

To begin this analysis we must first know the number of reachable beliefs that can exist when forecasting K stages into the future. For the duration of our analysis we will assume that a POMDP system have $|\mathcal{X}|$ states, $|\mathcal{U}|$ actions, and $|\mathcal{Y}|$ observations. It is further assumed that each belief in the reachable space is unique.

Lemma A1. *Then the total number of reachable beliefs for a POMDP system with a finite horizon K is $\frac{(|\mathcal{U}||\mathcal{Y}|)^{K+1}-1}{|\mathcal{U}||\mathcal{Y}|-1}$.*

Proof. A POMDP system proceeds from an initial belief, then each stage of expansion from stage k to $k+1$ requires that each belief at stage k build out each of the $|\mathcal{U}|$ actions, each of which requires $|\mathcal{Y}|$ observations to be added to the tree, thus leading to $|\mathcal{U}||\mathcal{Y}|$ new beliefs. The number of beliefs at the kth stage is $T^e_{i-1}|\mathcal{U}||\mathcal{Y}|$. By induction it is trivial to show that, starting at stage 0 with a single belief, the total time complexity of expanding the kth stage is $(|\mathcal{U}||\mathcal{Y}|)^k$. The total cost of expanding over all k stages is sum of the time complexity for each of the stages: $\sum_{k=1}^{K}(|\mathcal{U}||\mathcal{Y}|)^k$. The sum of this quantity can be solved explicitly to be $\frac{(|\mathcal{U}||\mathcal{Y}|)^{K+1}-1}{|\mathcal{U}||\mathcal{Y}|-1}$ (i.e., $\sum_{k=0}^{K} ar^k = a\frac{r^{K+1}-1}{r-1}$). Finally, there is the initial hyperbelief (assuming a single belief with probability one) at stage 0. □

With a bound on the number of possible reachable beliefs, the time complexity of expanding (or simulating) each possible reachable belief can be derived.

Lemma A2. *The total time complexity, T^e_{total}, to expand all of the reachable beliefs of a given POMDP system up to stage K is of the order*

$$T^e_{total} = O(|\mathcal{X}|^2\,(|\mathcal{U}||\mathcal{Y}|)^K).$$

Proof. Each belief requires $O(|\mathcal{X}|^2)$ to predict the belief and $O(|\mathcal{X}|)$ to update. From

Lemma there are a total of $\frac{(|\mathcal{U}||\mathcal{Y}|)^{K+1}-1}{|\mathcal{U}||\mathcal{Y}|-1}$ beliefs expanded up to the stage K (the initial belief is not expanded). Each belief generated takes $O(|\mathcal{X}|^2)$ time to generate. Thus, the total computational effort is greater than

$$|\mathcal{X}|^2 \frac{(|\mathcal{U}||\mathcal{Y}|)^{K+1}-1}{|\mathcal{U}||\mathcal{Y}|-1} = O(|\mathcal{X}|^2 (|\mathcal{U}||\mathcal{Y}|)^K).$$

The inequality results from the fact that, if $r \geq 1$ and $K \geq 1$,

$$\begin{aligned} \frac{r^{K+1}-1}{r-1} &\leq 2r^K \\ r^{K+1}-1 &\leq 2r^{K+1} - r^K \\ -1 &\leq r^K(r-1). \end{aligned} \tag{5.5}$$

The inequality in the last equation holds for $r \geq 1$ as the right-hand side is always positive for nonnegative values of $r \geq 1$. The time complexity is proven by substituting $|\mathcal{U}||\mathcal{Y}|$ for r and observing that the quantity is bounded by some constant. □

From Lemma A2, we observe that there is an exponential growth in the time complexity at each stage and that the total time complexity incurred to simulate the entire reachable set of beliefs is exponential in the time horizon. Now we need to determine the additional cost incurred when optimizing the graph after each iteration to determine the total time complexity of finding the optimal policy for a given POMDP system.

Lemma A3. *The total time complexity T^o to optimize the the exact solution at each stage from $0 \leq k \leq K$, is of the order*

$$T^0 = O((|\mathcal{U}||\mathcal{Y}|)^K).$$

Proof. At each stage k, the optimal policy must be propagated back to the initial belief, since the system is a finite-horizon problem implying a time varying policy. Starting from

the set of beliefs at the $k-1$ stage (or level of in the reachable belief tree) to stage 0, the optimal action for each belief must be selected. Thus, each belief must select from $|\mathcal{U}|$ actions, which corresponds to $|\mathcal{U}||\mathcal{Y}|$ edges in the tree being evaluated. Total computation cost then becomes

$$\begin{aligned} T_k^0 &= |\mathcal{U}||\mathcal{Y}|T_{k-1}^e \\ &= O\left(|\mathcal{U}||\mathcal{Y}|\right)^k. \end{aligned}$$

The second step follows from the the inequality used in Lemma A1: so the optimization cost at stage k is $O(|\mathcal{U}||\mathcal{Y}|\left(|\mathcal{U}||\mathcal{Y}|\right)^{k-1})$. The total time complexity T^o is the sum of the time complexity to optimize the POMDP system at each stage (starting from stage 1):

$$\begin{aligned} T^o = \sum_{k=1}^{K} T_k^0 = &\sum_{k=1}^{K} O((|\mathcal{U}||\mathcal{Y}|)^k) \\ &= O\left(|\mathcal{U}||\mathcal{Y}|\right)^K. \end{aligned}$$

The inequality resulting in the second equation is a direct application of the geometric series from Lemma A1 and the exponential equation inequality (5.5) in Lemma A2. □

We can now state the total cost to both expand and optimize a POMDP system with the cost to expand the belief tree and the cost to optimize over the tree established.

Theorem 5.1. *The exhaustive, anytime algorithm for determining the optimal solution to K stage POMDP for a given initial belief has time complexity of $O(|\mathcal{X}|^2\left(|\mathcal{U}||\mathcal{Y}|\right)^K)$.*

Proof. For any cost function, the worst case solution will require evaluation of each belief in the tree of of possibilities up to stage K. There are two factors that contribute to the time complexity, the cost of expanding the tree of beliefs (T^e) and the cost of optimizing over the tree (T^o) at each stage $0 \leq k \leq K$, such that the total computational effort T_{total} is given as

$$T_{total} = \sum_{i=0}^{k} [T_i^o + T_i^e] = T^o + T^e$$
$$= O(|\mathcal{X}|^2 (|\mathcal{U}||\mathcal{Y}|)^K) + O((|\mathcal{U}||\mathcal{Y}|)^K)$$
$$= O(|\mathcal{X}|^2 (|\mathcal{U}||\mathcal{Y}|)^K).$$

The the second equation follows directly from both Lemma A2 and Lemma A3. □

5.7.1.1 SHOT time complexity

We start by restating the set of assumptions we assume for the following analysis (from Section 5.4.2). Let there exist a finite sequence of subsets optimal goal subsets $\mathcal{O}^g = \{\mathcal{O}_0^g, \mathcal{O}_1^g, \ldots, \mathcal{O}_K^g\}$. For each goal set $\mathcal{O}_k^g$, let there also exist a subset $\mathcal{O}_k^b$, which we denote as a basin subset. Let these subsets be defined such that for all k, $0 \leq k \leq K$:

1. Each of the subsets has measure, with $\mathcal{O}_k^g \subseteq \mathcal{P}_\beta$ and $\mathcal{O}_k^b \subseteq \mathcal{P}_\beta$;
2. The initial hyperbelief, be within $\mathcal{O}_0^b$, or $\beta_0 \in \mathcal{O}_0^b$;
3. The optimal goal set$\mathcal{O}_k^g$ is a subset of the basin set $\mathcal{O}_{k+1}^b$;
4. Each local policy π with the target hyperbelief $\beta \in \mathcal{O}_k^g$ that starts from $\beta^{start} \in \mathcal{O}_k^b$ will terminate with the final hyperbelief in $\mathcal{O}_k^g$;
5. Each local policy π with goal hyperbelief $\beta \in \mathcal{O}_k^g$ is invariant for any starting hyperbelief in $\mathcal{O}_k^b$: the exact same sequence of actions will be executed for each information vector;
6. If there exists a vertex β in the graph $\mathcal{G}$, whereby $\beta \in \mathcal{O}_{k-1}^g$, and a target hyperbelief β^{target} is sampled from $\mathcal{O}_k^g$, then the SHOT algorithm will select β as the source vertex to generate an edge between β and β^{target}.

Theorem 5.2. *Given a POMDP system with $p_\beta > 0$, the probability that the SHOT algorithm finds the optimal solution after n iterations is $1 - \frac{1}{2}e^{-2(np_\beta - K+1)^2/n}$, where $n \geq Kp_\beta$.*

Proof. The fifth assumption ensure that the optimal solution does not change for any hyperbelief in the sequence of goal sets. The second through fourth assumptions ensure that a path exists between optimal goal sets. The sixth assumption provides that if the SHOT graph G has a vertex $\beta \in \mathcal{O}^g{}_k$, then if a target hyperbelief β' is randomly sampled is in $\mathcal{O}^g_{k+1}$ the SHOT algorithm with connect $\beta \to \beta'$. The probability of sampling any of the goal sets at each stage is at least p_β. From the first assumption, p_β is nonzero.

Starting from the initial hyperbelief, the probability of connecting to $\mathcal{O}^g_1$ is at least p_β. Once the edge between β_0 and a hyperbelief sample in $\mathcal{O}^g_1$, which we denote as β_1, is added to the graph, the next step is connecting β_1 to $\mathcal{O}^g_2$. Again, the probability of sampling from $\mathcal{O}^g_2$ is at least p_β. This repeats until the entire path from 0 to K has been sampled. In the worst case, we can assume that any sample not within the next goal set will not be useful in finding the optimal solution. Thus, the extending the optimal path at iteration i to the next goal set is described by a Bernoulli random variable S_i: probability of success is p_β and, conversely, the chance of failure is $1 - p_\beta$.

The random process of describing the number successes in extending the optimal path after n iterations is

$$S = S_0 + S_1 + \cdots + S_n,$$

where the set of $\{S_i\}$ are i.i.d. Bernoulli distributions with parameter p_β. The random process describing the chance of extending the path K times to find the optimal path is a Binomial distribution:

$$p_S(S = K; n, p_\beta) \equiv \binom{n}{K} p_\beta^K (1 - p_\beta)^{n-K}.$$

Now to determine the total probability of having at least K successes in n iterations we note that the total probability of having at most $K - 1$ successes is

$$p_S(S \leq K-1; n, p_\beta) = \sum_{i=0}^{K-1} p_S(S = i; n, p_\beta) = \sum_{i=0}^{K-1} \binom{n}{i} p_\beta^i (1-p_\beta)^{n-i}.$$

Applying the Hoeffding's inequality, the bounds on the probability becomes

$$p_S(S \leq K-1; n, p_\beta) \leq \frac{1}{2} e^{-2(np_\beta - K+1)^2/n}.$$

The probability of having K or more successes is then

$$\begin{aligned} p_S(S \geq K; n, p_\beta) &= 1 - p_S(S \leq K-1; n, p_\beta) \\ &\geq 1 - \frac{1}{2} e^{-2(np_\beta - K+1)^2/n}. \end{aligned}$$

This inequality only holds when $n \geq Kp_\beta$. □

Corollary 5.1. *The expected number of stages for the SHOT algorithm to find the optimal solution is K/p_β.*

Proof. The expected number of successes of the SHOT algorithm is $E[S] = np_\beta$. Thus the number of iterations the algorithm must run to have an expected K successes is $n = K/p_\beta$. □

Lemma 5.1. *The SHOT algorithm executing for $n \leq |\mathcal{X}|^2 |\mathcal{Y}|^K$ iterations on a K stage finite-horizon optimization of a POMDP system with a uniform sampling method over the hyperbelief space has a time complexity of*

$$O(n|\mathcal{X}|^2 |\mathcal{Y}|^K)$$

if only a single vertex needs to be expanded at each iteration and, at each of the n iterations, a new policy edge is added to the graph. It is further assumed that the time complexity of selecting the vertex is negligible.

Proof. Using the same terminology for the exact solution in Theorem 5.1, the total time complexity is

$$\begin{aligned} T_{total} &= \sum_{i=1}^{n} [T_i^e + T_i^o] \\ &\leq \sum_{i=1}^{n} |\mathcal{X}|^2 |\mathcal{Y}|^K + \sum_{i=1}^{n} i \\ &\leq n|\mathcal{X}|^2|\mathcal{Y}|^K + n(n-1)/2 \\ &\leq n|\mathcal{X}|^2|\mathcal{Y}|^K + n^2 \\ &\leq 2n|\mathcal{X}|^2|\mathcal{Y}|^K. \end{aligned} \tag{5.6}$$

The first equation is the total sum of the time complexity for both the expansion and optimization of the SHOT algorithm. The time complexity of the expansion, T_i^e, at each iteration i, can be bounded above by the assuming that each hyperbelief has the maximum possible number of beliefs and that the cost of transitioning the hyperbelief is $|\mathcal{X}|^2$. We can bound the optimization cost, T_i^0, at each stage by observing that, in the worst case, each vertex in the graph has to be updated when a new vertex is added to the graph. The last equation holds if we assume that $n \leq |\mathcal{X}|^2|\mathcal{Y}|^K$.

□

Theorem 5.3. *For SHOT to have a lower time complexity cost than the exact solution, the minimal probability of sampling each goal set must exceed:*$p_\beta \geq \frac{K}{|\mathcal{U}|^{K/2}}$.

Proof. To determine the point at which SHOT no longer obtains a lower expected cost than the exact solution we solve for inequality between Lemma 5.1 and Theorem 5.1:

$$
\begin{aligned}
n|\mathcal{X}|^2|\mathcal{Y}|^K &\leq |\mathcal{X}|^2 \left(|\mathcal{U}||\mathcal{Y}|\right)^K \\
\frac{K}{p_\beta}|\mathcal{X}|^2|\mathcal{Y}|^K &\leq |\mathcal{X}|^2 \left(|\mathcal{U}||\mathcal{Y}|\right)^K \\
p_\beta &\geq \frac{K|\mathcal{X}|^2|\mathcal{Y}|^K}{|\mathcal{X}|^2 \left(|\mathcal{U}||\mathcal{Y}|\right)^K} \\
p_\beta &\geq \frac{K}{|\mathcal{U}|^K}.
\end{aligned}
$$

The second equation results from substituting n for the expected number of iterations assuming a planning horizon K. The third follows from solving for p_β. □

5.7.2 time complexity gains from approximation

We start by restating the assumptions for the following analysis (from Section 5.4.3). Let there exists an ϵ-goal set $\mathcal{O}_k^{g,\epsilon}$ around each goal set $\mathcal{O}_k^g$, such that the distance between any set of hyperbeliefs $\beta^1 \in \mathcal{O}_k^g$ and $\beta^2 \in \mathcal{O}_k^{g,\epsilon}$ is less than ϵ: $d(\beta^1, \beta^2) \leq \epsilon$. For each ϵ-goal set $\mathcal{O}_k^{g,\epsilon}$we define an ϵ-basin set $\mathcal{O}_k^{b,\epsilon}$ all $0 \leq k \leq K$. The ϵ-basin set is analogous to the original basin set but now it defines the region for which the local policy will terminate in the ϵ-goal set. Let ϵ_{min} represent the minimum distance of all ϵ-goal sets $\mathcal{O}_k^{g,\epsilon}$ for $0 \leq k \leq K$. The updated assumptions are as follows for all $0 \leq k \leq K$:

1. The ϵ-goal set $\mathcal{O}_k^{g,\epsilon}$ is a subset of the ϵ-basin set $\mathcal{O}_{k+1}^{b,\epsilon}$;

2. Each local policy π with the target hyperbelief $\beta \in \mathcal{O}_k^{g,\epsilon}$ that starts from $\beta^{start} \in \mathcal{O}_k^{b,\epsilon}$ will terminate with the final hyperbelief in $\mathcal{O}_k^{g,\epsilon}$;

3. For any target hyperbelief $\beta \in \mathcal{O}_k^{g,\epsilon}$, the POMDP system under the local policy π is dissipative: if the minimum distance from $\beta_k \in \mathcal{O}_k^{b,\epsilon}$ to $\mathcal{O}_{k-1}^{g,\epsilon}$ is d_k and the minimum distance from β_{k+1} to $\mathcal{O}_k^{g,\epsilon}$ is d_{k+1}, then $d_k \leq d_{k+1}$.

4. If there exists a vertex β in the graph $\mathcal{G}$, whereby $\beta \in \mathcal{O}_{k-1}^{g,\epsilon}$, and a target hyperbelief β^{target} is sampled from $\mathcal{O}_k^{g,\epsilon}$, then the SHOT algorithm will select β as the source vertex to generate an edge between β and β^{target}.

5. Each ϵ-goal set $\mathcal{O}_k^{g,\epsilon}$is strictly larger than the goal set $\mathcal{O}_k^g$: $\epsilon_{min} > 0$.

Theorem 5.4. *For the SHOT technique to achieve a nearly optimal policy with error* $0 < V_{err} \leq \xi\epsilon_{min}/d_{max}$ *using exact expansion of the hyperbelief, the total expected time complexity of*

$$O(\frac{\xi K^2}{V_{err}}|\mathcal{X}|^2|\mathcal{Y}|^K).$$

It is apparent that the time complexity savings for a specified error V_{err} *is strongly dependent on the time horizon and the Lipschitz bound.*

Proof. The total approximation error is equal to the sum of the error at each stage:

$$V_{err} = \sum_{k=0}^{K} c_k + \delta c_k - V^* = \sum_{k=0}^{K} \delta c_k.$$

We know that $\delta c_k = \xi d(\tilde{\beta}_k, \beta_k)$. Since by assumption 3, the local policy is dissipative, meaning $d(\tilde{\beta}_k, \beta_k) \leq d(\tilde{\beta}_k, \beta_{k+1})$, the total approximation error is

$$V_{err} = \sum_{k=0}^{K} \xi d(\tilde{\beta}_k, \beta_k) \leq K\xi d(\tilde{\beta}_0, \beta_0) = K\xi\epsilon$$

$$\frac{V_{err}}{K\xi} \leq \epsilon.$$

Taking the result from Corollary5.1 and substituting in the sampling error variable $p_{samp} = \epsilon_{samp}/d_{max}$ in for p_β, we obtain

$$\frac{K}{p_{samp}}|\mathcal{X}|^2|\mathcal{Y}|^K = \frac{K}{(\frac{V_{err}}{K\xi})}|\mathcal{X}|^2|\mathcal{Y}|^K = \frac{\xi K^2}{V_{err}}|\mathcal{X}|^2|\mathcal{Y}|^K$$

for the time complexity described in Lemma 5.1. □

Theorem 5.5. *The time complexity of the SHOT algorithm having executed for n iterations and using hyper-particle filtering with m beliefs is of the order $O(nm|\mathcal{X}|^2|\mathcal{Y}| + n^2)$.*

Proof. If we substitute $n|\mathcal{X}|^2|\mathcal{Y}|$ into equation (5.6) from Lemma 5.1, we obtain

$$nm|\mathcal{X}|^2|\mathcal{Y}| + n^2.$$

□

Theorem 5.6. *Let the hyper-particle filtering approximation be ϵ_{hpf}, then the time complexity of generating a policy with the approximated value error V_{err} is*

$$O\left(K^2(\frac{K\xi}{V_{err}} - \frac{1}{\epsilon_{hpf}})^2 d_{max}^2 m|\mathcal{X}||\mathcal{Y}|\right),$$

if the sampling approximation error ϵ_{samp} satisfies $\epsilon_{samp} \leq \epsilon_{min} - \epsilon_{hpf}$.

Proof. If the hyper-filtering approximation is ϵ_{hpf} and the sampling approximation is ϵ_{samp}, then the total distance from the goal set is less than $\epsilon_{samp} + \epsilon_{hpf} = \epsilon_{err}$. For the system to obey assumption 2, a hyperbelief sample $\beta_k \in \mathcal{O}_k^{b,\epsilon}$ will need to terminate within $\mathcal{O}_{k+1}^{b,\epsilon}$ under the local policy π with the target hyperbelief in $\mathcal{O}_{k+1}^{g,\epsilon}$. Therefore, the approximate basin region must be at least ϵ_{err} for the assumptions to hold. The total error in the value function is proportional to the distance from the goal set at each stage and the Lipschitz bound (as established in Theorem 5.4):

$$V_{err} \leq K\xi\epsilon_{err} = K\xi(\epsilon_{samp} + \epsilon_{hpf}).$$

Solving for ϵ_{samp} in the above equation we obtain

$$\epsilon_{samp} \leq \frac{V_{err}}{K\xi} - \epsilon_{hpf}.$$

Since the probability of sampling any of the goal sets is $p_{samp} \geq \epsilon_{samp}/d_{max}$, we can substitute

$$p_{samp} \geq \frac{V_{err}}{K\xi d_{max}} - \frac{\epsilon_{hpf}}{d_{max}}$$

then the expected number of iterations to find the V_{err} optimal solution is K/p_{samp}, or

$$K(\frac{K\xi}{V_{err}} - \frac{1}{\epsilon_{hpf}})d_{max}.$$

Substituting the above as n in Theorem 5.5, we can determine the total expected time complexity:

$$\begin{aligned} n(m|\mathcal{X}|^2|\mathcal{Y}| + n) &= K(\frac{K\xi}{V_{err}} - \frac{1}{\epsilon_{hpf}})d_{max}\left(m|\mathcal{X}||\mathcal{Y}| + K(\frac{K\xi}{V_{err}} - \frac{1}{\epsilon_{hpf}})d_{max}\right) \\ &\leq K^2(\frac{K\xi}{V_{err}} - \frac{1}{\epsilon_{hpf}})^2 d_{max}^2 m|\mathcal{X}||\mathcal{Y}|. \end{aligned}$$

□

Corollary 5.2. *Let a POMDP system has hyper-particle filtering convergence rate*

$$\epsilon_{hpf} \leq \frac{C_{sys}}{m},$$

where C_{sys} is a system dependent constant term. The total time complexity of the SHOT algorithm relative to the number of hyper particle samples m and the desired error $0 < V_{err} \leq \epsilon_{max}/d_{max}$, cost function Lipschitz bound ξ and time horizon K is

$$O\left(K^2(\frac{K\xi}{V_{err}} - \frac{m}{C_{sys}})^2 d_{max}^2 m|\mathcal{X}||\mathcal{Y}|\right).$$

Proof. Since we are assuming a finite observation space and a finite time horizon, all hyperbeliefs will consist of a finite number of beliefs. Thus, we know for any $\epsilon_d \geq 0$ there must exist m, such that the metric distance is between the hyperbelief approximated with

m samples is less than ϵ_d. □

5.7.3 Temporal abstraction

Theorem 5.7. *Let the minimal temporal abstraction be t stages, so that instead of K optimal goal sets there are $\lceil K/t \rceil$. Let all the other assumptions still hold. The expected number of stages to find the optimal solution reduces to $\frac{K}{tp_\beta}$ and the total expected time complexity reduces to*

$$O\left(\frac{K}{tp_\beta}|\mathcal{X}|^2|\mathcal{Y}|^K + \left(\frac{K}{tp_\beta}\right)^2\right).$$

Proof. Following the same steps in Theorem 5.1 the total time complexity of the temporally abstracted solution is

$$\begin{aligned} T_{total} &= \sum_{i=1}^{n} [T_i^e + T_i^o] \\ &\leq \sum_{i=1}^{n} tm|\mathcal{X}|^2|\mathcal{Y}| + \sum_{i=1}^{n} i \\ &\leq nmt|\mathcal{X}|^2|\mathcal{Y}| + n^2. \end{aligned} \tag{5.7}$$

The second equation for the cost of expansion is where the time complexity changes: instead of just executing a single stage, each policy executes for t stages. Taking the expectation relative to $n = \frac{K}{tp_\beta}$ of (5.7), the total expected computation complexity to find the optimal solution becomes

$$O\left(\frac{K}{tp_\beta}m|\mathcal{X}|^2|\mathcal{Y}| + \left(\frac{K}{tp_\beta}\right)^2\right).$$

□

Chapter 6

PERTURBATION ANALYSIS OF THE FORECASTED EVOLUTION OF POMDPS

We derive a representation of the sensitivity of partially observed Markov decision processes (POMDPs) to determine the effect a perturbation on the starting location of the policy has on the forecasted evolution its associated running cost. Leveraging this analysis, we develop a methodology to chain forecasted evolution results together. At the cost of approximation error, the chained representation eliminates the need to simulate each of the component forecasted evolutions.

We formulate our technique and demonstrate analysis on standard benchmark problems from the POMDP literature. Our method will be presented in Section 6.3. First we provide background information: the formulation of POMDPs and related research in Section 6.2. Examples are provided in Section 6.5. We conclude with some future directions and final remarks in Section 6.6.

6.1 Introduction

Partially observable Markov decision processes (POMDPs) provide a general and expressive formalism for representing uncertainty in both the execution of actions and observations from sensor data. However, their use for planning for uncertain systems has been limited due to problems of scalability and tractability. The time complexity of simulating policies for discrete-time and -space POMDPs is a direct result of size of the state space compounded by the number of possible observations. These two factors are the root cause of the intractability of determining optimal policies for general POMDP systems [2].

When described in terms of a system's state space, the evolution of a POMDP is governed by a set of transition probabilities that describe the effects of control actions, and an observation model that specifies uncertainty in the sensing process. If, instead, the system is described in terms of the belief space (i.e., the space of possible a posteriori probability functions on the state space), the evolution of the system can be modeled as a Markov decision process (MDP). This corresponds to lifting the system description from a lower dimensional state space to a higher dimensional belief space. Most POMDP algorithms operate, and approximate the system, at the level of the belief space.

Planning in the belief space amounts to constructing a belief-space policy (i.e., mapping from beliefs to control inputs), which in turn requires the ability to forecast the evolution of policies into the future. Forecasting the exact behavior of a POMDP is intractable. The number of possible beliefs grows exponentially in the time-horizon, which necessitates the use of approximation methods. Ensemble forecasting is often used in POMDP research to approximate the forecasted evolution, which is a collection of sample paths. Ensemble forecasting generates a small, but representative, collection of sample paths instead of the exhaustive set of possibilities. A sample path is a sequence of beliefs, $b_0, ..., b_k$, defined by specific sequence of observations and control inputs. POMDPs evolve randomly in the belief space as a function of the uncertainty in the observation. A sample path is instantiated by iteratively generating, for a fixed number of stages, a deterministic control action and a random observation. This process is repeated starting at a specified initial belief b_0 to generate a collection of sample paths that approximate the forecasted evolution.

Simulation using ensemble forecasting requires significant computation. Each sample path has a time complexity that is, primarily, linear on the planning horizon and quadratic in number of states in the POMDP system.[1] The overall time complexity of ensemble forecasting is the product of time complexity of each sample path and the number of sample paths. If numerous simulations must be performed to evaluate and/or determine a pol-

[1] Approximate filtering methods can reduce the time complexity to be sub-quadratic [57].

icy, it would be advantageous to derive methods to reuse simulated results because of the computational requirements.

In this chapter, we do exactly that and derive a method to reuse existing simulations via a perturbation analysis approach. The contributions of our formulation are

1. a representation of the sensitivity analysis of the running cost[2] of a policy in the neighborhood around the starting belief b_0;

2. an approximation of the forecasted evolution at the terminal stage b_k due to a perturbation in the initial belief b_0; and

3. the framework to chain forecasted evolutions together that does not require re-simulation.

The third contribution is beneficial when numerous forecasted evolutions from various initial beliefs must be performed. If it is detected that a terminal belief b_k of one forecasted evolution, which starts at b_0, is in the neighborhood of an initial belief b_0' of an existing simulation, we can use the results of the existing simulation to approximate the forecasted evolution of the policy from the terminal belief b_k. When forecasting we simulate forward in time. However, when optimizing, using a dynamic programming approach, analysis moves backwards in time. In the simulation case, this memoization reduces the running time by avoiding re-simulation. In the backward optimization scenario, this enables effective estimation of the cost-to-go of a reference policy starting at some initial belief based on the cost-to-go of the forecasted evolution of neighboring policies.

Our approach derives the sensitivity using the ensemble forecasting results. For all sample paths, both sensitivity results for the running cost along the path and the perturbation to the terminal belief are studied. By evaluating the effect of these perturbations over the collection of sample paths, we estimate the the average running cost and an approximate distribution over the terminal beliefs. The perturbation analysis we derive collapses the

[2]The term *cost-to-go* to represents the cost of the system evaluated from some stage k to the end of the time horizon K. The term *running cost* denotes the cost from some initial stage 0 to some intermediate stage $k < K$.

stage-wise representation of each of the sample paths to represent Δb_k in terms of Δb_0. This implies that we can simulate the effect of a perturbation of Δb_0 has on Δb_k in one iteration instead of k iterations.

Obtaining a reduction in the running cost over the set of sample paths under a perturbation is more complex than just the deriving the perturbed beliefs for each sample path. The expectation taken over the collection of sample paths is coupled to the running cost along each path. Because the probability of each path may vary for a perturbation, there is a coupling of a normalizing constant to each of the otherwise independent sample paths. However, we use a generalized version of the chain rule to approximate this relationship to obtain a collapsed representation along the set of sample paths.

The collapsed representation we derive enables the chaining of the existing forecasted evolutions without requiring re-simulation. Each path of a new forecasted evolution is joined to an existing forecasted evolution. Using the same technique that enables us to collapse sample paths and running costs, each of the new sample paths is joined to the existing forecasted evolution. The total number of stages increases to the sum of the two paths, but the perturbation of the running cost and terminal belief is collapsed and can be determined in a single iteration. This second level abstraction enables efficient analysis of the behavior of a system as long as a sufficient number of forecasted evolutions exist at a representative sampling of initial beliefs. This process can be recursively applied to obtain an approximation a chained forecasted evolution over a series of existing forecasted evolutions.

There are various considerations that must be weighed when considering our approach. We derive the sensitivity of the forecasted evolution of a system using a static set of sample paths. It is not only possible that the actions executed by a policy change due to a perturbation in the initial belief, but it is also almost certain that the likelihood of the sampled observations that generate the sample path will occur with differing probability. Thus, it is possible that the sampled set of observations from an unbiased simulation would change as well. However, we postulate that a sufficient number of possible paths for the policy are

captured for the neighborhood around each initial belief if a representative set of sample paths are generated by the ensemble forecasting. In essence, we are assuming that there is no chaotic behavior in the POMDP dynamics when coupled to the control policy. If this is the case, then a perturbation can reasonably be approximated by a shift in the probabilities along each sample path.

Our belief is that our methodology will prove valuable to POMDP temporal and spatial abstraction optimization techniques [37, 44–49, 56]. Such approaches create global, optimal policies from local policies, and, as such, are hierarchical techniques. By compressing time and space, these optimization methods attempt to quickly explore the value landscape. Recent inroads in solving large, continuous, and multi-agent POMDP systems using temporal and spatial abstraction methods demonstrate their potential [48, 81].

The perturbation analysis we present should enhance temporal and spatial abstraction techniques by providing analogous gains of the α-vector approach over point-based method of solving POMDPs. The α-vector representation determines the cost-to-go at each sampled point as a linear function instead of a constant value. This linear representation of the value function provides better estimation of the value during each backup stage. The benefits of the α-vector approach in sampling based methods was first demonstrated in [6].

6.2 Related Research: Simulation and Optimization of POMDPs

Our approach is built upon a rich set of existing methodologies that are pervasive in the literature. In particular, our approach sits at the intersection of three primary areas of research: stochastic simulation, perturbation/sensitivity analysis, and POMDP optimization. We will concisely discuss related research in these three fields.

6.2.1 Stochastic simulation

For systems subject to uncertainty, stochastic simulation techniques may either estimate of a system's state, known as filtering, or predict the future evolution, known as forecasting. Early filtering techniques relied on mathematical properties of stochastic systems to derive closed-form solutions. By exploiting system dynamics and uncertainty models that result in conjugate distributions, solutions to the filtered state are readily obtained, in closed-form, and can be iteratively updated. The most widely cited example of such an approach is Kalman filtering [16]. Kalman filtering assumes a linear Gaussian system. As such, the predicted state at each stage is described by a Gaussian distribution over the state. However, the restrictions placed on the system dynamics and uncertainty are often too severe. Researchers expanded Kalman filtering to nonlinear systems (refer to [17]) and later to multi-modal uncertainty via mixture of Gaussians [18]. These and related projection-based approaches, e.g. [22–24, 82], are referred to as parametric approaches because they restrict the uncertainty model to a parametrized family of distributions.

Nonparametric methods make no such restrictions about the family of distributions and are often sampling based. Based on Monte Carlo methods, sequential Monte Carlo or particle filtering [58–69] recently has come to dominate the stochastic simulation literature. By retaining a set of samples, a point-based simulation is possible, which alleviates the difficulty of propagating distributions through the process model. However, sampling from arbitrary distributions may not be feasible and significant research has focused on sampling methods (see [57]). Provable convergence under general conditions [69] and their ease of implementation contributed to the quick adoption of particle filtering.

Our approach utilizes a nonparametric method for forecasting the behavior of a system. The method we use expands the particle filtering framework to forecasting. A description of this approach is provided in Section 4.1. The method we present is not limited to this method of forecasting; other forecasting methods that generate sample paths may be adapted to our

framework. Our method relies on a sequential chain of actions and observations to represent one possible path, whereby the entire forecasted evolution comprises a set of sample paths.

6.2.2 Perturbation analysis

Perturbation analysis computes power series expansions around a small parameter to obtain approximations of systems that cannot be solved exactly (refer to [83]). Function approximation using Taylor's series is related in the sense of approximating an arbitrary function using a power series. The Stone-Weierstrass theorem [84] established that any continuous function on a bounded interval can be uniformly approximated to any precision by a polynomial. Local, or neighborhood, sensitivity analysis (refer to [85]) evaluates the derivative at a sample point to estimate the effect a small change has on the output of the system. In this way, the effect of uncertainty in the input on the output can be measured. Such analysis is a first order formulation of perturbation analysis where the parameter of interest is the input variable.

Significant research has been invested to study the sensitivity of the stationary point of Markov chains and semi-Markov chains to perturbations in the transition matrix [86,87] or to the total cost [88]. Research into hidden Markov models, uncontrolled variants of POMDPs, analyze the the Lyapunov exponents of hidden Markov models to determine their forgetting, i.e. reduction of the effect of a perturbation on the initial condition [89] and entropy rate [90], which is the average entropy and is a function of the stationary distribution. Much of the research into sensitivity analysis of POMDPs has been directly towards policy gradient methods [91–93]. As an extension of policy iteration, policy gradient methods derive an approximate derivative over the policy space. Using standard optimization methods like gradient descent, a locally optimal policy can be attained. Using bounds on the parameters defining a POMDP, namely the transition, observation, and cost functions, Ross et al. in [94] explore the effect such changes have on the optimal policy.

The technique we present focuses on the effect of a perturbation of the initial belief for a given policy. The effect of a perturbation is not only informative in understanding the sensitivity of the transition function under a given policy, but crucial in composition of policies.

6.2.3 POMDP optimization

The two canonical methods for finding optimal policies are value iteration, a dynamic programming (DP) approach, and policy iteration. Value iteration operates backwards, propagating optimal *value* or cost-to-go, from the terminal stage to the initial stage. Conversely, policy iteration operates in the policy space rather than on the belief (or state) space. It begins with an initial policy and at each iteration searches for changes to the policy that improve performance. With either algorithm, finding the exact solution is intractable [32], with a best known time complexity that is exponential in the time horizon K, i.e. the number of stages into the future optimize considers.

However, finding policies for partially observed systems is desired for many real-world problems, so researchers have focused on finding efficient approximation methods to solve POMDP models. Many recent approximation methods sample points in belief space and use the Bellman equation to compute the value function over this subset of the belief space. Thrun in [5] proposed a sampling based on Monte Carlo integration to estimate the one stage Bellman backup. Soon after, this method was augmented by retaining a linear function around each sampled belief (referred to as the α-vector) in [6]. Because of the better representation of the value function over the entire of the belief space, this technique was adopted in most of the subsequent sampling-based methods, e.g. [6, 8, 10, 11].

Sampling-based methods, however, suffer from the computational burden of performing the Bellman backup for a single stage. This requires a significantly more dense sampling of the belief space to obtain a representative set of meaningful beliefs. To overcome this limi-

tation, temporal and spatial abstraction techniques have been developed. Researchers first developed the notion of temporal abstraction for optimizing options (policies) for Markov decision processes [50–55]. Methods to build a sampling-based abstraction for partially observed systems soon followed [37,44–49,56]. While varied in their approaches and aims, each of these methods use a hierarchical representation to sparsely sample both the temporal and spatial domain of POMDP systems.

The approach we present in Section 6.3, attempts to bring the same gains of α-vectors to spatially and temporally abstracted techniques. Furthermore, the sensitivity analysis developed may further enhance active learning techniques that use adaptive sampling to effectively explore the belief space Chapter 7.

6.3 Perturbation Analysis

The technique we present enables simulations from various initial positions in the belief space to be generated and then connected together to build chained simulations. Because it is unlikely that any terminal belief of any simulation will coincide exactly with the starting beliefs of other simulations, we characterize the effect a perturbation from the starting belief has on both the running cost of a policy as well as the set of terminal beliefs. Using this result, we can chain forecasted evolutions together regardless of whether the terminal beliefs of one simulation attain the starting beliefs of other simulations. We note, however, that the effectiveness of the chaining process is directly related to the richness of the set of sampled simulations, including the locations of initial belief; a poorly sampled set of initial beliefs will produce poor approximations of chained simulations.

On the surface, such an approach may appear dubious: it seems the error will compound from one stage to the next, from simulation to simulation, and, thus, the resulting error will grow rapidly, quickly rendering the analysis meaningless. This, however, is not the case. Both Markov chains as well as hidden Markov models, which consider partial observability

for uncontrolled systems, are naturally insensitive. Insensitivity in our context refers to systems for which there is a non-strict contraction of the distance between pairs of beliefs from one stage to the next. This definition encapsulates both conservative and dissipative systems. Locally, insensitivity refers to systems for which the non-strict contraction only holds in the neighborhood around some reference belief. Global insensitivity holds when the non-strict contraction is observed for any pair of beliefs in the belief space. Irreducible ergodic[3] Markov chains (MCs) and HMMs are globally insensitive.

To establish the global insensitivity of MCs and HMMs, we must first define the measure for which the contraction is evaluated. Information theoretic analysis often uses the Kullback-Leibler divergence, or relative entropy, as a measure in the belief space.

The Kullback-Leibler (KL) divergence between two beliefs b and b', both from $\mathcal{P}_b$ is defined as

$$d_{KL}(b'||b) = \sum_x b'(x) \ln \frac{b'(x)}{b(x)},$$

where $b(x) > 0$ for any x such that $b'(x) > 0$.

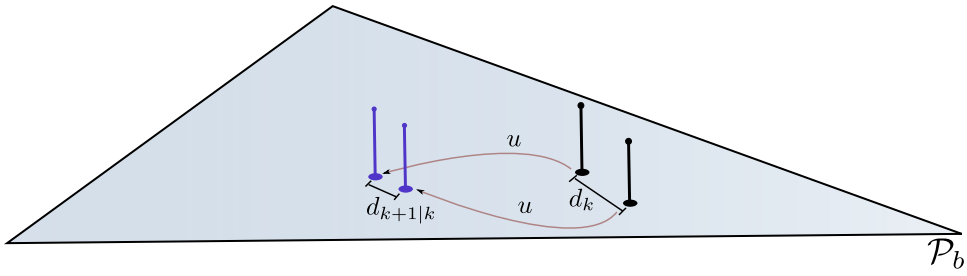
KL-divergence is a non-symmetric measure of the similarity of two probability functions.

For Markov chains (see [95, pg. 34]), it has long been established that the KL divergence between beliefs is greater or equal to the KL-divergence of their predicted beliefs:

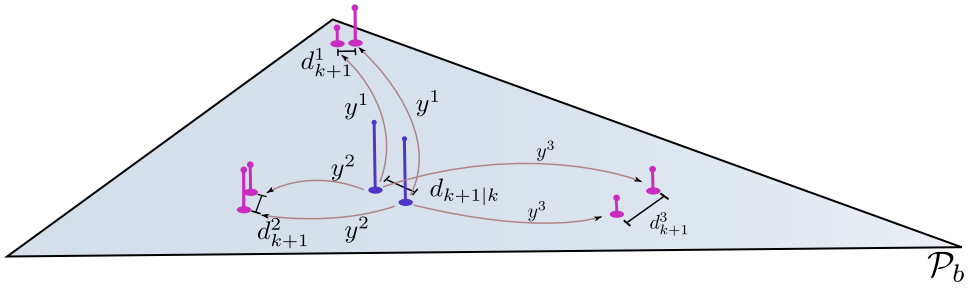
$$d_{KL}(b'||b) \geq d_{KL}(T_u b'||T_u b).$$

This property holds for any transition function T_u that is ergodic and for any pair of beliefs b' and b in the belief space. Because of this property, every ergodic Markov chain has a stationary distribution. An illustration of this is depicted in Figure 6.1a. KL-divergence between each pair of updated beliefs taken over all observations will not increase relative to

[3] A Markov chain is said to be ergodic if every state is aperiodic (the transition back to a state occurs at irregular times) and positive recurrent (transition back to a state occurs with positive probability in finite time).



(a) The distance between beliefs at stage d_k can only decrease after prediction, i.e, $d_{k+1|k} \leq d_k$



(b) The expected distance d_{k+1} of the updated hyperbelief (taken with respect to the random observations) can only be less than $d_{k|k+1}$

Figure 6.1: Example of convergence properties of Markov processes

the KL-divergence between two beliefs b and b' :

$$d_{KL}(b'||b) \geq E_y \left[d_{KL} \left(\frac{O_{\mathbf{y}} b'}{\mathbf{1}^T O_{\mathbf{y}} b'} || \frac{O_{\mathbf{y}} b}{\mathbf{1}^T O_{\mathbf{y}} b} \right) \right].$$

In Figure 6.1b the updated beliefs for several observations are illustrated. While the distance may grow between updated beliefs for a particular observation (as shown for y^3), the expected distance taken over all possible observations is non-increasing. The combination of these two results implies that expectation over all pairs of future beliefs is non-strictly decreasing from one stage to the next:

$$d_{KL}(b'_k||b_k) \geq E_{y_{k+1}} \left[d_{KL} \left(\frac{O_{\mathbf{y}_{k+1}} T_{u_k} b'}{\mathbf{1}^T O_{\mathbf{y_{k+1}}} T_{u_k} b'} || \frac{O_{\mathbf{y}_{k+1}} T_{u_k} b}{\mathbf{1}^T O_{\mathbf{y}_{k+1}} T_{u_k} b} \right) \right] \tag{6.1}$$

$$= E_{y_{k+1}} [d_{KL} \left(\phi(b'_k, u_k, \mathbf{y}_{k+1}) || \phi(b_k, u_k, \mathbf{y}_{k+1}) \right)]. \tag{6.2}$$

Boyen and Koller [24] build on these two properties to define a minimal mixing rate

between states in the transition model of a dynamic Bayesian network, which is a generalized version of an HMM. All properties they prove extend to HMMs, and POMDPs are controlled HMMs. They then show that if the minimal mixing rate is nonzero, then the expected distance between two initial beliefs will strictly decrease from one stage to the next. Methods that derive a similar notion of mixing demonstrate the exponential uniform convergence (or forgetting) of the initial belief [96]. The mixing assumption of Boyen and Koller is quite strong. Weaker assumptions including pseudo-mixing in [97] were established that broadened the analysis to more general systems. Our approach was developed to take advantage of these properties.

When considering controlled partially observed systems (i.e., POMDPs), we must also consider the effect of the policy on the insensitivity of the system's evolution. Fortunately, it is often the case that feedback policies attenuate disturbances such as variation in the initial condition. The coupling of the inherent insensitivity of stochastic systems with the attenuating property of feedback policies supports the notion that perturbations Δb_k from the reference belief b_k at each stage k will in fact decrease from stage to stage along a sample path, so that $d(b_k||b_k + \Delta b_k) \leq d(b_{k-1}||b_{k-1} + \Delta b_{k-1})$.

While feedback policies typically attenuate disturbances, bifurcations in the forecasted evolution are still possible for small perturbations. Bifurcation occurs when sample paths between two beliefs diverge because the policy executed by the system selects actions that pull the beliefs at each stage further from one another. Consider the example illustrated in Figure 6.2. The switching surface between u_4 and u_7 is shown. While the initial action for both b_k and b'_k were both u_7, the action changes between b'_{k+1} and b_{k+1}. Because there may exist an arbitrary region for each control action, it is possible that the paths between b'_0 and b_0 to diverge when executing a general feedback policy.

However, such bifurcations will often only occur in a small set of regions of the belief space. For the duration of our analysis we will assume these regions are negligible and that, for some specified policy, local insensitivity holds within some measurable neighborhood

around every belief in the belief space. In Section 5.5, experimental results of the sensitivity of various benchmark problems will be presented to support this claim.

The policy set Π comprises all policies π for a given POMDP system, with belief transition function ϕ (refer to (2.7) in Section 2.1), such that, for some $\delta > 0$,

$$E_{y_{k+1}}[d(\phi(b'_k, \pi(b'_k), \mathbf{y}_{k+1}) \,||\, \phi(b_k, \pi(b_k), \mathbf{y}_{k+1})] \leq d(b'_k||b_k), \text{ or}$$
$$E_{y_{k+1}}[d(b'_{k+1} \,||\, b_{k+1})] \leq d(b'_k||b_k)$$

for all $b \in \mathcal{P}_b$ and $b' \in \mathcal{P}_b$ such that $d(b'||b) \leq \delta$. The measure $d(\cdot||\cdot)$ is KL-divergence or another appropriately defined measure over the belief space.

By restricting policies thusly we adapt the contraction of the expected distance of beliefs under the same action in (6.2) to the contraction of the expected distance of beliefs under the same policy. This assumption eliminates the possibility of bifurcations occurring for a policy within some local neighborhood, which ensures that insensitivity is retained. The results obtained by our perturbation analysis and chaining process, therefore, will be representative of the chained forecasted evolution. This is a strong assumption/limitation. Relaxing this restriction is an important future step for this research.

Instead of restricting the policy space, we could perform perturbation analysis on the policy itself. Policy gradient methods [91–93] are one such approach. These methods modify the policy to be a continuous function of the belief space. This formulation would allow us

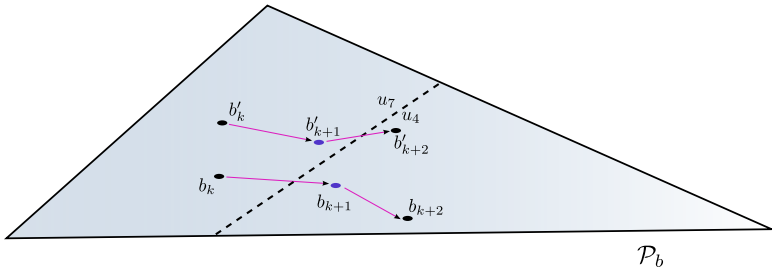


Figure 6.2: Switching surface for two actions resulting in a change in policy for a large enough perturbation to the initial belief b_k

to perform perturbation analysis on the policy itself, which could then be integrated into the perturbation analysis we present below.

We present the perturbation analysis in three parts. First, we determine the sensitivity of the evolution of the system as represented by a set of sample paths. Next, we determine the sensitivity of the running cost for a sample path set. Finally, we outline the method to chain forecasted evolutions together, so we can represent the running cost along an arbitrarily long chain of forecasted evolutions.

6.3.1 Policy sensitivity

As a forecasted evolution is being simulated, a set of representative sample paths under a given policy originating from the initial belief is generated. Each sample path represents a possible evolution of the system. In our analysis, we use hyper-particle filtering [56][4] to generate the sample paths. Each k-stage sample path is characterized by an information vector $I_k^{(i)} = \{u_0^{(i)}, u_0^{(i)}, \ldots, u_{k-1}^{(i)}, u_k^{(i)}\}$.[5] We use the notation $I_k^{(i)}$ to denote the information vector at stage k of the i-th sample path in the ensemble forecast. Each stage of evolution appends an action generated by policy π and observation to the information vector.

The complete set of sample paths $\mathcal{I}_k$ from b_0 under π represents the complete set of possible evolutions of the POMDP system. There are in actuality $|\mathcal{Y}|^k$ possible sample paths (if all actions are considered the number grows to $(|\mathcal{U}||\mathcal{Y}|)^k$. We will assume, henceforth, that a small subset of possible paths has been simulated, which we will denote as $\tilde{\mathcal{I}}_k$. The set of information vectors represents likely sample paths the system follows under the policy π. Thus, we define a forecasted evolutions as the collection of sample paths $\tilde{I}_k$. As an example, in Figure 6.3, the information vector $I_4^{(3)}$ evolves from b_0 under $\{u_0^{(3)}, y_1^{(3)}, \ldots, y_4^{(3)}\}$ to terminate at $b_4^{(3)}$.

We desire a representation of the effect of a perturbation to an initial belief $b_0^{(i)}$ has on

[4] An overview of hyper-particle filtering is provided in Section 4.1.

[5] We remove b_0 from the information vector because we will be analyzing the effect perturbations Δb_0 to the sample path represented by $I_k^{(i)}$.

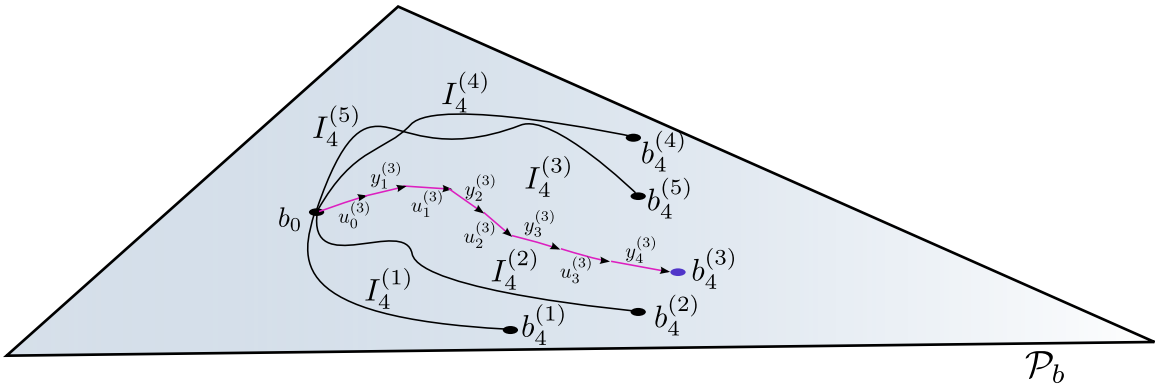


Figure 6.3: Forecasted evolution as represented by a series of sample paths $\{I_4^{(i)}\}_i = \tilde{\mathcal{I}}_4$, which is generated from vertex b_0 under policy π

the belief $b_k^{(i)}$ for the sample path represented by $I_k^{(i)}$ for all $I_k^{(i)}$ in $\tilde{\mathcal{I}}_k$. To achieve this we will first determine, the effect a perturbation of a belief on the next stage, e.g. the effect $\Delta b_{k-1}^{(i)}$ has on $\Delta b_k^{(i)}$. Next, we use this result to derive a formulation of $\Delta b_k^{(i)}$ directly in terms of $\Delta b_0^{(i)}$. Finally, we derive the effect a perturbation in the initial belief has on the probability of a sample path occurring.

The sensitivity Δb_k, for a single stage, due to perturbation Δb_{k-1} to b_{k-1}, subject to control action u_{k-1} and observation y_k is

$$b_k + \Delta b_k = \frac{\phi_{y_k,u_{k-1}}(b_{k-1} + \Delta b_{k-1})}{\mathbf{1}^T \phi_{y_k,u_{k-1}}(b_{k-1} + \Delta b_{k-1})} \tag{6.3}$$

$$\Delta b_k = \frac{\phi_{y_k,u_{k-1}}(\mathbf{I} - \frac{b_k \mathbf{1}^T \phi_{y_k,u_{k-1}}}{\mathbf{1}^T \phi_{y_k,u_{k-1}} b_{k-1}})\Delta b_{k-1}}{\mathbf{1}^T \phi_{y_k,u_{k-1}}(b_{k-1} + \Delta b_{k-1})}. \tag{6.4}$$

We arrive at this simply by substituting b_{k-1} with $b_{k-1} + \Delta b_{k-1}$ and b_k with $b_k + \Delta b_k$ into (2.7), from Section 2.1, and then solve for Δb_k. The derivation of this result is in Remark 6.1 from Section 6.7.1.

Because the transition function for any observation and action is a ratio of linear functions we can recursively represent the evolution along a sample path represented by the information

vector I_k at stage k as

$$b_k = \frac{\phi_{I_k} b_0}{\mathbf{1}^T \phi_{I_k} b_0}, \tag{6.5}$$

where the composite belief transition function is defined as

$$\phi_{I_k} = \phi_{y_k,u_{k-1}} \phi_{y_{k-1},u_{k-2}} \cdots \phi_{y_1,u_0}.$$

This is established in Proposition 6.1 in Section 6.7.1. To derive the perturbation at stage k relative to the initial stage 0, we substitute $b_0 + \Delta b_0$ for b_0 and $b_k + \Delta b_k$ for b_k into (6.5) and solve for Δb_k to obtain:

$$\Delta b_k = \frac{\phi_{I_k}(\mathbf{I} - \frac{b_0 \mathbf{1}^T \phi_{I_k}}{\mathbf{1}^T \phi_{I_k} b_0}) \Delta b_0}{\mathbf{1}^T \phi_{I_k} (b_0 + \Delta b_0)}. \tag{6.6}$$

The denominator in (6.5) is a normalization factor. If we neglect this normalization term and instead propagate the unnormalized perturbation $\bar{b}_k$, where $b_k = \frac{\bar{b}_k}{\mathbf{1}^T \bar{b}_k} = \frac{\phi_{I_k} b_0}{\mathbf{1}^T \phi_{I_k} b_0}$, we observe that the projection is a linear function. Essentially, we analyze the evolution of the system in the Euclidean space the belief space is embedded within, rather than projecting $\bar{b}_k$ onto the simplex at every stage along the sample path, we delay normalization until stage k. This notion is illustrated in Figure 6.4, for a simple example comprising three states. As we can see in the figure the vector $\bar{b}_k = \phi_{I_k} b_0$ transforms b_0 to a new location $\bar{b}_k \in \mathbb{R}^3$. However, the predicted belief is not a true probability function. In fact, it will necessarily sum to less than one (the eigenvalues of both T_u and O_y are less than or equal to one for all $u \in \mathcal{U}$ and $y \in \mathcal{Y}$). Normalization projects the predicted belief onto the belief simplex to produce b_k.

We can extend this insight to (6.6). Neglecting the denominator, the unnormalized perturbation becomes

$$\Delta \bar{b}_k \;=\; \phi_{I_k} \left(\mathbf{I} - \frac{\bar{b}_0 \mathbf{1}^T \phi_{I_k}}{\mathbf{1}^T \phi_{I_k} \bar{b}_0} \right) \Delta b_0$$

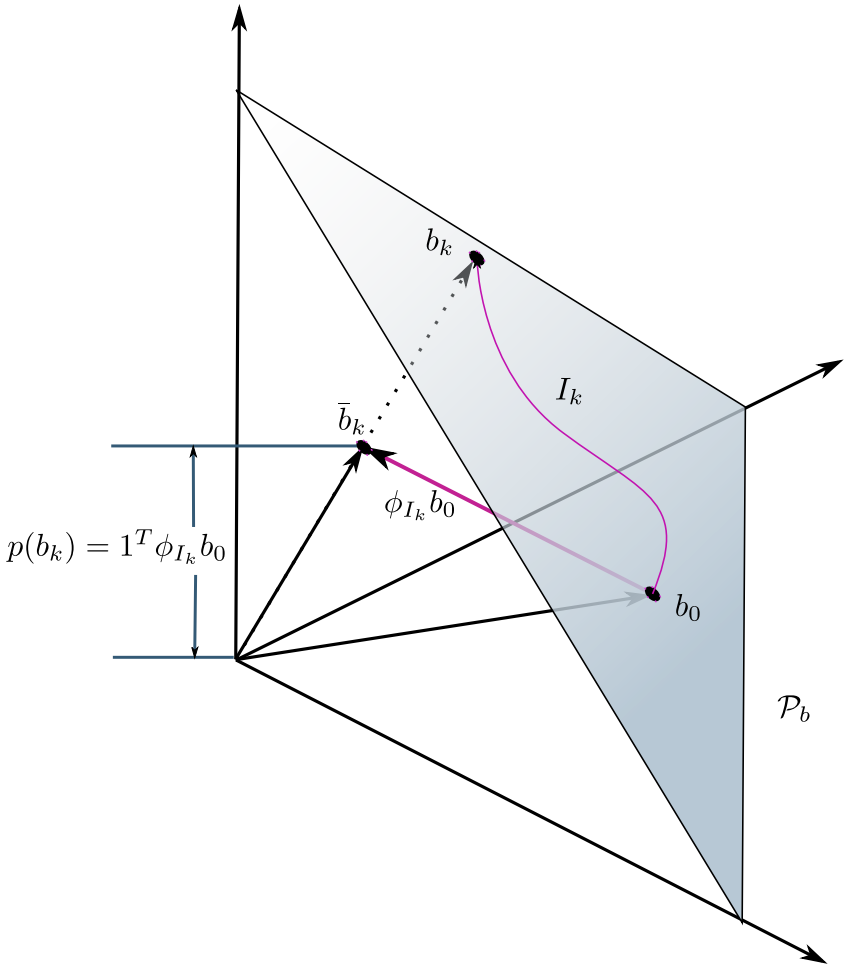


Figure 6.4: Delayed projection of predicted beliefs onto the belief simplex

When propagating the unnormalized perturbation, we can scale the perturbation arbitrarily, i.e, $b_k = \frac{\alpha \bar{b}_k}{\mathbf{1}^T(\alpha \bar{b}_k)} = \frac{\bar{b}_k}{\mathbf{1}^T \bar{b}_k}$, where, $\alpha \neq 0$. Scaling by $\mathbf{1}^T \phi_{I_k} b_0$, so that

$$\begin{aligned} \Delta b_k &\approx \frac{\phi_{I_k}(\mathbf{I} - \frac{b_0 \mathbf{1}^T \phi_{I_k}}{\mathbf{1}^T \phi_{I_k} b_0})}{\mathbf{1}^T \phi_{I_k} b_0} \Delta b_0 \\ &= \underbrace{\frac{\phi_{I_k}}{\mathbf{1}^T \phi_{I_k} b_0}(\mathbf{I} - \frac{b_0 \mathbf{1}^T \phi_{I_k}}{\mathbf{1}^T \phi_{I_k} b_0})}_{\nabla_{b_0} \eta(\phi_{I_k} b_0)} \Delta b_0, \end{aligned} \tag{6.7}$$

brings the unnormalized delta closer to the surface of the simplex and, thus, the perturbed belief nearer to the normalized belief. Interestingly, this result in (6.7) is identical to the first derivative $\nabla_{b_0} \eta(\phi_{I_k} b_0)$ along sample path I_k taken with respect to b_0 (as derived in Theorem A1 in Section 6.7.1).

By evaluating the perturbation in the evolution this way, we will need to normalize the belief at stage k before evaluating the cost. The unnormalized belief is just a linear function of the initial unnormalized belief. This property will become useful when chaining forecasted evolutions, which we will demonstrate below in Section 6.3.2.

To determine the perturbation of the the forecasted evolution of the system, we need the formulation of not only the effect of the perturbation of the initial belief b_0 on $b_k^{(i)}$ at each stage k, but also the probability of the sample path $I_k^{(i)}$ occurring for each $I_k^{(i)} \in \tilde{\mathcal{I}}_k$. We show in Proposition 6.2 in Section 6.7.1 that if each sample path produces a unique belief (so that no two sample paths produce the same belief), the probability of a belief b_k occurring starting from b_0 is equal to the probability of the sample path for which the belief was generated, or

$$p(b_k^{(i)}) = p(I_k^{(i)} \mid b_0) = \mathbf{1}^T \phi_{I_k^{(i)}} b_0. \tag{6.8}$$

This probability of a sample path is defined over all possible paths at stage k, i.e $\mathcal{I}_k$. Using ensemble forecasting we approximate the forecasted evolution with only a subset $\tilde{\mathcal{I}}_k$ of the possible sample paths. The result is an approximation of the forecasted evolution whereby

the total probability of the subset of sample paths is less than one. To account for this, we normalize the resulting weight by

$$\mu_k = \sum_{I_k^{(i)} \in \tilde{\mathcal{I}}_k} p(I_k^{(i)}|b_0).$$

The approximated sampling weight, which approximates the probability of belief b_k, being sampled is

$$w_k^{(i)} = \frac{\mathbf{1}^T \phi_{I_k^{(i)}} b_0}{\mu_k}.$$

Weighting each sample path this way only holds when the sampling method used to generate sample paths is an unbiased sampling method, which is an assumption of Proposition 6.2 in Section 6.7.1. If a different sampling function is used, such as an arbitrary importance sampling function, then the sensitivity of the sampling function generating the sample path must also be considered, as any other sampling function biases the probability of a sample path occurring.

If the sampling of $\tilde{I}_k$ is unbiased, by retaining the same set of sample paths for a perturbed belief, we are biasing the simulation for the perturbed belief. We address this bias using methods derived from importance sampling techniques. The existing set of sample paths will not occur with the same probability for a perturbed initial belief. The perturbation of the probability of the sample path is similar to the weighting of typical particle filtering methods (refer to [57] for a survey of particle filtering techniques).

Importance sampling techniques reduce bias by dividing the true probability distribution by probability of the importance sampling function generating the sampled value. For instance, consider a true distribution $p(\cdot)$ and some function $g(\cdot)$,

$$E[g(b)] = \sum_{b \in \mathcal{P}_b} g(b)p(b).$$

However, we will approximate the expectation using a set of samples generated from $q(b)$,

known as the importance sampling function. We denote the set of samples as $\tilde{\mathcal{P}}_b \subset \mathcal{P}_b$. The approximated expectation evaluated over $\tilde{\mathcal{P}}_b$ becomes

$$E[g(b)] \approx \sum_{b \in \tilde{\mathcal{P}}_b} g(b) \frac{w(b)}{\sum_{b \in \tilde{\mathcal{P}}_b} w(b)},$$

where

$$w(b) = \frac{p(b)}{q(b)}.$$

This result is established in [98, p. 54]. We can simplify notation by defining $\mu = \sum_{b \in \tilde{\mathcal{P}}_b} w(b)$ and expressing expressing $w(b)$ directly in terms of $p(b)/q(b)$:

$$E[g(b)] \approx \sum_{b \in \tilde{\mathcal{P}}_b} \frac{1}{\mu} g(b) \frac{p(b)}{q(b)}. \tag{6.9}$$

The expectation of a function of a random variable with a probability function $p(b)$ can be represented as the expectation of another random variable with the probability function $q(b)$ by weighting $g(b)$ by the ratio of $p(b)$ and $q(b)$. The weighting of each sample by the ratio $p(b)/q(b)$ reduces the bias of the importance sampling on the expected value:

$$E[g(b)] \approx \sum_{b \in \tilde{\mathcal{P}}_b} \frac{1}{\mu} g(b) \frac{p(b)}{q(b)} q(b) = \sum_{b \in \tilde{\mathcal{P}}_b} \frac{1}{\mu} g(b) p(b).$$

The expectation above is taken over $q(b)$ because $q(b)$ is the probability function used to sample $\tilde{\mathcal{P}}_b$.

Thus, to alleviate the bias introduced by reusing the same set of sample paths, we adjust the weight of each sample. Observing that $p(I_k^{(i)}|b_0 + \Delta b_0)$ plays the role of $p(b)$ in (6.9) and the importance sampling function, $q(b) = p(I_k^{(i)}|b_0)$, the adjusted weight of each path

becomes

$$\begin{aligned}\frac{1}{\mu}\frac{p(b)}{q(b)} &= \frac{1}{\mu}\left(\frac{p(I_k^{(i)}|b_0+\Delta b_0)}{p(I_k^{(i)}|b_0)}\right) \\ &= \frac{1}{\mu}\left(\frac{\mathbf{1}^T\phi_{I_k^{(i)}}(b_0+\Delta b_0)}{\mathbf{1}^T\phi_{I_k^{(i)}}b_0}\right) \\ &= \frac{1}{\mu}\left(1+\frac{\mathbf{1}^T\phi_{I_k^{(i)}}\Delta b_0}{\mathbf{1}^T\phi_{I_k^{(i)}}b_0}\right).\end{aligned} \tag{6.10}$$

To ensure we are taking the expectation, we normalize the weights sum to one via $\frac{1}{\mu}$, where

$$\mu = \sum_{I_k^{(i)}\in\tilde{\mathcal{I}}_k} \frac{p(I_k^{(i)}|b_0+\Delta b_0)}{p(I_k^{(i)}|b_0)}.$$

In essence, we scale the weight of each path to account for the changing likelihood of a sample path occurring.

The forecasted evolution under a perturbation in the initial belief can be approximated using this biased set of sample paths. What we see from the perturbation analysis is that the unnormalized belief (6.7) and belief probability (6.8) are linear functions of the initial belief. We will demonstrate below in Section 6.3.2 that we can reformulate the problem to avoid weighting each belief at each stage.

Each perturbed belief along a sample path is a ratio of affine function of the initial belief b_0, where the numerator is a vector and the denominator is a scalar. The denominator normalizes the vector to project the product of $\phi_{I_k}b_0$ onto the belief simplex. This normalization is the only nonlinear factor and is the only complication when predicting the perturbed evolution along a sample path. However, because the normalization is an inverse value of a scalar field, as we show in Theorem A3 from Section 6.7.3 , higher order approximations do not require an exponential increase in the number of parameters describing the Taylor's series approximation.

The composite belief transition function under information vector $I_k^{(i)}$ can by be obtained

iteratively through the product of $\phi_{I_{s-1}^{(i)}} \phi_{y_s,u_{s-1}}$ for $s = 1 \dots k$. If X is the number of states in $\mathcal{X}$, the product at each stage is effectively a matrix multiplication, which has $O(X^{2.38})$ time complexity [99] for the exact solution and an approximate solution can be obtained in $O(X^2)$ [100]–both using $O(X^2)$ space. If the transition matrix is sufficiently sparse so that the number of nonzero entries is $O(X^{\frac{1}{2}})$. If the composite matrix remains sparse (which we can enforce via approximation methods), the time complexity of matrix multiplication can be reduced to sub-quadratic time complexity. This process must be performed for each of the P sample paths representing the forecasted evolution, which requires storage of $\phi_{I_k^{(i)}}$, for each $I_k^{(i)} \in \tilde{\mathcal{I}}_k$.

We define $O(S)$ as the time complexity of sampling an observation, K as the time horizon, P as the number of sample paths in $\tilde{\mathcal{I}}_k$, and $O(M)$ as the time complexity required for the policy to select an action. If we consider approximate matrix multiplication, both standard ensemble forecasting and the perturbation analysis we present have a time complexity of $O(KP(X^2 + M + S))$. However, once simulated our approach requires only $O(PX^2)$ to evaluate the effect of a perturbation. Unfortunately, the additional space complexity increases from a constant, $O(1)$, to $O(PX^2)$. Both this time complexity and space requirement may indicate that further compression/approximation of the sample paths should be performed. This remains an area of future research.

6.3.2 Running cost sensitivity

Besides generating the effect of a perturbation on the set of terminal beliefs, we need to represent the effect a perturbation has on the running cost of a forecasted evolution. For the duration of the analysis, we assume an expected-state cost function, so the cost is a linear function over the belief space. We will denote the expected-state cost function $c(b_k^{(i)}, u_k^{(i)})$ as $c_{u_k^{(i)}}^T b_k^{(i)}$, where $c_{u_k^{(i)}}^T$ is a row vector. We derive results for general nonlinear cost functions in Section 6.7.3.

The total cost for a time horizon K under policy π for an expected state cost, from 3.1 in Section 3.1 with $\gamma = 1$[6], is given by

$$V(b_0) = E\left[\sum_{k=0}^{K} c(b_k, \pi(b_k)) \mid b_0\right] \tag{6.11}$$

$$= \sum_{k=0}^{K} \sum_{I_k^{(i)} \in \mathcal{I}_k} c_{u_k^{(i)}}^T b_k^{(i)} p(I_k^{(i)}|b_0) \tag{6.12}$$

$$= \sum_{k=0}^{K} \sum_{I_k^{(i)} \in \mathcal{I}_k} c_{u_k^{(i)}}^T \frac{\phi_{I_k^{(i)}} b_0}{\mathbf{1}^T \phi_{I_k^{(i)}} b_0} \mathbf{1}^T \phi_{I_k^{(i)}} b_0 \tag{6.13}$$

$$= \left(\sum_{k=0}^{K} \sum_{I_k^{(i)} \in \mathcal{I}_k} c_{u_k^{(i)}}^T \phi_{I_k^{(i)}}\right) b_0. \tag{6.14}$$

The first equation is the definition of the total cost from b_0. In (6.12), we give the explicit formulation of the expected cost. The probability $p(I_k^{(i)}|b_0) = \mathbf{1}^T \phi_{I_k^{(i)}} b_0$ from Proposition 6.2 and $b_k^{(i)} = \frac{\phi_{I_k^{(i)}} b_0}{\mathbf{1}^T \phi_{I_k^{(i)}} b_0}$ from Proposition 6.1 are used to obtain (6.13). Finally, (6.14) is achieved by simplifying (6.13).

The important observation is that b_0 can be pulled outside both summations. Thus, each of the vectors, $c_{u_k^{(i)}}^T \phi_{I_k^{(i)}}$, can be summed together into one vector α^T, so that the total running cost is simply:

$$V(b_0) = \alpha^T b_0.$$

The running cost for a perturbed belief $b_0 + \Delta b_0$ is simply: $V_\pi(b_0 + \Delta b_0) = \alpha^T (b_0 + \Delta b_0)$.

Unfortunately, this result only holds if all possible sample paths are simulated. When only a subset of sample paths are simulated, the effect of the approximation has to be considered. To avoid re-simulation, we use the existing sample paths, which act as the importance sampling function for the perturbed initial belief. We must, therefore, attenuate bias introduced when using the existing sample paths for a perturbed initial belief. For

[6]By setting $\gamma = 1$, we derive an expression for indefinite, finite-time horizon problems. Simply substituting γ back into the following equations produces a formulation for discounted infinite horizon problems.

simplicity we will denote $b_0 + \Delta b_0 = b_0'$ and $\mu_k + \Delta\mu_k = \mu_k'$. The bias adjusted running cost, derived from (6.11), becomes:

$$V(b_0') = E\left[\sum_{k=0}^{K} c(b_k, \pi(b_k)) \mid b_0'\right] \tag{6.15}$$

$$\sum_{k=0}^{K} \frac{1}{\mu_k'} \sum_{I_k^{(i)} \in \tilde{\mathcal{I}}_k} c_{u_k^{(i)}}^T b_k^{(i)\prime} \frac{p(b_k'|b_0')}{q(b_k'|b_0')} \tag{6.16}$$

$$\sum_{k=0}^{K} \frac{1}{\mu_k'} \sum_{I_k^{(i)} \in \tilde{\mathcal{I}}_k} c_{u_k^{(i)}}^T b_k^{(i)\prime} \frac{p(I_k^{(i)}|b_0')}{p(I_k^{(i)}|b_0)} \tag{6.17}$$

$$= \sum_{k=0}^{K} \left(\sum_{I_k^{(j)} \in \tilde{\mathcal{I}}_k} \frac{\mathbf{1}^T \phi_{I_k^{(j)}} b_0'}{\mathbf{1}^T \phi_{I_k^{(j)}} b_0} \right)^{-1} \sum_{I_k^{(i)} \in \mathcal{I}_k} c_{u_k^{(i)}}^T \frac{\phi_{I_k^{(i)}} b_0'}{\mathbf{1}^T \phi_{I_k^{(i)}} b_0'} \frac{\mathbf{1}^T \phi_{I_k^{(i)}} b_0'}{\mathbf{1}^T \phi_{I_k^{(i)}} b_0} \tag{6.18}$$

$$= \sum_{k=0}^{K} \Bigg(\underbrace{\left[\sum_{I_k^{(j)} \in \tilde{\mathcal{I}}_k} \frac{\mathbf{1}^T \phi_{I_k^{(j)}}}{\mathbf{1}^T \phi_{I_k^{(j)}} b_0} \right]}_{\xi_k^T} b_0' \Bigg)^{-1} \underbrace{\left[\sum_{I_k^{(i)} \in \mathcal{I}_k} \frac{c_{u_k^{(i)}}^T \phi_{I_k^{(i)}}}{\mathbf{1}^T \phi_{I_k^{(i)}} b_0} \right]}_{\alpha_k^T} b_0' \tag{6.19}$$

$$= \sum_{k=0}^{K} \frac{1}{\xi_k^T b_0'} \alpha_k^T b_0'. \tag{6.20}$$

The second equation, (6.16), follows from the formulation of the effect of biased sampling on the expected cost from (6.9). Next, (6.17), follows from 6.9 where $p(b)$ is represented as $p(b_k'|b_0')$ and $q(b)$ is represented by $q(b_k'|b_0')$. We use the existing set of sample paths for the importance sampling function so $q(b_k'|b_k') = p(b_k|b_0)$. From Proposition 6.2 in Section 6.7.1, we observe that $p(b_k'|b_0') = p(I_k^{(i)}|b_0')$ and $p(b_k|b_0) = p(I_k^{(i)}|b_0)$. The term $1/\mu_k'$ is a normalizing factor defined as the sum of the weights of each belief at stage k, where

$$\mu_k' = \sum_{I_k^{(i)} \in \tilde{\mathcal{I}}_k} \frac{p(I_k^{(i)}|b_0')}{p(I_k^{(i)}|b_0)}.$$

Because we generated the sample path set from b_0 using the true probability function, the

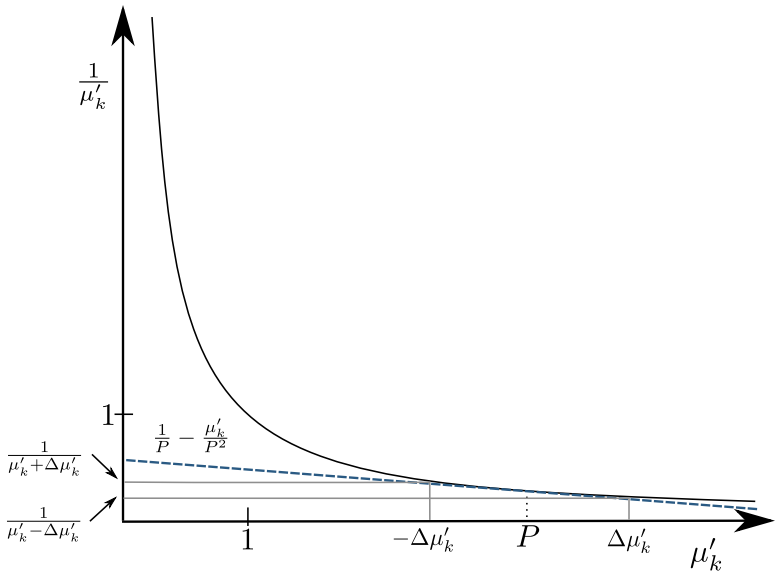


Figure 6.5: Linear approximation of the normalizing function: $\frac{1}{\mu'_k}$

biased probability the sample path occurring is $p(I_k^{(i)}|b_0) = \mathbf{1}^T \phi_{I_k^{(i)}} b_0$, which was derived from Proposition 6.2 in Section 6.7.1 and substituting $b_k^{(i)\prime} = \frac{\phi_{I_k^{(i)}} b_0'}{\mathbf{1}^T \phi_{I_k^{(i)}} b_0'}$. The perturbed initial belief Δb_0 is pulled out of the summations to arrive at (6.19).

Unfortunately, the result (6.20) is a ratio of linear functions. The inverse dependence on b_0' infringes upon our ability to reduce the expression into a compact form that is expressed as a single term. Instead, (6.20) is expressed as a summation over all stages. We note, however, a linear approximation of $1/\mu'_k$ is often reasonable for perturbations within a neighborhood of b_0, which suggests we can approximate the ratio of linear functions as a linear function. We can sum the components of a linear representation to obtain a compact form.

For perturbations within a neighborhood of b_0, $\mu'_k \approx P$, which is the number of sample paths, i.e.

$$\sum_{I_k^{(i)} \in \tilde{\mathcal{I}}_k} \frac{p(I_k^{(i)}|b_0 + \Delta b_0)}{p(I_k^{(i)}|b_0)} \approx \sum_{I_k^{(i)} \in \tilde{\mathcal{I}}_k} \frac{p(I_k^{(i)}|b_0)}{p(I_k^{(i)}|b_0)} = \sum_{I_k^{(i)} \in \tilde{\mathcal{I}}_k} 1 = P,$$

which is strictly greater than one. This notion is illustrated in Figure 6.5. As depicted, where $\mu'_k = \xi_k^T b'_0$, the approximation due to a perturbation is better bounded as μ'_k increases. The

illustration in Figure 6.5 is for a scalar value–not a scalar field–but the concept extends to higher dimensions. As can be seen a perturbation $\Delta\mu'_k$ results in a small perturbation in $\frac{1}{\mu'_k+\Delta\mu'_k}$ in the neighborhood around $\mu'_k = P$.

A linear approximation is a reasonable representation as long as $\xi_k^T b'_0$ is not significantly less than one. Fortunately, this is the case as

$$\begin{aligned}\xi_k^T(b_0+\Delta b_0) &= \sum_{I_k^{(j)}\in\tilde{\mathcal{I}}_k} \frac{\mathbf{1}^T\phi_{I_k^{(j)}}(b_0+\Delta b_0)}{\mathbf{1}^T\phi_{I_k^{(j)}}b_0} \\ &= \sum_{I_k^{(j)}\in\tilde{\mathcal{I}}_k}\left[1+\frac{\mathbf{1}^T\phi_{I_k^{(j)}}\Delta b_0}{\mathbf{1}^T\phi_{I_k^{(j)}}b_0}\right].\end{aligned}$$

Dividing by $\mathbf{1}^T\phi_{I_k^{(j)}}b_0$ shifts $\mathbf{1}^T\phi_{I_k^{(j)}}(b_0+\Delta b_0)$ closer to one (or greater than one). Moreover, the sum is taken over all the sample paths which further increases the total weight.

To determine the first order Taylor series approximation, we first need to determine the first order derivative of (6.20) at each stage k:

$$\begin{aligned}\nabla_{b_0}\left(\frac{1}{\xi_k^T b'_0}\alpha_k^T b'_0\right) &= \frac{1}{\xi_k^T b'_0}\nabla_{b_0}\left(\alpha_k^T b'_0\right)+\nabla_{b_0}\left(\frac{1}{\xi_k^T b'_0}\right)\alpha_k^T b'_0 \\ &= \frac{1}{\xi_k^T b'_0}\alpha_k^T-\frac{\xi_k^T}{(\xi_k^T b'_0)^2}\alpha_k^T b'_0 \qquad (6.21)\\ &= \frac{\alpha_k^T}{\xi_k^T b'_0}-\frac{\alpha_k^T b_0\xi_k^T}{(\xi_k^T b'_0)^2}. \qquad (6.22)\end{aligned}$$

Now, the first order Taylor series expansion, which becomes our approximated running cost, is given by

$$V(b_0 + \Delta b_0) = \sum_{k=0}^{K} \frac{1}{\xi_k^T b_0'} \alpha_k^T b_0'. \tag{6.23}$$

$$\approx \sum_{k=0}^{K} \left[\frac{\alpha_k^T b_0}{\xi_k^T b_0} + \frac{1}{1!} \nabla_{b_0} \left(\frac{1}{\xi_k^T b_0'} \alpha_k^T b_0' \right) \Delta b_0 \right] \tag{6.24}$$

$$= \underbrace{\sum_{k=0}^{K} \frac{\alpha_k^T b_0}{\xi_k^T b_0}}_{V^{sim}} + \underbrace{\sum_{k=0}^{K} \left(\frac{\alpha_k^T}{\xi_k^T b_0} + \frac{\alpha_k^T b_0 \xi_k^T}{(\xi_k^T b_0)^2} \right)}_{\alpha^T} \Delta b_0 \tag{6.25}$$

$$= V^{sim} + \alpha^T \Delta b_0, \tag{6.26}$$

where V^{sim} denotes the simulated result for the reference belief b_0. The first equation is a restatement of (6.20) for $b_0' = b_0 + \Delta b_0$. The first order Taylor series expansion is the simulated value $\frac{\alpha_k^T b_0}{\xi_k^T b_0}$ plus the first order derivative derived in (6.22) weighted by Δb_0. Substituting this first order approximation into (6.23) to obtain (6.25). The final result in (6.26) is a linear function of Δb_0. Note that α^T is the running cost Jacobian that is the weighted sum of each Jacobian α_k^T at stage k.

The result with (6.26) is that the running cost over all stages and sample paths is approximated by a single linear equation. This compact representation comes at the trade-off of exactness. To achieve, we assume that the same set of actions and observations occur for a perturbed belief. While locally this is likely the case, the question remains to the size of the neighborhood around the sampled belief b_0 for which this is the case. This issue is somewhat mitigated by the fact that stochastic systems are generally insensitive and perturbations are attenuated over time.

Using (6.26), computing the running cost for a forecasted evolution under a perturbed belief is reduced from $O(KP(X^2+M+S))$ to $O(PX^2)$, where $O(M)$ is the time complexity of determining the next control action for each belief, $O(S)$ is the time complexity of generating the sample observation, P is the number of sample paths in $\tilde{\mathcal{I}}_K$, and X is the number of states in the system. If a first-order approximation is used instead, the time complexity reduces to

$O(X)$, which results from computing a dot product, which has $O(X)$ time complexity, and and adding a constant value. We note, however, that generating (6.26) requires simulating the system which has time complexity $O(KP(X^2+M+S))$, as described earlier. Once this term is calculated, re-evaluation of the running cost has the reduced $O(X)$ time complexity

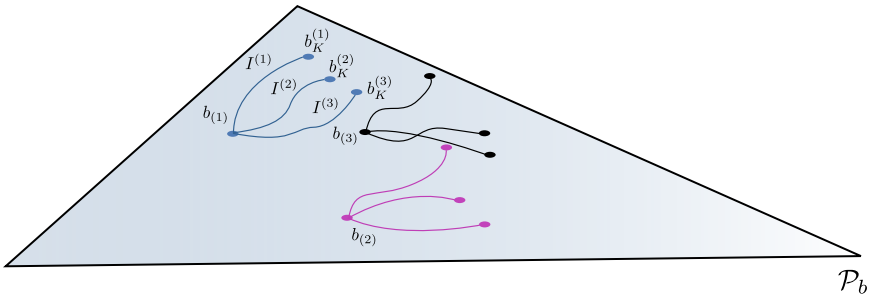
.

We can see that if there are U actions and the policy function takes minimally $O(X^2U)$ time complexity to determine the next best action, e.g., expected distance from goal state, the savings is substantial when the time horizon is long. However, as with the transition sensitivity function. The savings in time complexity is offset by space complexity. Each sample path requires the running cost Jacobian to be retained requiring $O(X)$ space. Thus, the total space complexity for each forecasted evolution is $O(X)$.

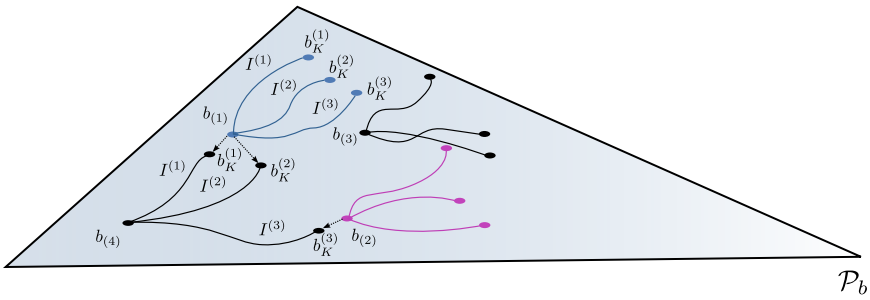
6.4 Chaining Sensitivity Functions

Once a sufficient set of forecasted evolutions has been simulated, the chaining process may proceed. The intent of the chaining process is to extend forecasting of a POMDP's behavior beyond a single forecasted evolution. This way, forecasted evolutions for long time-horizons that start from a variety of initial beliefs can be approximated by leveraging the existing set of forecasted evolutions. In Figure 6.6 we illustrate this concept. Up to this point, we have only discussed a single forecasted evolution as represented by $\tilde{\mathcal{I}}_K$. Now, however, we will consider a set of forecasted evolutions. For clarity we drop the time index and add notation to index the forecasted evolution, so that set of forecasted evolutions is denoted as $\{\tilde{\mathcal{I}}_{(s)}\}_s$. The initial belief for the forecasted evolution $\tilde{\mathcal{I}}_{(s)}$ is denoted as $b_{(s)}$ with no time index.

Suppose that we have an existing set of forecasted evolutions $\{\tilde{\mathcal{I}}_{(s)}\}_s$ (Figure 6.6a). Consider, for example, the case when we want to forecast the system starting from the new belief $b_{(4)}$. First, we simulate the system from $b_{(4)}$ for a number of stages to create a new forecasted simulation $\tilde{\mathcal{I}}_{(4)}$. The result of this process is depicted in Figure 6.6b. Then, the



(a) Existing set of forecasted evolutions: $\tilde{\mathcal{I}}_{(1)}$,$\tilde{\mathcal{I}}_{(2)}$, and$\tilde{\mathcal{I}}_{(3)}$.



(b) Adding a new forecasted evolution and selecting the forecasted evolutions to chain.

Figure 6.6: An illustration of the chaining process

terminal beliefs of the new forecasted evolution are compared to the starting beliefs of the existing forecasted simulations. The nearest starting belief $b_{(s)}$ to each terminal belief $b_K^{(i)}$ is selected to join with the new forecasted evolution. We can see in Figure 6.6b that $b_K^{(1)}$ and $b_K^{(2)}$ are closest to $b_{(1)}$ and $b_K^{(3)}$ is closed to $b_{(2)}$. Thus, $\tilde{\mathcal{I}}_{(4)}$ is to be joined with $\tilde{\mathcal{I}}_{(1)}$ and $\tilde{\mathcal{I}}_{(2)}$. To ease description, we will henceforth, describe the new forecasted evolution as a *source forecasted evolution* and each of the forecasted evolutions to be joined as a *target forecasted evolution*. In this example $\tilde{\mathcal{I}}_{(4)}$ is the source forecasted evolution and $\tilde{\mathcal{I}}_{(1)}$ and $\tilde{\mathcal{I}}_{(2)}$ are target forecasted evolutions.

Chaining forecasted evolutions together using our approach relies on a set of forecasted evolutions having already been simulated, whose initial beliefs cover a representative portion of the belief space. If the set of forecasted evolutions is too sparse, then the quality of the chaining approximation will suffer. We do not present any analysis of the error of sparsity on the chaining process beyond the residual error of the perturbation approximation (refer to Theorem A6 from Section 6.7.3).

6.4.1 Chaining sample paths

Once the set of forecasted evolutions to be chained are selected, the next step is to generate the chained representations of the sample paths that comprise the chained forecasted evolution. Chained sample paths are easily computed; joining paths is equivalent to joining the information vectors of two sample paths. For instance, suppose we are generating a chained forecasted evolution from the source $\tilde{\mathcal{I}}_{(s)}$. We denote the resulting chained forecasted evolution as $\tilde{\mathcal{I}}_{(w)}$. If we combine $I^{(i)} \in \tilde{\mathcal{I}}_{(s)}$ with target $I^{(j)} \in \tilde{\mathcal{I}}_{(s)}$ to generate $I^{(l)} \in \tilde{\mathcal{I}}_{(w)}$, then the chained information vector is

$$I^{(l)} = I^{(i)} \cup I^{(j)} \equiv \{u_0^{(i)}, y_1^{(i)}, \ldots u_{K-1}^{(i)}, y_K^{(i)}, u_0^{(j)}, y_1^{(j)}, \ldots u_{K-1}^{(j)}, y_K^{(j)}\},$$

where both $I^{(i)}$ and $I^{(j)}$ execute for K stages[7]. The chained sample path transition matrix under the joined information vector is just the product of the transition matrices: $\phi_{I^{(l)}} = \phi_{I^{(j)}}\phi_{I^{(i)}}$. The resulting terminal belief along the joined path becomes

$$b_K^{(l)} = \frac{\phi_{I^{(j)}}\phi_{I^{(i)}}b_{(s)}}{\mathbf{1}^T\phi_{I^{(j)}}\phi_{I^{(i)}}b_{(s)}},$$

where the combined path represents the system evolution for $K + K$ stages.

Each source sample path is chained with all sample paths of the target forecasted evolution. For example, Figure 6.6 illustrates a case where $I^{(1)} \in \tilde{\mathcal{I}}_{(4)}$ is joined with each path in $\tilde{\mathcal{I}}_{(1)}$, i.e $\{I^{(1)}, I^{(2)}, I^{(3)}\}$, which produces three new sample paths. This results in geometric increase in the number of sample paths for each chaining operation. Assuming that there are P sample paths for each forecasted evolution, then the chained representation will comprise P^2 sample paths. To stem the potential exponential growth of chained sample paths, we use a sampling technique to retain a representative subset. Each path has the unbiased

[7]Each forecasted evolution may execute for a varying number of stages. However, to keep our formulation simple, we assume that they execute for the same number of stages.

probability

$$p(I^{(l)}|b_{(w)}) = \mathbf{1}^T \phi_{I^{(l)}} b_{(w)}.$$

If we sample each path according to the probability of the path occurring, we generate an unbiased subset of chained sample paths. Alternate sampling methods may be employed to achieve even better representation, but we leave that as a future area of research.

6.4.2 Chaining the running cost

Given this approach to chaining sample paths, we now derive the corresponding chained running cost along the set of sample paths. To obtain the running cost along chained forecasted evolutions, we sum the running cost of the source forecasted evolution with the running cost of each target forecasted evolution. The running cost for each existing forecasted evolution will be denoted as

$$V_{\tilde{\mathcal{I}}_{(s)}}(b_{(s)} + \Delta b_{(s)}) = V_{(s)}^{sim} + \alpha_{(s)}^T \Delta b_{(s)},$$

which is obtained from (6.26). To keep notation as simple as possible, we denote $\tilde{I}_{(*)}$ as the target forecasted evolution for each $I^{(i)} \in \tilde{\mathcal{I}}_{(s)}$. As an example, in Figure 6.6, $I^{(1)}$ and $I^{(2)}$ have target $\tilde{\mathcal{I}}_{(*)} = \tilde{\mathcal{I}}_{(1)}$, while $I^{(3)}$ has the target $\tilde{\mathcal{I}}_{(*)} = \tilde{\mathcal{I}}_{(3)}$. The resulting chained running cost is denoted by $V_{\tilde{\mathcal{I}}_{(w)}}$. Expanding from the definition of the running cost for $2K$ stages (assuming each forecasted evolution executes for K stages), we obtain

$$V_{\tilde{\mathcal{I}}_{(w)}}(b_{(s)}) = E[\sum_{k=0}^{2K} c(b_k, u_k)|b_{(s)}] \tag{6.27}$$

$$= \underbrace{E[\sum_{k=0}^{K} c(b_k, u_k)|b_{(s)}]}_{\approx V_{\tilde{I}_{(s)}}(b_{(s)})} + E[\sum_{k=K+1}^{2K} c(b_k, u_k)|b_{(s)}]. \tag{6.28}$$

The first equation, (6.27), is the definition of the running cost for $2K$ stages starting from $b_{(s)}$. The running cost is split into to summations in (6.28) and we observe from (6.11) that $V_{\tilde{I}_{(s)}}(b_{(s)}) \approx E[\sum_{k=0}^{K} c(b_k, u_k)|b_{(s)}]$.

Next, we recognize $E[\sum_{k=K+1}^{2K} c(b_k, u_k)|b_{(s),}]$ as the portion of the running cost being approximated by the target forecasted evolutions. Each target contributes a portion of the sample paths, and, hence, a portion of the running cost. Together the target forecasted evolutions approximate the expected cost starting after stage K. Since we have already determined the running cost for each target, we just need to evaluate the running cost for each target at the terminal belief of the source sample path. However, we must also weight the contribution of each target by the probability of the source sample path. Since we use importance sampling, we must weight each contribution by the true probability of the source sample path by the biased probability:

$$E[\sum_{k=K+1}^{K} c(b_k, u_k)|b_{(s)}] \approx \frac{1}{\mu} \sum_{I^{(i)} \in \tilde{I}_{(s)}} V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)}) \frac{p(b_K^{(i)}|b_0)}{q(b_K^{(i)}|b_0)},$$

which is a consequence of (6.9), where we adapt the notation to $b_K^{(i)}$ conditioned on b_0, so $p(b_K^{(i)}|b_0)$ denotes $p(b)$ and $p(b_K^{(i)}|b_0)$ denotes $q(b)$. From Proposition 6.2 in Section 6.7.1, we observe that $p(b_K^{(i)}|b_0) = p(I^{(i)}|b_{(s)})$. Likewise, $p(b_K^{(i)}|b_0) = p(I^{(i)}|b_{(s)})$ since we are not evaluating the effect of a perturbation to $b_{(s)}$ yet. Substituting these results into (6.28), we obtain

$$V_{\tilde{\mathcal{I}}_{(w)}}(b_{(s)}) \approx V_{\tilde{I}_{(s)}}(b_{(s)}) + \frac{1}{\mu} \sum_{I^{(i)} \in \tilde{I}_{(s)}} V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)}) \frac{p(I^{(i)}|b_{(s)})}{p(I^{(i)}|b_{(s)})} \tag{6.29}$$

$$= V_{\tilde{I}_{(s)}}(b_{(s)}) + \frac{1}{P} \sum_{I^{(i)} \in \tilde{I}_{(s)}} V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)}). \tag{6.30}$$

To normalize the result, we divide by the number of sample paths $\frac{1}{\mu} = \frac{1}{P}$. This is analogous

to standard sample weighting used in particle filtering and other Monte Carlo methods. Finally, we arrive at (6.30).

The above result holds when evaluating the chained solution from $b_{(s)}$. To keep the form in (6.26) that allows the result of a chaining process to be used in future chaining processes, we must consider the effect of a perturbation $\Delta b_{(s)}$ in the chaining process. Starting with 6.29 and substituting $b_{(s)} + \Delta b_{(s)}$ into the equation and noting that $q(I^{(i)}|b_{(s)}) = p(I^{(i)}|b_{(s)})$, the chained running cost becomes

$$V_{\tilde{I}_{(w)}}(b_{(s)} + \Delta b_{(s)}) = V_{\tilde{I}_{(s)}}(b_{(s)} + \Delta b_{(s)}) + \sum_{I^{(i)} \in \tilde{I}_{(s)}} \frac{1}{\mu} V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)} + \Delta b_K^{(i)}) \frac{p(I^{(i)}|b_{(s)} + \Delta b_{(s)})}{p(I^{(i)}|b_{(s)})}, \quad (6.31)$$

where

$$\mu = \sum_{I^{(i)} \in \tilde{I}_{(s)}} \frac{p(I^{(i)}|b_{(s)} + \Delta b_{(s)})}{p(I^{(i)}|b_{(s)})} = \underbrace{\sum_{I^{(i)} \in \tilde{I}_{(s)}} \frac{\mathbf{1}^T \phi_{I^{(i)}}}{\mathbf{1}^T \phi_{I^{(i)}} b_{(s)}}}_{\xi^T} b'_{(s)}. \quad (6.32)$$

Here the true probability for the sample path is $p(I^{(i)}|b_{(s)} + \Delta b_{(s)})$, but we use the existing set of sample paths to approximate the result, which were generated from $p(I^{(i)}|b_{(s)})$.

Yet again, we are presented with a ratio of linear functions. Fortunately, yet again, the ratio is well approximated by a linear function, as the denominator is often much greater than one. The first order Taylor series approximation of the expectation of future sample paths from (6.31) is

$$\sum_{I^{(i)} \in \tilde{I}_{(s)}} \frac{1}{\mu} V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)} + \Delta b_K^{(i)}) \approx \sum_{I^{(i)} \in \tilde{I}_{(s)}} \left[\frac{1}{P} V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)}) + \nabla_{b'_{(s)}} \left(\frac{1}{\xi^T b'_{(s)}} V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)\prime}) \frac{\mathbf{1}^T \phi_{I^{(i)}} b'_{(s)}}{\mathbf{1}^T \phi_{I^{(i)}} b_{(s)}} \right) \Delta b_K^{(i)} \right], \quad (6.33)$$

where we substitute $\mu = \xi^T b'_{(s)}$ from (6.32). The first term in the summation is obtained the

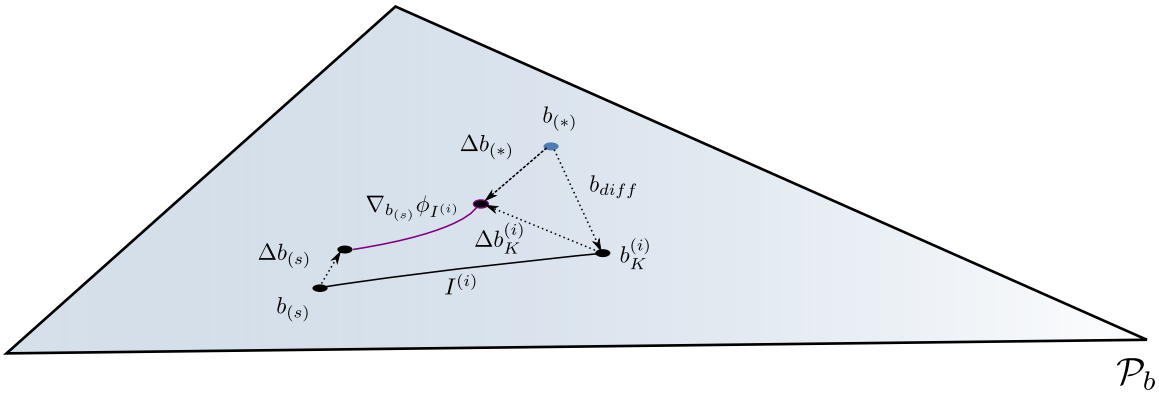


Figure 6.7: Coupling perturbation between forecasted evolutions

unperturbed value: $V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)})\frac{\mathbf{1}^T\phi_{I^{(i)}}b_{(s)}}{\mathbf{1}^T\phi_{I^{(i)}}b_{(s)}} = \frac{1}{P}V_{s_*}(b_K^{(i)})$, where there are P sample paths. In the second term in the summation, the first order derivative is taken with respect to $b'_{(s)}$, which we use to denote the perturbed belief $b'_{(s)} = b_{(s)} + \Delta b_{(s)}$ and the perturbed terminal belief along sample path $I^{(i)}$ as $b_K^{(i)'} = b_K^{(i)} + \Delta b_K^{(i)}$.

We want to analyze the sensitivity of this function under the perturbation $\Delta b_{(s)}$. Furthermore, we need to express the running cost $\tilde{\mathcal{I}}_{(*)}$ as a function of the perturbation $\Delta b_{(*)}$, or $V_{\tilde{\mathcal{I}}_{(*)}}(b_{(*)} + \Delta b_{(*)})$. To achieve this we first to express $\Delta b_{(*)}$ in terms of $b_K^{(i)}$ and $\Delta b_K^{(i)}$:

$$\begin{aligned} b_K^{(i)} + \Delta b_K^{(i)} &= b_{(*)} + \Delta b_{(*)} \\ \Delta b_{(*)} &= \underbrace{b_{(*)} - b_K^{(i)}}_{b_{diff}} - \Delta b_K^{(i)}. \end{aligned} \tag{6.34}$$

However, to achieve a compact representation, we need to express the perturbation in terms of $\Delta b_{(s)}$. In (6.7) from Section 6.3.1, we determined the first order approximation to be:

$$\Delta b_K^{(i)} \approx \nabla_{b'_{(s)}}\phi_{I^{(i)}}\Delta b_{(s)}. \tag{6.35}$$

This coupling is illustrated in Figure 6.7, where the geometric interpretation of $\Delta b_{(*)}$ is made clear.

Replacing (6.35) in (6.34), we produce: $\Delta b_{(*)} = b_{diff} - \nabla_{b_{(s)}}\phi_{I^{(i)}}\Delta b_{(s)}$. All monomial terms

are now expressed relative to $\Delta b_{(s)}$. Inserting this result into the second term of (6.33), we obtain

$$\begin{aligned}
&\sum_{I^{(i)}\in\tilde{I}_{(s)}} \frac{1}{1!}\nabla_{b'_{(s)}}\left(\frac{1}{\xi^T b'_{(s)}}V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)'})\frac{\mathbf{1}^T\phi_{I^{(i)}}b'_{(s)}}{\mathbf{1}^T\phi_{I^{(i)}}b_{(s)}}\right)\\
&\qquad=\sum_{I^{(i)}\in\tilde{I}_{(s)}}\underbrace{V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)})\frac{\mathbf{1}^T\phi_{I^{(i)}}b_{(s)}}{\mathbf{1}^T\phi_{I^{(i)}}b_{(s)}}\nabla_{b'_{(s)}}\left(\frac{1}{\xi^T b'_{(s)}}\right)}_{(A)}+\\
&\qquad\quad\sum_{I^{(i)}\in\tilde{I}_{(s)}}\underbrace{\frac{1}{P}\frac{\mathbf{1}^T\phi_{I^{(i)}}b_{(s)}}{\mathbf{1}^T\phi_{I^{(i)}}b_{(s)}}\nabla_{b'_{(s)}}\left(V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)'})\right)}_{(B)}+\\
&\qquad\quad\sum_{I^{(i)}\in\tilde{I}_{(s)}}\underbrace{\frac{1}{P}V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)})\nabla_{b'_{(s)}}\left(\frac{\mathbf{1}^T\phi_{I^{(i)}}b'_{(s)}}{\mathbf{1}^T\phi_{I^{(i)}}b_{(s)}}\right)}_{(C)} \qquad (6.36)\\
&\qquad=\sum_{I^{(i)}\in\tilde{I}_{(s)}}\underbrace{-(V_{(*)}^{sim}+\alpha_{(*)}^T\Delta b_{diff})\frac{\xi^T}{(\xi^T b_{(s)})^2}}_{(A)}\Delta b_{(s)}+\\
&\qquad\quad\sum_{I^{(i)}\in\tilde{I}_{(s)}}\underbrace{\frac{1}{P}\frac{\alpha_{(*)}^T\phi_{I_k}}{\mathbf{1}^T\phi_{I_k}b_0}\left(\mathbf{I}-\frac{b_0\mathbf{1}^T\phi_{I_k}}{\mathbf{1}^T\phi_{I_k}b_0}\right)}_{(B)}\Delta b_{(s)}+\\
&\qquad\quad\sum_{I^{(i)}\in\tilde{I}_{(s)}}\underbrace{\frac{1}{P}(V_{(*)}^{sim}+\alpha_{(*)}^T\Delta b_{diff})\frac{\mathbf{1}^T\phi_{I^{(i)}}}{\mathbf{1}^T\phi_{I^{(i)}}b_{(s)}}}_{(C)}\Delta b_{(s)} \qquad (6.37)\\
&\qquad=\underbrace{\sum_{I^{(i)}\in\tilde{I}_{(s)}}}_{\alpha_{(w)}^T}[(A)+(B)+(C)]\Delta b_{(s)}. \qquad (6.38)
\end{aligned}$$

The first equation, (6.36), is a representation of each of the components that from application of the product rule. The result of the first order derivative of each component are captured in (6.37). There are several key observations that help us simplify the expression. First, in both (A) and (C), we have the term $V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)})$. However, our representation for each running

cost is relative to a perturbation: $V_{\tilde{\mathcal{I}}_{(*)}}(b_{(*)} + \Delta b_{(*)}) = V_{(*)}^{sim} + \alpha_{(*)}^T \Delta b_{(*)}$ as defined in (6.26). To evaluate this term, we substitute $b_K^{(i)} = b_{(*)} + b_{diff}$ to obtain $V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)}) = V_{(*)}^{sim} + \alpha_{(*)}^T \Delta b_{diff}$.

In component (B) we have

$$\nabla_{b'_{(s)}} \left(V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)'}) \right) = \nabla_{b'_{(s)}} \left(V_{\tilde{\mathcal{I}}_{(*)}} \left(\frac{\phi_{I^{(i)}} b'_{(s)}}{\mathbf{1}^T \phi_{I^{(i)}} b'_{(s)}} \right) \right) \tag{6.39}$$

$$= \nabla_{b'_{(s)}} V_{\tilde{\mathcal{I}}_{(*)}} \nabla_{b'_{(s)}} \left(\frac{\phi_{I^{(i)}} b'_{(s)}}{\mathbf{1}^T \phi_{I^{(i)}} b'_{(s)}} \right) \tag{6.40}$$

$$= \nabla_{b'_{(s)}} V_{\tilde{\mathcal{I}}_{(*)}} \nabla_{b'_{(s)}} \eta(\phi_{I^{(i)}} b'_{(s)}) \tag{6.41}$$

$$= \alpha_{(*)}^T \underbrace{\frac{\phi_{I_k}}{\mathbf{1}^T \phi_{I_k} b_0} \left(I - \frac{b_0 \mathbf{1}^T \phi_{I_k}}{\mathbf{1}^T \phi_{I_k} b_0} \right)}_{\nabla_{b'_{(s)}} \eta(\phi_{I^{(i)}} b'_{(s)})}. \tag{6.42}$$

The first equation results from just substituting $b_K^{(i)'} = \frac{\phi_{I^{(i)}} b'_{(s)}}{\mathbf{1}^T \phi_{I^{(i)}} b'_{(s)}}$. Next, in (6.40) we apply the chain rule. Using the result from (6.7) to expand obtain (6.41). Finally, we obtain (6.42) by observing that the term $\nabla_{b'_{(s)}} V_{\tilde{\mathcal{I}}_{(*)}}$ is the Jacobian of the running cost, which is precisely $\alpha_{(*)}^T$ from (6.26).

Because we can pull out $\Delta b_{(s)}$ from each component (i.e. (A), (B), and (C)), we arrive at the simplified expression in (6.38) as a representation of the second term in (6.33). We obtain a simplified expression of the first term in (6.33) by substituting $b_K^{(i)} = b_{(*)} + b_{diff}$:

$$\sum_{I^{(i)} \in \tilde{I}_{(s)}} \frac{1}{P} V_{\tilde{\mathcal{I}}_{(*)}}(b_K^{(i)}) = \sum_{I^{(i)} \in \tilde{I}_{s_1}} \frac{1}{P} V_{\tilde{\mathcal{I}}_{(*)}}(b_{(*)} + b_{diff})$$

$$= \underbrace{\sum_{I^{(i)} \in \tilde{I}_{s_1}} \frac{1}{P} \left[V_{(*)}^{sim} + \alpha_{(*)}^T \nabla_{b_{(s)}} \phi_{I^{(i)}} b_{diff} \right]}_{V_{(w)}^{sim}}.$$

Now, if we insert the above results into (6.33), we arrive at the chained representation of the

running cost:

$$V_{\tilde{I}_{(w)}}(b_{(s)} + \Delta b_{(s)}) \approx V_{(w)}^{sim} + \alpha_{(w)}^{T} \Delta b_{(s)}. \tag{6.43}$$

The chained running cost is a linear approximation, with no increase in the degree or complexity of the representation over the individual running costs.

While generating a running cost sensitivity function eliminates the need to simulate running cost sensitivity functions along a forecasted evolution to determine the cost along a policy due to perturbations, it does so with some trade-offs. A perturbation may cause the closest belief between the terminal belief to switch to the initial belief of another forecasted evolution. However, the chained function is not flexible in this regard and all associations are static. As discussed above, the impact of perturbations in stochastic systems generally decrease as POMDPs under some general conditions are naturally dissipative systems. This implies the effect of a perturbation early on should decrease further along the policy/sample path evolution.

Compositing the running cost sensitivities of a policy with the running cost sensitivity has $O(PX^2)$ time complexity. Each of the constant, vector, and matrix terms are summed for the P sample paths. Approximate matrix-matrix multiplication has $O(X^2)$ time complexity, where X is the number of of states and, hence, the dimension of the matrices. The computational savings of the perturbation analysis presented is significant. However, the potential application to temporally and spatially abstracted POMDP optimization methods may prove to be of even greater utility.

6.4.3 Selecting target forecasted evolutions

The methodology of selecting which of the forecasted evolutions are chained together is application dependent. If the same policy is used throughout, with only the initial belief varying, then selecting forecasted evolutions that minimize the distance between the terminal beliefs

of the forecasted evolution being expanded to the initial belief the remaining forecasted evolutions will minimize the effect of perturbations along the chained forecasted evolution. In Figure 6.6b, this corresponds to selecting $\tilde{\mathcal{I}}_{(2)}$ and $\tilde{\mathcal{I}}_{(3)}$ when expanding $\tilde{\mathcal{I}}_{(4)}$ as each of the terminal beliefs of forecasted evolution evolution $\tilde{\mathcal{I}}_{(4)}$, i.e. $\{b_K^{(i)}\}_i$, fall closest to $b_{(2)}$ and $b_{(3)}$. However, if different policies are simulated and the goal is to choose the best running, then the method for selection would be to weight both the distance and running cost along the existing forecasted evolutions. Such a scenario corresponds to temporally abstracted version of the Bellman backup as each forecasted evolution is representative of multiple stages.

Our instantiation generates a chained representation that expands a source forecasted evolution to a set of target forecasted evolutions. Thus, the chained forecasted evolution begins from the source's belief. It is conceivable that the chaining process might be reversed for some applications (e.g., rare event detection), in which case the chaining would join the starting belief of a target to the terminal belief of a source. However, we will not address methodologies of constructing chained paths for such applications.

6.4.4 Application to POMDP optimization

The need for so much infrastructure may seem unwarranted–especially when considering expected-state cost functions. However, most current POMDP value iteration methods retain a representation of the value function around a belief sample via what is often referred to as α-vectors. For expected-state cost functions, the value function is piecewise linear and convex [25]. Most methods represent the value function as the minimum over a set of α-vectors, which correspond to the cost-to-go for the set of sampled beliefs. The α-vectors are defined over the entire belief space and correspond to sensitivity around a belief sample, which is an exact representation not an approximation. However, such derivation only holds for single stage optimization techniques wherein all observations are considered. This is a result of the expectation over all observations canceling out the probability of the observation,

which reduces the formulation to a linear function of the cost:

$$\begin{aligned} V^*(b) &= \min_{u\in\mathcal{U},\alpha\in A}\left[c_u^T b + \sum_{y\in\mathcal{Y}}\left(\alpha^T\frac{\phi_{u,y}}{\mathbf{1^T}\phi_{u,y}b}b\right)p(y|b,u)\right] \\ &= \min_{u\in\mathcal{U},\alpha\in A}\left[c_u^T b + \sum_{y\in\mathcal{Y}}\left(\alpha^T\frac{\phi_{u,y}}{\mathbf{1^T}\phi_{u,y}b}b\right)\mathbf{1^T}\phi_{u,y}b\right] \\ &= \min_{u\in\mathcal{U},\alpha\in A}\left[\left(c_u^T + \sum_{y\in Y}\alpha^T\phi_{u,y}\right)b\right]. \end{aligned}$$

The cost function C_u under action u is a linear function of belief. The first equation is the definition of the Bellman backup. The second equation follows from substituting $p(y|b,u) = \mathbf{1^T}\phi_{u,y}b$. The third equation pulls b out side of the constant terms.

For systems with a large number of observations, this requirement to evaluate all observations becomes excessively burdensome–even for one stage. Such a representation is not practical for temporal abstraction as it would require the consideration of an exponential number of observations in the number of stages temporally abstracted. However, if only a subset of sample paths are considered, there is a normalization term μ_k at each stage k as derived above in (6.17), that transforms the simple linear function above to into a ratio of linear function.

Because temporal abstraction introduces the need to sample a subset of the possible paths, the method we propose, which uses a linear form to derive an approximation of the cost-to-go, can be used as the replacement for the α-vector representation that obtained when performing the exact single stage backup. The trade-off for extending from point-based to perturbation analysis comes at the cost of approximation error and increased time complexity. The majority of the time complexity in this analysis is a result of the matrix-matrix multiplication to generate $\phi_{I_K^{(i)}}$. As alluded to earlier, approximate matrix multiplication is one potential tool that to increase the efficiency of this approach. Another method could be to drop full expressiveness of the perturbation in all dimensions and use the primary Lyapunov exponent, or a subset of the largest Lyapunov exponents to approximate the

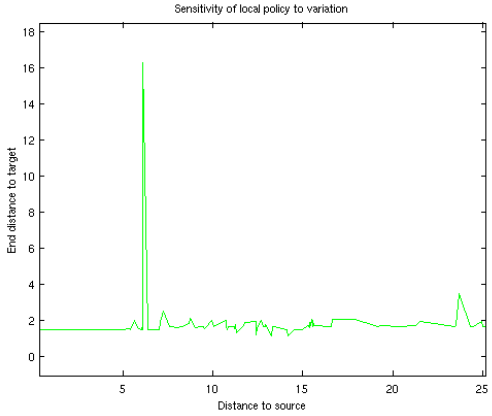
sensitivity in all dimensions.

For POMDP optimization the objective is to build a value function, which does not require chaining of perturbed beliefs. Policies are composed similar to the Bellman backup for one stage but for a plurality of stages, so only the running cost perturbation for each policy needs to be retained. A first order approximation of the running cost requires matrix-vector multiplication instead of matrix-matrix multiplication. Reducing the time complexity with this approach results in a running time for backup that is similar to the backup time complexity for single-stage backup of typical sampling-based value iteration POMDP optimization methods.

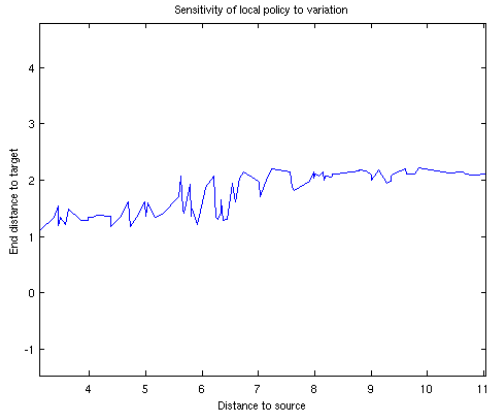
6.5 Results

One of the tenets of the presented approach is that, under general conditions, local feedback policies along with the inherent convergence, or forgetting, conditions of stochastic processes result in locally insensitive systems. We take advantage of this assumption to generate a temporal and spatial abstraction technique. To validate these properties, we present experimental sensitivity results for various benchmark problems.

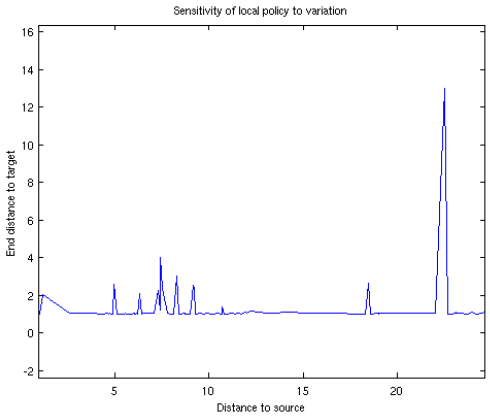
In the following results, a random hyperbelief was selected as a target for a greedy feedback policy. Then a initial hyperbelief was generated that was able to reach within the neighborhood of the target hyperbelief. Then 100 perturbed source hyperbeliefs were generated. The local policy was executed for each of the perturbed hyperbeliefs. The result of these simulations are depicted in Figure 6.8. The x-axis represents the result of the perturbed hyperbelief and their distance to the initial source hyperbelief sorted by initial distance. Lukaszyk-Karmowski metric [78] was used as the distance measure. This is a pseudo metric–it fails to satisfy the identity of the indiscernibles. Thus, the minimum distance is just a relative measure between two hyperbeliefs. The y-axis represents the distance to the target hyperbelief.



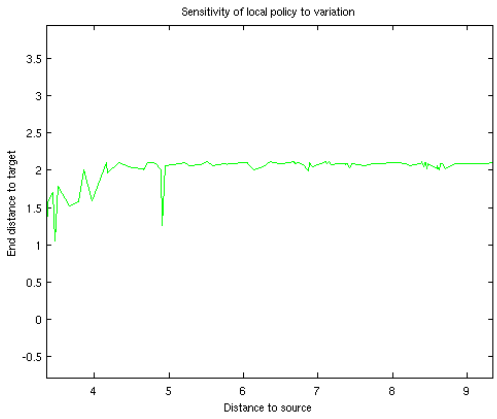
(a) First target hyperbelief for the Maze20 example



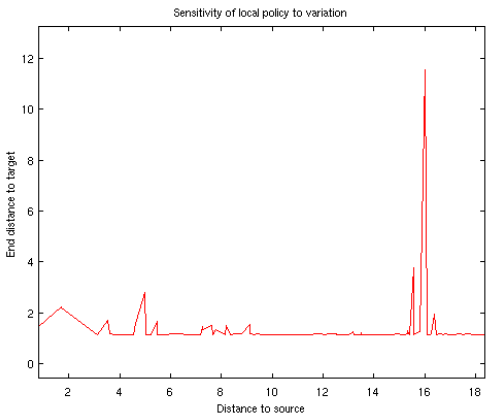
(b) First target hyperbelief for the CIT example



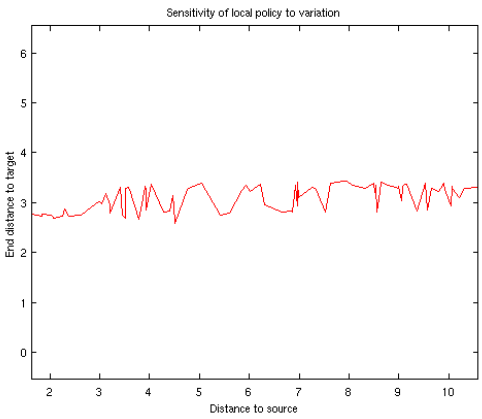
(c) Second target hyperbelief for the Maze20 example



(d) Second target hyperbelief for the CIT example



(e) Third target hyperbelief for the Maze20 example



(f) Third target hyperbelief for the CIT example

Figure 6.8: Experimental sensitivity results for two benchmark POMDP systems

There are two key observations regarding this result. First, the majority of the samples demonstrate pronounced insensitivity to the starting hyperbelief. Second, there are evident perturbed hyperbeliefs that fail to make significant progress towards the target. It is reasonable to assume that there are regions in the space that the greedy policy is insufficient to reach the target over the entire space. The perturbed hyperbelief may fall into one of these regions under a large enough perturbation. The discontinuities in the graph are likely due to sampling into several of these regions. It is interesting to note that the system is insensitive for similar distances, which implies are large perturbations for which the system is dissipative while simultaneously chaotic. The insensitivity for these examples for large perturbations demonstrates the a dense representation of the space is likely unnecessary.. In the following section will will provide support for this via several benchmark POMDP systems.

6.6 Conclusion

The perturbation analysis we presented establishes not only an approximation of the sensitivity of a forecasted evolution but also a methodology to chain existing forecasted evolutions together into a combined representation. The combined representation can be used to estimate the running cost and evolution without re-simulating each of the forecasted evolutions. Computational time complexity is the greatest hindrance to the proposed method. Each perturbation analysis retains a linear approximation of the running cost and perturbation to set of terminal beliefs. A formulation for higher order approximations was also provided, but computational requirements suggest that approximations greater than quadratic are likely of little use.

Even a linear approximation has a time complexity that is quadratic in the number of states describing the POMDP system. However, a linear approximation will be more efficient than resimulation if the stages of the forecasted evolution is large enough. Methods

to approximate the transition function, which requires matrix-matrix multiplication, may alleviate this to some extent. Future work may focus on developing a more efficient approximation using a subset of the Lyapunov exponents to estimate both an upper and lower bound on the running cost and perturbation of the forecasted evolution. Finally, we will investigate the effect of perturbations in the policy itself, via policy gradient methods, which may extend the presented analysis to fully capture the effect of perturbations on the system evolution.

6.7 Perturbation Analysis Proofs

We now present the analysis forecasted evolutions of POMDPs. First, we demonstrate how to generate a compact representation of the forecasted evolution that reduces the the formulation of a forecasted evolution to be expressed only in terms of the initial belief–not the beliefs from one stage to the next. This formulation is then used to analyze the effect of perturbations in the initial belief on the forecasted evolution. This core analysis is used earlier in Section 6.3.1. Next, we introduce tensor notation. The perturbation analysis that follows evaluates derivatives of vector fields. Formulating the derivative of vector fields using tensor notation simplifies the expression of these derivatives. Finally, using the tensor notation introduced, we derive results for the running cost sensitivity of forecasted evolutions, which is a generalized derivation of what we presented in Section 6.3.2 to nonlinear cost functions and higher order approximations.

6.7.1 Belief transition sensitivity

To determine the effect of a perturbation of an initial belief on the terminal belief for a given sample path, we will first determine the effect a perturbation has on a belief from one stage to the next. Then, we will determine the perturbation of the probability of a belief occurring from one stage to the next.

Remark 6.1. *The sensitivity of for a single stage (from k to $k+1$) due to perturbation Δb_k starting from b_k, subject to control action u_k and observation y_{k+1}, is given by*

$$
\begin{aligned}
b_{k+1} + \Delta b_{k+1} &= \frac{\phi_{u_k,y_{k+1}}(b_k + \Delta b_k)}{\mathbf{1}^T \phi_{u_k,y_{k+1}}(b_k + \Delta b_k)} \\
\Rightarrow \Delta b_{k+1} &= \frac{\phi_{u_k,y_{k+1}}(b_k + \Delta b_k)}{\mathbf{1}^T \phi_{u_k,y_{k+1}}(b_k + \Delta b_k)} - \frac{\phi_{u_k,y_{k+1}} b_k}{\mathbf{1}^T \phi_{u_k,y_{k+1}} b_k} && (6.44) \\
&= \frac{\phi_{u_k,y_{k+1}}(b_k + \Delta b_k) - \frac{\phi_{u_k,y_{k+1}} b_k \mathbf{1}^T \phi_{u_k,y_{k+1}}(b_k + \Delta b_k)}{\mathbf{1}^T \phi_{u_k,y_{k+1}} b_k}}{\mathbf{1}^T \phi_{u_k,y_{k+1}}(b_k + \Delta b_k)} && (6.45) \\
&= \frac{\phi_{u_k,y_{k+1}}(b_k + \Delta b_k) - \phi_{u_k,y_{k+1}} b_k - \frac{\phi_{u_k,y_{k+1}} b_k \mathbf{1}^T \phi_{u_k,y_{k+1}} \Delta b_k}{\mathbf{1}^T \phi_{u_k,y_{k+1}} b_k}}{\mathbf{1}^T \phi_{u_k,y_{k+1}}(b_k + \Delta b_k)} && (6.46) \\
&= = \frac{\phi_{u_k,y_{k+1}} \Delta b_k - \frac{\phi_{u_k,y_{k+1}} b_k \mathbf{1}^T \phi_{u_k,y_{k+1}} \Delta b_k}{\mathbf{1}^T \phi_{u_k,y_{k+1}} b_k}}{\mathbf{1}^T \phi_{u_k,y_{k+1}}(b_k + \Delta b_k)} && (6.47) \\
&= \frac{\phi_{u_k,y_{k+1}}(\mathbf{I} - \frac{b_k \mathbf{1}^T \phi_{u_k,y_{k+1}}}{\mathbf{1}^T \phi_{u_k,y_{k+1}} b_k}) \Delta b_k}{\mathbf{1}^T \phi_{u_k,y_{k+1}}(b_k + \Delta b_k)}. && (6.48)
\end{aligned}
$$

The first equation is the definition of the updated belief $b_{k+1} + \Delta b_{k+1}$ from $b_k + \Delta b_k$ for the observation y_{k+1} and action u_k as given in (2.7) in Section 2.1. Solving for the perturbation Δb_{k+1}, we obtain (6.44). Solving for a common denominator results in (6.45). Simplifying the equation to cancel out $\mathbf{1}^T \phi_{u_k,y_{k+1}} b_k$ that appears in both the numerator and denominator of the right hand side of the equation results in (6.46). Next, we obtain (6.47) by canceling like terms. Finally, (6.48) is obtained by pulling out like terms. Thus, we are able to represent of the perturbation Δb_{k+1} relative to the perturbation Δb_k.

The derivation above is for a single stage. We want, however, to derive the effect of a perturbation along the entire sample path from stage 0 to stage k. The evolution under an information vector I_k can be derived as a ratio of linear functions dependent on b_0. First, we introduce the notion of the composite belief transition function.

The *composite belief transition function* ϕ_{I_k} for information vector I_k is defined as

$$\phi_{I_k} = \phi_{y_k,u_{k-1}}\phi_{y_{k-1},u_{k-2}}\cdots\phi_{y_1,u_0}.$$

The belief at stage b_k can be expressed relative to b_0 using the composite belief transition function.

Proposition 6.1. *The belief b_k under the information vector $I_k = \{b_0, u_0, y_1, \ldots, u_{k-1}, y_k\}$ can be computed as*

$$b_k = \frac{\phi_{I_k} b_0}{\mathbf{1}^T \phi_{I_k} b_0}.$$

Proof. This elementary result follows by induction. First at stage 1,

$$b_1 = \frac{\phi_{y_1,u_0} b_0}{\mathbf{1}^T \phi_{y_1,u_0} b_0} = \frac{\phi_{I_1} b_0}{\mathbf{1}^T \phi_{I_1} b_0}.$$

At stage k, we can represent the b_k relative to $b_{k-1} = \frac{\phi_{I_{k-1}} b_0}{\mathbf{1}^T \phi_{I_{k-1}} b_0}$ as

$$b_k \;=\; \frac{\phi_{y_k,u_{k-1}} b_{k-1}}{\mathbf{1}^T \phi_{y_k,u_{k-1}} b_{k-1}} = \frac{\phi_{y_k,u_{k-1}} \frac{\phi_{I_{k-1}} b_0}{\mathbf{1}^T \phi_{I_{k-1}} b_0}}{\mathbf{1}^T \phi_{y_k,u_{k-1}} \frac{\phi_{I_{k-1}} b_0}{\mathbf{1}^T \phi_{I_{k-1}} b_0}} = \frac{\phi_{I_k} b_0}{\mathbf{1}^T \phi_{I_k} b_0},$$

where ϕ_{I_k} is from Definition 6.7.1. Thus, the result holds for all k. □

With a representation of the perturbation along a path, we now determine the probability of the sample path occurring. If each belief b is unique for each stage up to and including k, then the probability of belief b_k occurring is equal to the probability of the information vector $I_k \in \mathcal{I}_k$ occurring (i.e., the sample path along I_k), where I_k at stage k generated b_k. The probability of a belief under a given information vector evolves as a linear function of the initial belief. In fact the denominator in Proposition 6.1 is precisely $p(I_k \,|\, b_0)$.

Lemma A1. *Assuming that each sample path I_k produces a unique belief so that no belief ever occurs more than once, the probability of the information vector I_{k+1} occurring given*

policy π *and belief* b_k *is*

$$p(I_{k+1}|b_k) = \mathbf{1}^T O_{y_{k+1}} T_{u_k} b_k.$$

Proof. Expanding the definition of $p(I_{k+1}|b_k)$, we obtain:

$$\begin{aligned}
p(I_{k+1}|b_k) &= p(y_{k+1}, u_k, I_k|b_k) && (6.49)\\
&= p(y_{k+1}|u_k, I_k, b_k)p(u_k, I_k|b_k) && (6.50)\\
&= p(y_{k+1}|u_k, I_k, b_k)p(u_k|I_k, b_k)p(I_k|b_k) && (6.51)\\
&= p(y_{k+1}|u_k, b_k) && (6.52)\\
&= \int_{b\in\mathcal{P}_b} p(y_{k+1}|b, u_k, b_k)p(b|u_k, b_k) && (6.53)\\
&= p(y_{k+1}|b_{k+1|k})p(b_{k+1|k}|u_k, b_k) && (6.54)\\
&= \mathbf{1}^T O_{y_{k+1}} T_{u_k} b_k. && (6.55)
\end{aligned}$$

The first equation, (6.49), is the result of splitting I_{k+1} into y_{k+1}, u_k, and I_k. By conditioning on $p(u_k, I_k|b_k)$ and using properties of conditional probability the result in (6.50) is obtained. We then condition on I_k to obtain $p(u_k, I_k|b_k) = p(u_k|I_k, b_k)p(I_k|b_k)$. Next, using the fact that b_k is a sufficient statistic for I_k, we observe that $p(y_{k+1}|u_k, I_k, b_k) = p(y_{k+1}|u_k, b_k)$. Furthermore, $p(u_k|I_k, b_k) = p(u_k|b_k) = 1$ because we assume a deterministic policy, $\pi(b_k) = u_k$ with probability one. Also, by assumption of the uniqueness of b_k, $p(I_k|b_k) = 1$. Next, we marginalize over b in (6.53). The prediction step is deterministic, so

$$p(b_{k+1|k}|u_k, b_k) = \begin{cases} 1 & \text{if } b = b_{k+1|k} \\ 0 & \text{else} \end{cases}$$

where, from (2.5) in Section 2.1,

$$b_{k+1|k} = T_{u_k} b_k.$$

Inserting this result into (6.53) produces (6.54). Finally, $p(y_{k+1}|b_{k+1|k}) = \mathbf{1}^T O_{y_{k+1}} b_{k+1|k}$ from

(2.6) in Section 2.1. By joining these two terms together we obtain the stated result in (6.55). □

As the action is deterministic, the only new unknown in I_{k+1} over I_k is y_{k+1}. Thus, the result is the probability of the observation y_{k+1} occurring for the predicted belief $b_{k+1|k}$. Using this result along with the assumed uniqueness[8] of each belief, we can obtain the probability of the belief b_k occurring under the policy π.

Proposition 6.2. *Assuming that each belief is unique (implying that no other path reaches b_k except I_k), the probability of b_k occurring under policy π can be expressed as*

$$w_k = p(b_k|\pi, b_0) = p(I_k \,|\, \pi, b_0) = \mathbf{1}^T \phi_{I_k} b_0,$$

where $b_k = \frac{\phi_{I_k} b_0}{\mathbf{1}^T \phi_{I_k} b_0}$ and I_k is the unique sample path that results in b_k under the deterministic policy π.

Proof. First we marginalize the probability of the belief on the information vector I_k:

$$\begin{aligned} p(b_k|\pi, b_0) &= \sum_{I \in \mathcal{I}_k} p(b_k|I, \pi, b_0) p(I|\pi, b_0) \\ &= p(I_k|\pi, b_0) \end{aligned}$$

as we already assumed that b_k is unique for a single information vector I_k, thus

$$p(b_k|I, \pi, b_0) = \begin{cases} 1 & \text{if } I = I_k \\ 0 & \text{else} \end{cases}.$$

For notation purposes, we denote $p(I_k|\pi, b_0) = p(I_k|b_0)$.

[8]The analysis we present can be extended to the case where beliefs are not unique. However, including this capability unnecessarily complicates our analysis.

Now, using this result we show how to obtain $p(I_k \,|\, b_0) = \mathbf{1}^T \phi_{I_k} b_0$. This basic result follows from induction. Starting at stage 1,

$$p(I_1 \,|\, b_0) \;\; = \;\; \mathbf{1}^T O_{y_1} T_{u_0} b_0 = \mathbf{1}^T \phi_{I_1} b_0$$

from Lemma A1. Then at stage k, the probability of b_k occurring is provided by

$$\begin{aligned}
p(I_k \,|\, b_0) &= \int_{b_{k-1}} p(I_k \,|\, b_{k-1}) p(b_{k-1} \,|\, , b_0) \\
&= p(I_k \,|\, b_{k-1}) p(b_{k-1} \,|\, , b_0) && (6.56) \\
&= \mathbf{1}^T \phi_{y_k, u_{k-1}} b_{k-1} \mathbf{1}^T \phi_{I_{k-1}} b_0 && (6.57) \\
&= \mathbf{1}^T \phi_{y_k, u_{k-1}} \frac{\phi_{I_{k-1}} b_0}{\mathbf{1}^T \phi_{I_{k-1}} b_0} \mathbf{1}^T \phi_{I_{k-1}} b_0 && (6.58) \\
&= \mathbf{1}^T \phi_{y_k, u_{k-1}} \phi_{I_{k-1}} b_0 && (6.59) \\
&= \mathbf{1}^T \phi_{I_k} b_0. && (6.60)
\end{aligned}$$

The first equation is obtained by marginalizing on b_{k-1}. Since we assume each belief is unique, the integral reduces to a single instance in (6.56). Substituting $p(I_k \,|\, b_{k-1}) = \mathbf{1}^T \phi_{y_k, u_{k-1}} b_{k-1}$ from Lemma A1 and $p(b_{k-1} \,|\, , b_0) = \mathbf{1}^T \phi_{I_{k-1}} b_0$ from Proposition 6.2 into (6.56), we obtain (6.52). The result in (6.58) follows from substitution of Proposition 6.1 for b_{k-1} in terms of b_0 along sample path I_{k-1}. Simplifying we obtain (6.54) and, finally, (6.55) follows from the definition of ϕ_{I_k}. Thus, the result holds for all k. □

When only a subset of the possible sample paths are generated, i.e., $\tilde{\mathcal{I}}_k$, the total probability of the sample paths does not sum to one. The weight for the set of beliefs must then be normalized:

$$w_k^{(i)} = \frac{\mathbf{1}^T \phi_{I_k^{(i)}} b_0}{\sum_{I_k^{(j)} \in \tilde{\mathcal{I}}_k} \mathbf{1}^T \phi_{I_k^{(j)}} b_0}.$$

The notation $I_k^{(i)}$ represents the i-th sample path to stage k (and $b_k^{(i)}$ the corresponding belief along the same path) taken from the set $\tilde{\mathcal{I}}_k$. This normalization process results in a

dependence of the probability of each sample path on every other sample path. Moreover, the weight is no longer a linear function of the initial belief. Because we are evaluating the expected cost over the forecasted evolution, we can evaluate evolution and cost of the system relative to the probability of each sample path, as we demonstrate below in Section 6.7.3, instead of the expected cost at each stage k taken over all beliefs $\{b_k^{(i)}\}$.

When chaining forecasted evolutions together, we can add the cost along multiple sample paths and then evaluate the normalization relative to the composited sample paths. This formulation enables us to pull the normalization term, which is nonlinear, out of the approximation and the compositing process, which simplifies the analysis. However, this consideration of the normalization term is merely delayed until we chain running costs.

From Proposition 6.1, it can be seen that the belief at stage k is normalized along the sample path I_k. Normalization is an inverse sum over the probability of each belief occurring:

$$\eta(b_0) \equiv \frac{b_0}{\mathbf{1}^T b_0}.$$

To simply the expression of the belief at stage k by eliminating the denominator, we approximate the evolution along the path relative to the unnormalized belief using a first order Taylor series approximation.

Theorem A1. *For the belief normalizing function:*

$$\eta(\phi_{I_k} b_0) \equiv \frac{\phi_{I_k} b_0}{\mathbf{1}^T \phi_{I_k} b_0},$$

the first order derivative taken for the belief $\phi_{I_k} b_0$ is given by

$$\frac{\phi_{I_k}}{(\mathbf{1}^T \phi_{I_k} b_0)} \left(\mathbf{I} - \frac{b_0 \mathbf{1}^T \phi_{I_k}}{\mathbf{1}^T \phi_{I_k} b_0} \right).$$

Proof. We will establish via induction. First note that $\nabla_{b_0}\phi_{I_k}b_0 = \phi_{I_k}$. Then

$$\begin{aligned}
\nabla_{b_0}(\eta(\phi_{I_k}b_0)) &= \nabla_{b_0}\left(\frac{\phi_{I_k}b_0}{\mathbf{1}^T\phi_{I_k}b_0}\right) \\
&= \frac{1}{\mathbf{1}^T\phi_{I_k}b_0}\nabla_{b_0}\left(\phi_{I_k}b_0\right) + \phi_{I_k}b_0\nabla_{b_0}\left(\frac{1}{\mathbf{1}^T\phi_{I_k}b_0}\right) \\
&= \frac{\phi_{I_k}}{\mathbf{1}^T\phi_{I_k}b_0} - \frac{\phi_{I_k}b_0\mathbf{1}^T\phi_{I_k}}{\left(\mathbf{1}^T\phi_{I_k}b_0\right)^2} \\
&= \frac{\phi_{I_k}}{\left(\mathbf{1}^T\phi_{I_k}b_0\right)}\left(\mathbf{I} - \frac{b_0\mathbf{1}^T\phi_{I_k}}{\mathbf{1}^T\phi_{I_k}b_0}\right).
\end{aligned}$$

The second equation is just the definition of the product rule. The result of which is the third equation. Finally, simplifying the expression results in the last equation. □

Having established the effect perturbations have on the evolution of a POMDP, we now move to understanding the effect a perturbation has on the running cost of a POMDP. Analysis for first order approximations of linear cost functions is presented in Section 6.3.2. A general analysis for nonlinear cost functions an higher order approximations is presented below. However, before we can present this analysis, we must first introduce tensor notation, which will be used extensively in the following analysis.

6.7.2 Introduction to tensors

Tensors are multi-linear transformations defined over an underlying finite dimensional vector space. Tensors are a generalization of scalars, vectors, and matrices into higher dimensional spaces. Tensor representation is often used to express derivatives of vector fields. The following definitions and their properties can be found in [101].

Before we define tensors, we first introduce the notion of a dual space for vector fields.

Definition 6.1. *If given a finite vector space S over a field F, then the* dual space S^* *is the set of all linear maps from S to F . S^* has the same dimension as S.*

Definition 6.2. *Given a basis* $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ *in* S *and field* F*, the* dual basis S^* *is a set* $\{\varepsilon^1, \ldots, \varepsilon^n\}$

$$\varepsilon^i(\zeta_1\mathbf{e}_1 + \cdots + \zeta_n\mathbf{e}_n) = \zeta_i, \ i = 1, \ldots, n$$

for any choice of coefficients $\zeta_i \in F$*. This can be simplified by defining for each* i *the coefficient* ζ_i *equal to one and the rest of the coefficients equal to zero to obtain:*

$$\varepsilon^i(\mathbf{e}_j) = \delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases},$$

where δ *is the Kronecker delta function.*

This form has a natural interpretation: if we define the vector space S in $\mathbb{R}^n$ as the space of column vectors, the dual space is expressed as the row vectors (covectors).

With the definition of a dual space for a vector field established, we can now define tensors.

Definition 6.3. *A type* (n, m) *tensor* A*, with order* $n + m$*, is defined as*

$$A : \underbrace{S^* \times \cdots \times S^*}_{n \text{ times}} \times \underbrace{S \times \cdots \times S}_{m \text{ times}} \to \mathbf{R}$$

where S *is a vector space and* S^* *is the dual space of* S*. The result is linear in each of the arguments. A tensor can be represented as a multi-linear map. Given the multi-linear map* A *of type* (n, m) *to a basis* $\{\mathbf{e}_j\}$ *for* S *and a canonical dual basis* $\{\varepsilon^i\}$ *for* S^* *an* $n + m$ *dimensional array of components can be obtained:*

$$A^{i_1 \ldots i_n}_{j_1 \ldots j_m} \equiv A(\varepsilon^{i_1}, \ldots, \varepsilon^{i_n}, \mathbf{e}_{j_1}, \ldots, \mathbf{e}_{j_m}).$$

Note, that A *is a linear functional and* $A(\cdot)$ *denotes the function taken with respect to the input arguments. Different choices of basis will yield different components.*

The tensor product produces the tensor $A \otimes D$ from the tensors A and D, which have order n and m, respectively. The new tensor has an order that is the sum of the two orders: $n + m$. When described as multi-linear maps, the tensor product simply multiplies the two tensors, i.e.

$$(A \otimes D)(v_1, \ldots, v_n, v_{n+1}, \ldots, v_{n+m}) = A(v_1, \ldots, v_n)D(v_{n+1}, \ldots, v_{n+m}).$$

The result is a map that is linear in all its arguments. When expressed in the component form, the effect similarly is to multiply the components of the two input tensors:

$$(A \otimes D)^{i_1 \ldots i_l i_{l+1} \ldots i_{l+m}}_{j_1 \ldots j_g j_{g+1} \ldots j_{g+n}} = A^{i_1 \ldots i_l}_{j_1 \ldots j_g} D^{i_{l+1} \ldots i_{l+m}}_{j_{g+1} \ldots j_{g+n}}.$$

If A is of type (g, l) and D is of type (n, m), then the tensor product $A \otimes D$ is type $(g+n, l+m)$.

While all properties of order zero tensors do not apply to higher order variations, several key properties hold for tensors of arbitrary order. In particular, both the associative and distributive laws hold:

$$\text{associative:} \begin{cases} (A \otimes D) \otimes H &= A \otimes (D \otimes H) \end{cases}$$

$$\text{distributive:} \begin{cases} A \otimes (D + H) &= A \otimes D + A \otimes H \\ (A + D) \otimes H &= A \otimes H + D \otimes H, \end{cases}$$

whenever A, D, and H are defined such that these operations make sense. We will make extensive these two properties in the analysis that follows.

In the following analysis we will also use the generalized dot product, or contraction operator for tensors.

Definition 6.4. *Let A be a type (g, l) tensor and$D$ be a type (n, m) tensor. Then the* dot

product $A \cdot B$ *results in a type* $(g-m, l-n)$ *tensor, which, as given in component form, is*

$$A \cdot D = A^{i_1 \ldots i_g}_{j_1 \ldots j_l} D^{i_{l-m+1} \ldots i_l}_{j_{g-n+1} \ldots j_g},$$

which holds when $n \leq g$ *and* $m \leq l$*. Thus, the order of the tensors subtract, reducing the overall order of the resulting tensor.*

Having established the core principles of tensor theory that we use in the analysis in the following sections, we now move to understanding the sensitivity of the belief transition function.

6.7.3 Running cost sensitivity for nonlinear and higher order approximations

Besides generating the effect of a perturbation on the set of terminal beliefs, we need to represent the effect a perturbation has on the cost of a forecasted evolution. The total cost for a time horizon K under policy π is given by

$$\begin{aligned} V_\pi(b_0) &= E\left[\sum_{k=0}^{K} c(b_k, \pi(b_k)) \,|\, b_0\right] \\ &= \sum_{k=0}^{K} \sum_{I_k^{(i)} \in \mathcal{I}_k} c(b_k^{(i)}, u_k^{(i)}) p(I_k^{(i)} | b_0). \end{aligned}$$

In Section 6.3.2, we derived a formulation that approximates the running cost at each stage. However, to derive the running cost sensitivity for general, nonlinear cost functions, we first select a form of the total cost that enables us take the expectation over the set of sample paths.

Theorem A2. *The expected total cost*

$$V(b_0) = E\left[\sum_{k=0}^{K} c(b_k, \pi(b_k)) \mid b_0\right]$$
$$= \sum_{k=0}^{K} \sum_{I_k^{(i)} \in \mathcal{I}_k} c(b_k^{(i)}, u_k^{(i)}) p(I_k^{(i)} | b_0)$$

for a time horizon K under policy π can alternately be defined as

$$V(b_0) = \sum_{I_K^{(i)} \in \mathcal{I}_K} \sum_{k=0}^{K} c(b_k^{(i)}, u_k^{(i)}) p(I_K^{(i)} | b_0).$$

Proof. To prove this we will reduce one form to the other. Before we begin, we use the note that $I_{k+1,K}^{(i)} = \{u_k, y_{k+1}, \ldots u_{K-1}, y_K\}$. Starting with the sample path representation, we observe:

$$\begin{aligned}
\sum_{I_K^{(i)} \in \mathcal{I}_K} \sum_{k=0}^{K} c(b_k^{(i)}, u_k^{(i)}) p(I_K^{(i)} | b_0) &= \sum_{I_K^{(i)} \in \mathcal{I}_K} \sum_{k=0}^{K} c(b_k^{(i)}, u_k^{(i)}) p(I_K^{(i)} | I_k^{(i)}) p(I_k^{(i)} | b_0) \\
&= \sum_{k=0}^{K} \sum_{I_{k+1,K}^{(i)}} \sum_{I_K^{(i)} \in \mathcal{I}_K} c(b_k^{(i)}, u_k^{(i)}) p(I_K^{(i)} | I_k^{(i)}) p(I_k^{(i)} | b_0) \\
&= \sum_{k=0}^{K} \sum_{I_K^{(i)} \in \mathcal{I}_K} c(b_k^{(i)}, u_k^{(i)}) \left[\sum_{I_{k+1,K}^{(i)}} p(I_K^{(i)} | I_k^{(i)}) \right] p(I_k^{(i)} | b_0) \\
&= \sum_{k=0}^{K} \sum_{I_K^{(i)} \in \mathcal{I}_K} c(b_k^{(i)}, u_k^{(i)}) p(I_k^{(i)} | b_0).
\end{aligned}$$

The first equation follows from Bayes rule: $p(c|d)p(d) = p(c, d)$. In the second equation, we break the summation into two parts: stage 0 to stage k and stage $k+1$ to stage K. Each $I_k^{(i)}$ is summed over all permutations of $I_{k+1,K}^{(i)}$. Next, in third equation, the summation of $\sum_{I_{k+1,K}^{(i)}} p(I_K^{(i)} | I_k^{(i)})$ is independent of $b_k$and $p(I_k^{(i)} | b_0)$, so we pull the summation into the

inside of the equation. Since we are summing over all possible $I^{(i)}_{k+1,K}$, $\sum_{I^{(i)}_{k+1,K}} p(I^{(i)}_K|I^{(i)}_k) = 1$. Thus, we arrive at the final equation, which is common representation of the expect total cost. □

This formulation is a direct result of the Markov property of POMDPs.

We approximate the complete forecasted evolution using ensemble forecasting, which generates only a subset of sample paths $\tilde{\mathcal{I}}_K$. To approximate the expectation over $\tilde{\mathcal{I}}_K$ we need to normalize the results because the total probability of the subset of sample paths is less than one:

$$V(b_0) \approx \frac{1}{P} \sum_{I^{(i)}_K \in \tilde{\mathcal{I}}_K} v_{I^{(i)}_K}(b_0), \tag{6.61}$$

which follows when using unbiased sampling and there are P sample paths.

When evaluating the effect of a perturbation Δb_0 of the initial belief b_0 has on the running cost $V_\pi(b_0 + \Delta b_0)$, the bias introduced by reusing the existing sample paths must be considered. In Section 6.3.1 we derived the bias for each belief at each stage. However, we have reduced the formulation to just weighting each sample path (and not each belief at each stage). Thus, we only need to reduce the bias for each sample path.

$$\begin{aligned} V(b_0 + \Delta b_0) &\approx \frac{1}{\mu} \sum_{I^{(i)}_K \in \tilde{\mathcal{I}}_K} v_{I^{(i)}_K}(b_0 + \Delta b_0) \left(\frac{p(I^{(i)}_K|b_0 + \Delta b_0)}{p(I^{(i)}_K|b_0 + \Delta b_0)} \right) \\ &= \frac{1}{\mu} \sum_{I^{(i)}_K \in \tilde{\mathcal{I}}_K} v_{I^{(i)}_K}(b_0 + \Delta b_0) \left(\frac{\mathbf{1}^T \phi_{I^{(i)}_K}(b_0 + \Delta b_0)}{\mathbf{1}^T \phi_{I^{(i)}_K} b_0} \right) \\ &= \sum_{I^{(i)}_K \in \tilde{\mathcal{I}}_K} v_{I^{(i)}_K}(b_0 + \Delta b_0) \frac{\left(\eta^{(i)}_K\right)^T (b_0 + \Delta b_0)}{\eta^T (b_0 + \Delta b_0)}, \end{aligned} \tag{6.62}$$

where the row vector $\left(\eta^{(i)}_K\right)^T = \frac{\mathbf{1}^T \phi_{I^{(i)}_K}}{\mathbf{1}^T \phi_{I^{(i)}_K} b_0}$ is the perturbed weight for each sample path and

$\eta^T = \sum_{I_K^{(i)} \in \tilde{\mathcal{I}}_K} (\eta^{(i)})^T$ is a normalizing factor.

By choosing to represent the expected running cost in this manner, we are able to nearly decouple the analysis along sample paths. The only coupling between sample paths occurs through $\mu + \Delta\mu$. Alternately, if the expectation of the running cost is taken as the average along every stage (verses every path), coupling occurs at each stage through a normalization term. By reducing the coupling to just the normalization along the set of paths, we can derive a compact representation of the running cost along the set of sample paths that is a function of the initial belief–not all the beliefs at each stage in the sample path. This formulation enables a simple and effective method for chaining forecasted evolutions together.

To determine the sensitivity of the forecasted evolution, we first determine the sensitivity of a single sample path. To do that we need to derive the sensitivity of the cost function relative to each belief along the path. From Proposition 6.1, it can be seen that the belief at stage k is normalized along the sample path I_k. To simplify the analysis of perturbations on the running cost, specifically for determination of the sensitivity functions associated with each sample path, we will decompose the cost function into a modified form that is the composition of the original cost function and the belief normalization function. Thus, for any $c(\cdot,\cdot)$ and belief normalizing function $\eta(\bar{b}) = \frac{\bar{b}}{\mathbf{1}^T\bar{b}} \in \mathcal{P}_b$, we represent the cost as $(c \circ \eta)(\bar{b}, u) = c(b, u)$ for any action $u \in \mathcal{U}$ and $b = \frac{\bar{b}}{\mathbf{1}^T\bar{b}}$, $\bar{b} \in \mathbb{R}_+^{|\mathcal{X}|}$ (positive quadrant of the $\mathbb{R}^{|\mathcal{X}|}$, where $|\mathcal{X}|$ is the number of states in the state-space).

Before we continue with analysis of the cost, we establish properties of the belief normalizing function that will be useful later. The following is a generalization of Theorem A1 to higher order derivatives.

Theorem A3. *For the belief normalizing function:*

$$\eta(\bar{b}) \equiv \frac{\bar{b}}{\mathbf{1}^T\bar{b}},$$

where $\bar{b} \in \mathbb{R}^m_+$*, the n-th order derivative is given by*

$$\frac{(-1)^n n!}{\left(\mathbf{1}\cdot\bar{b}\right)^n}\left(\mathbf{I}-\frac{\bar{b}\otimes\mathbf{1}}{\mathbf{1}\cdot\bar{b}}\right)\otimes\mathbf{1}^{\otimes(n-1)},$$

where $\mathbf{1}^{\otimes n}=\underbrace{\mathbf{1}\otimes\mathbf{1}\otimes\cdots\otimes\mathbf{1}}_{n\ times}$.

Proof. We will establish via induction. First note that $\nabla_{\bar{b}}(\bar{b})=\mathbf{I}$, and that , $\nabla^n_{\bar{b}}(\bar{b})=0$ for $n>1$ (as $\bar{b}$ is a linear function). Then for $n=1$,

$$\begin{aligned}\nabla_{\bar{b}}(\eta(\bar{b}))=\nabla_{\bar{b}}\left(\frac{\bar{b}}{\mathbf{1}\cdot\bar{b}}\right)=&\frac{1}{\mathbf{1}\cdot\bar{b}}\nabla_{\bar{b}}\bar{b}+\bar{b}\otimes\nabla_{\bar{b}}\left(\frac{1}{\mathbf{1}\cdot\bar{b}}\right)\\=&\frac{1}{\mathbf{1}\cdot\bar{b}}\mathbf{I}-\frac{\bar{b}\otimes\mathbf{1}}{\left(\mathbf{1}\cdot\bar{b}\right)^2}\\=&\frac{1}{\mathbf{1}\cdot\bar{b}}\left(\mathbf{I}-\frac{\bar{b}\otimes\mathbf{1}}{\mathbf{1}\cdot\bar{b}}\right).\end{aligned}$$

Assume that the form holds for the n-th derivative. Then

$$\begin{aligned}\nabla^{n+1}_{\bar{b}}(\eta(\bar{b})) &= \nabla_{\bar{b}}\left[\nabla^n_{\bar{b}}(\eta(\bar{b}))\right]=\nabla_{\bar{b}}\left[\frac{(-1)^{n-1}n!}{\left(\mathbf{1}\cdot\bar{b}\right)^n}\left(\mathbf{I}-\frac{\bar{b}\otimes\mathbf{1}}{\mathbf{1}\cdot\bar{b}}\right)\otimes\mathbf{1}^{\otimes(n-1)}\right]\\
&= -\frac{(-1)^{n-1}n!n}{\left(\mathbf{1}\cdot\bar{b}\right)^{n+1}}\mathbf{I}\otimes\mathbf{1}^{\otimes(n-1)}\otimes\mathbf{1}-\frac{(-1)^{n-1}n!}{\left(\mathbf{1}\cdot\bar{b}\right)^{n+1}}\mathbf{I}\otimes\mathbf{1}\otimes\mathbf{1}^{\otimes(n-1)}\\
&\quad+\frac{(-1)^{n-1}n!(n+1)}{\left(\mathbf{1}\cdot\bar{b}\right)^{n+2}}\bar{b}\otimes\mathbf{1}^{\otimes(n-1)}\otimes\mathbf{1}\\
&= \frac{(-1)^n(n+1)!}{\left(\mathbf{1}\cdot\bar{b}\right)^{n+1}}\left(\mathbf{I}-\frac{\bar{b}\otimes\mathbf{1}}{\mathbf{1}\cdot\bar{b}}\right)\otimes\mathbf{1}^{\otimes n}.\end{aligned}$$

Applying the chain rule to the first equation we arrive at the second equation. The simplified formula of the third equation is the stated result. Therefore, the form holds for $n+1$ and, thus, for all n. □

Corollary A1. *The n-th derivative of normalization of a belief under the sample path generated by* I_k *is*

$$\nabla^n_{\bar{b}}\phi_{I_k} = \nabla^n_{\bar{b}}\left(\frac{\phi_{I_k}\bar{b}}{\mathbf{1}^T\phi_{I_k}\bar{b}}\right) = \frac{(-1)^{n-1}n!}{\left(\mathbf{1}\cdot\phi_{I_k}\cdot\bar{b}\right)^n}\phi_{I_k}\cdot\left(\mathbf{I} - \frac{\bar{b}\otimes\mathbf{1}\cdot\phi_{I_k}}{\mathbf{1}\cdot\phi_{I_k}\cdot\bar{b}}\right)\otimes\left(\mathbf{1}\cdot\phi_{I_k}\right)^{\otimes(n-1)}.$$

This follows from substituting $\phi_{I_k}\bar{b}$ for $\bar{b}$ in Theorem A3.

Corollary A2. *If the n-th derivative of normalization function is taken relative to a normalized belief such that $|\bar{b}| = 1$, then Theorem A3 reduces to*

$$(-1)^n(n+1)!\left(\mathbf{I} - \bar{b}\otimes\mathbf{1}\right)\otimes\mathbf{1}^{\otimes(n-1)}.$$

The result in Theorem A3 is a specific instantiation of the generalized Leibniz rule, which describes the nth order derivative of the product of two functions. We can see that there is little additional information added as we increase the rank of the derivative–all that occurs is that the tensor is copied into a higher dimensional space (via $\mathbf{1}^{\otimes(n-1)}$) and scaled (via $(-1)^{n-1}(n+1)!$). Intuitively this makes sense as the nonlinearity is the result of a scalar value in the denominator.

Approximating the running cost along the path relative to the unnormalized belief using a Taylor series approximation, requires that we have the representation of the n-th order derivative of the normalizing function. To determine the effect of a perturbation on the running cost for each sample path, we must determine the effect of the perturbation at each stage along the sample path. Each perturbation must be normalized. Thus, we will derive a represent ion of the n-th derivative of the cost $c(\cdot)$ with respect to $\bar{b}_k$ relative to the normalization function:

$$\nabla^n_{\bar{b}_k} c = \nabla^n_{\bar{b}_k}(c\circ\eta).$$

Unfortunately, there exists no tensor extension or the composite derivative (via the chain rule) of two vector field functions. There exist various multivariate representations, specifically [102] generates a multi-index, combinatorial formulation. However, this assumes a

scalar function composed with a scalar field.

Theorem A4. *Faà di Bruno's formula: Let $y = g(x)$. Then the following identity holds, assuming all necessary derivatives are defined,*

$$\frac{d^n}{dx^n} f(g(x)) = \sum \frac{n!}{m_1! m_2! \cdots m_n!} \frac{d^{m_1+m_2+\cdots+m_n}}{dy^{m_1+m_2+\cdots+m_n}} f(g(x)) \prod_{j=1}^{n} \left(\frac{1}{j!} \frac{d^j}{dx^j} g(x) \right)^{m_j},$$

where m_i are the order of the monomial terms for the derivative taken with respect to x^i and where $\sum_{i=1}^{n} i m_i = n$.

Unfortunately, we cannot formulate the tensor representation relative to the composite representation for scalar functions as defined above. Instead, we derive the formulation manually.

Fortunately, the composite of the cost function with the belief normalizing function has specific properties that simplify the representation of the n-th derivative.

Lemma A2. *The product of the the r-th and s-th derivative of the belief normalizing function is such that*

$$\nabla^r \eta \cdot \nabla^s \eta = \begin{cases} \frac{-1}{(\mathbf{1} \cdot \bar{b})} \nabla^s \eta & \text{if } r = 1 \\ \mathbf{0} & \text{otherwise} \end{cases}.$$

Proof. First we will explore the case where $r = 1$. From Theorem A3, we can derive

$$\begin{aligned} \nabla \eta \cdot \nabla^s \eta &= \frac{-1}{(\mathbf{1} \cdot \bar{b})} \left(\mathbf{I} - \frac{\bar{b} \otimes \mathbf{1}}{\mathbf{1} \cdot \bar{b}} \right) \frac{(-1)^{s-1} s!}{(\mathbf{1} \cdot \bar{b})^s} \left(\mathbf{I} - \frac{\bar{b} \otimes \mathbf{1}}{\mathbf{1} \cdot \bar{b}} \right) \otimes \mathbf{1}^{\otimes(s-1)}, \\ &= \frac{-(-1)^{s-1} s!}{(\mathbf{1} \cdot \bar{b}) (\mathbf{1} \cdot \bar{b})^s} \left(\mathbf{I} - 2 \frac{\bar{b} \otimes \mathbf{1}}{\mathbf{1} \cdot \bar{b}} + \frac{\bar{b} \otimes \mathbf{1} \cdot \bar{b} \otimes \mathbf{1}}{(\mathbf{1} \cdot \bar{b})^2} \right) \otimes \mathbf{1}^{\otimes(s-1)}, \\ &= \frac{-(-1)^{s-1} s!}{(\mathbf{1} \cdot \bar{b}) (\mathbf{1} \cdot \bar{b})^s} \left(\mathbf{I} - \frac{\bar{b} \otimes \mathbf{1}}{\mathbf{1} \cdot \bar{b}} \right) \otimes \mathbf{1}^{\otimes(s-1)} \\ &= \frac{-1}{\mathbf{1} \cdot \bar{b}} \nabla^s \eta. \end{aligned}$$

Now, for when $r > 1$:

$$\begin{aligned}\nabla^r\eta \cdot \nabla^s\eta &= \frac{(-1)^{r-1}}{\left(\mathbf{1}\cdot\bar{b}\right)^r}\left(\mathbf{I} - \frac{\bar{b}\otimes\mathbf{1}}{\mathbf{1}\cdot\bar{b}}\right)\otimes\mathbf{1}^{\otimes(r-1)} \cdot \frac{(-1)^{s-1}}{\left(\mathbf{1}\cdot\bar{b}\right)^s}\left(\mathbf{I} - \frac{\bar{b}\otimes\mathbf{1}}{\mathbf{1}\cdot\bar{b}}\right)\otimes\mathbf{1}^{\otimes(s-1)}, \\ &= \frac{(-1)^{r+s-2}}{\left(\mathbf{1}\cdot\bar{b}\right)^{r+s}}\left(\mathbf{I} - \frac{\bar{b}\otimes\mathbf{1}}{\mathbf{1}\cdot\bar{b}}\right)\otimes\mathbf{1}^{\otimes(r-2)}\otimes\mathbf{1}\cdot\left(\mathbf{I} - \frac{\bar{b}\otimes\mathbf{1}}{\mathbf{1}\cdot\bar{b}}\right)\otimes\mathbf{1}^{\otimes(s-1)}.\end{aligned}$$

By pulling out one of the tensor products with the one vector we can see that

$$\mathbf{1}\cdot\left(\mathbf{I} - \frac{\bar{b}\otimes 1}{\mathbf{1}\cdot\bar{b}}\right) = \mathbf{1} - \mathbf{1} = \mathbf{0}.$$

Therefore, for $r > 1$, the product of the rth and sth derivative is the zero vector, or

$$\nabla^r\eta \cdot \nabla^s\eta = \mathbf{0}.$$

The stated result is thus proved. □

Lemma A3. *The product of* $\mathbf{r} = \{r_1, \ldots, r_s\}$ *derivatives of the belief normalizing function is*

$$\prod_{i=1}^{s}\nabla^{r_i}\eta = \begin{cases}\frac{(-1)^s}{\left(\mathbf{1}\cdot\bar{b}\right)^s}\nabla^{r_s}\eta & \text{if } r_1, \ldots r_{s-1} = 1 \\ 0 & \text{otherwise}\end{cases}.$$

Proof. From Lemma A2 it follows that if any $r_{s-1} > 1$ for $1 \leq i \leq s-1$ then the product of $\nabla^{r_i}\nabla^{r_{i+1}} = 0$ and, thus, that the total product equals zero. This implies only r_s can be greater than one. The product then reduces to the product of $s-1$ first order derivatives, yielding

$$\frac{(-1)^{s-1}}{\left(\mathbf{1}\cdot\bar{b}\right)^{s-1}}\nabla\eta.$$

The product of this result is then $\nabla^{r_s}\eta$ taken:

$$\frac{(-1)^{s-1}}{\left(\mathbf{1}\cdot\bar{b}\right)^{s-1}}\left(\nabla\eta\nabla^{r_s}\eta\right) = \frac{(-1)^{s-1}}{\left(\mathbf{1}\cdot\bar{b}\right)^{s-1}}\left(\frac{(-1)}{\left(\mathbf{1}\cdot\bar{b}\right)}\nabla^{r_s}\eta\right).$$

And so we arrive at the stated result. □

The results from both Lemma A3 indicates that the combinatorial explosion of terms does arise in the derivation of the chained cost analysis. Thus, we can obtain a more compact representation of the derivative of the normalized cost using the above result of Lemma A3 in combination with Corollary A1.

Theorem A5. *The n-th derivative of the normalized cost along path I_k and action u_k is*

$$\nabla^1_{b_0} c(\phi_{I_k} b_0, u_k) = \nabla c_{u_k} \cdot (\phi) \cdot \nabla \phi_{I_k},$$

when $n = 1$. Otherwise for $n > 1$ the derivative is

$$\nabla^n_{b_0} c(\phi_{I_k} b_0, u_k) \quad = \quad \nabla^n c_{u_k} \cdot (\nabla \phi)^n + \nabla c_{u_k} \cdot (\phi) \cdot \nabla^n \phi_{I_k}.$$

We will abuse notation and represent $\nabla^n c(\phi_{I_k} b_0, u_k) = \nabla^n_{b_0} c_{u_k}$, where the dependence of the derivative on the information vector I_k is implicit.

Proof. First, the Jacobian is given by

$$\nabla_{b_0} c(\phi_{I_k} b_0, u_k) = \nabla c_{u_k} \nabla \phi_{I_k},$$

which follows from the chain-rule. The second derivative is

$$\nabla^2_{b_0} c(\phi_{I_k} b_0, u_k) = \nabla^2 c_{u_k} \cdot (\nabla \phi_{I_k})^2 + \nabla c_{u_k} \cdot \nabla^2 \phi_{I_k},$$

which again follows from direct application of the chain-rule. Now, we will proceed by

induction. Assuming that the the result holds for n, then for $n+1$ we have

$$\begin{aligned}
\nabla_{b_0}^{n+1} c(\phi_{I_k} b_0, u_k) &= \nabla \left[\nabla_{b_0}^{n} (c_{u_k} \circ \phi(b_0, I_k)) \right] \\
&= \nabla \left[\nabla^n c_{u_k} \cdot (\nabla \phi_{I_k})^n + \nabla c_{u_k} \cdot \nabla^n \phi_{I_k} \right] \\
&= \nabla^{n+1} c_{u_k} \cdot (\nabla \phi_{I_k})^{n+1} + n \nabla^n c_{u_k} \cdot (\nabla \phi_{I_k})^n \cdot \nabla \phi_{I_k} \\
&\quad \nabla^2 c_{u_k} \cdot \nabla^n \phi_{I_k} \cdot \nabla \phi_{I_k} + \nabla c_{u_k} \cdot (\phi) \cdot \nabla^{n+1} \phi_{I_k} \\
&= \nabla^n c_{u_k} \cdot (\phi)^n \cdot \nabla^n \phi_{I_k} + \nabla c_{u_k} \cdot (\phi) \cdot \nabla^n \phi_{I_k}.
\end{aligned}$$

All terms that are the product of $\nabla^s \phi_{I_k} \cdot \nabla^r \phi_{I_k} = 0$ from Lemma A3 as $s + r > 1$. Thus, we arrive at the stated result. □

Corollary A3. *The Jacobian of $c(\phi_{I_k} b_0, u_k)$ is*

$$\nabla_{b_0} c(b_k, u_k) = \nabla_b c_{u_k} \cdot \frac{1}{\mathbf{1}^T \phi_{I_k} b_0} \left(\mathbf{I} - \frac{\phi_{I_k} b_0 \otimes \mathbf{1}}{\mathbf{1}^T \phi_{I_k} b_0} \right).$$

Corollary A4. *The Hessian of $c(\phi_{I_k} b_0, u_k)$ is*

$$\nabla_{b_0}^2 c(b_k, u_k) = \frac{1}{(\mathbf{1}^T \phi_{I_k} b_0)^2} \left[\nabla^2 c_{u_k} \cdot \left(\mathbf{I} - \frac{\phi_{I_k} b_0 \otimes \mathbf{1}}{\mathbf{1}^T \phi_{I_k} b_0} \right) - \nabla c_{u_k} \cdot \left(\mathbf{I} - \frac{\phi_{I_k} b_0 \otimes \mathbf{1}}{\mathbf{1}^T \phi_{I_k} b_0} \right) \otimes \mathbf{1} \right].$$

To achieve the desired compact representation along each sample path we represent the cost function via a polynomial approximation, which leverages Theorem A5. Polynomial approximation is selected because the coefficients of the monomial terms can be summed over each stage to generate a polynomial representation of the running cost over all stages.

Remark 6.2. *The cost $c(\bar{b}_k, u_k)$ at stage k around $\bar{b}_k$ under the control action u_k can be approximated by the n-th degree polynomial of the the form*

$$c(\bar{b}_k + \Delta\bar{b}_k, u_k) \;\; = \;\; c(\bar{b}_k, u_k) + \sum_{n=1}^{N} \frac{1}{n!} \nabla^n_{\bar{b}_k} c_{u_k} \cdot \left(\Delta\bar{b}_k\right)^n + c_{err}(\Delta\bar{b}_k), \tag{6.63}$$

where $c_{err,k}(\cdot)$ is the residual for the higher order, greater than n, approximation terms. The tensor product recursively taken n times with $\Delta\bar{b}_k$ is denoted as Thus, $\nabla^n_{\bar{b}_k} c_{u_k} \cdot (\Delta\bar{b}_k)^n$. This follows directly from application of Taylor's theorem–assuming all necessary derivatives exist.

The result from (6.63) enables the analysis of the perturbation $\Delta\bar{b}_k$ at each stage. However, to generate a compact representation we need to formulate the perturbation at every stage k relative to the perturbation at the the initial stage 0 via Δb_0. This will be used to create an analytic representation along each sample path, thus enabling us to achieve temporal abstraction. In this context, temporal abstraction denotes the ability to represent the future evolution at some future stage k relative to the initial stage 0 without having to iterate through each intermediate stage $1, \ldots k-1$.

Theorem A6. *The cost function along the information vector $I_K = \{b_0, u_0, y_1, \cdots, y_K\}$ can be approximated by the n-th order polynomial relative to $\Delta\bar{b}_0$ as*

$$v_{I_K}(b_0 + \Delta b_0) \approx v_{I_K,sim} + \sum_{n=1}^{n} \frac{1}{n!} \nabla^n v_{I_K} \cdot \left(\Delta\bar{b}_0\right)^n + v_{I_K,err}(\Delta\bar{b}_0)$$

from the set of n-th order approximations of the cost at each stage $k = 0, \ldots, K$ represented by (6.63).

Proof. Using a Taylor series expansion, we can approximate the cost $c(\bar{b}_k, u_k)$ at stage k around $\bar{b}_k$ under the control action u_k as (6.63), where we use the notation $c(\bar{b}, u)$ as shorthand notation for $c(\eta(\bar{b}), u)$. The n-th order derivative is represented by n-tensor.

The approximated cost along the sample path I_K can then be approximated with

$$v_{I_K}(\Delta\bar{b}_0) = \sum_{k=0}^{K} c(\bar{b}_k + \Delta\bar{b}_k, u_k) \tag{6.64}$$

$$= \sum_{k=0}^{K} \left\{ c(\bar{b}_k, u_k) + \sum_{n=1}^{n} \frac{1}{n!} \nabla^n_{\bar{b}_k} c_{u_k} \cdot \left(\Delta\bar{b}_k\right)^n + c_{err}(\Delta\bar{b}_k) \right\} \tag{6.65}$$

by summing over all stages $k = 0, \ldots, K$.

The unnormalized belief transition function from stage 0 to stage k is $\bar{b}_k = \phi_{I_k} b_0$. The unnormalized perturbation of each belief at stage k can be represented as a linear function of the perturbation at the initial stage, i.e. $\Delta\bar{b}_k = \phi_{I_k}\Delta b_0$. Substituting this into the approximated cost function at stage k (6.65), we obtain

$$c(\bar{b}_k + \Delta\bar{b}_k, u_k) = c(\bar{b}_k, u_k) + \sum_{n=1}^{n} \frac{1}{n!} \nabla^n_{\bar{b}_k} c_{u_k} \cdot \left(\phi_{I_k}\Delta\bar{b}_0\right)^n + \bar{c}_{err}(\phi_{I_k}\Delta\bar{b}_0),$$

which is now a function of the initial perturbation Δb_0. Now, by replacing the cost at each stage k in (6.65) with the representation in equation above, the sensitivity

running cost function along the sample path I_K becomes

$$v_{I_K}(\bar{b}_0 + \Delta\bar{b}_0) = \sum_{k=0}^{K} \left\{ c(\bar{b}_k, u_k) + \sum_{n=1}^{n} \frac{1}{n!} \nabla^n_{\bar{b}_k} c_{u_k} \cdot (\phi_{I_k}\Delta b_0)^n + \bar{c}_{err}(\phi_{I_k}\Delta b_0) \right\}$$

$$= \underbrace{\sum_{k=0}^{K} c(\bar{b}_k, u_k)}_{v_{I_K,sim}} + \sum_{n=1}^{n} \frac{1}{n!} \underbrace{\left[\sum_{k=0}^{K} \nabla^n_{\bar{b}_k} c_{u_k} \cdot (\phi_{I_k})^n \right]}_{\nabla^n_{b_0} v_{I_K}} \cdot (\Delta b_0)^n + \underbrace{\sum_{k=0}^{K} \bar{c}_{err}(\phi_{I_k}\Delta b_0)}_{v_{I_K,err}(\Delta b_0)}.$$

The second equation follows from grouping like terms to obtain: $v_{I_K,sim}$, $\nabla^n_{b_0} v_{I_K}$, and $v_{I_K,err}(\Delta b_0)$. The base cost, v_{sim}, which is the simulated cost under no perturbation along the sample path I_K, is a constant at each stage, therefore, the sum over all stages can be performed to generate a single constant term. The $n-$th order derivative at each stage $k = 0, \ldots K$ is product of n times ϕ_{I_k}. The n-th order derivative of the cost function $\nabla^n_{\bar{b}_k} c_{u_k}$

is $n+1$ dimensional and is symmetric in the 2 to $n+1$ dimensions. The product of n times ϕ_{I_k} multiply on all the symmetric dimensions and produces a symmetric function. Then the product of $\Delta\bar{b}_k$ taken n times produces the n-th order inner-product. Thus, we can sum the set of tensors at each stage to generate a single tensor representation. For example in the quadratic case:

$$\begin{aligned}
\sum_{k=0}^{K} \nabla^2_{\bar{b}_k} c_{u_k} \cdot \left(\phi_{I_k} \Delta\bar{b}_0\right)^2 &= \sum_{k=0}^{K} \left(\Delta\bar{b}_0\right)^T \phi_{I_k}^T \nabla^2_{\bar{b}_k} c_{u_k} \phi_{I_k} \Delta\bar{b}_0 \\
&= \sum_{k=0}^{K} \left(\Delta\bar{b}_0\right)^T \left[\sum_{k=0}^{K} \phi_{I_k}^T \nabla^2_{\bar{b}_k} c_{u_k} \phi_{I_k}\right] \Delta\bar{b}_0 \\
&= \left[\sum_{k=0}^{K} \nabla^2_{\bar{b}_k} c_{u_k} \cdot \left(\phi_{I_k}\right)^2\right] \cdot \left(\Delta\bar{b}_0\right)^2.
\end{aligned}$$

The second equation follows from sum of the inner-product of symmetric tensors. The third equation follows from the definition of the tensor product. The end result is that the coefficients of each monomial terms, as represented as entries of the ranks 1 to n tensors, can be added together while retaining a nth order approximation. Thus we arrive at the stated result □

The simulated running cost along the unperturbed trajectory is $v_{I_K,sim}$ and $\nabla^n v_{I_K}$ represents the n-th order derivative of the cost along I_K with respect to b_0. The approximation error is captured by the residual term $v_{I_K,err}$. Each of the terms of the first equation a function of Δb_0 only. We exploit the fact that the sum of polynomials is equivalent to the sum of the coefficients of the monomial terms, e.g. $\sum_i (a_i x^2 + b_i x + c_i) = (\sum_i a_i)x^2 + (\sum_i b_i)x + (\sum_i c_i)$. This same property holds for tensors. This collapses the representation to be independent of the the stage. The result is a function that maps perturbations to the initial belief Δb_0 to a perturbation on the running cost.

The the residual (error) $v_{I,err}$ is difficult to quantify but Taylor series provides loose bounds. The approximation error generally decreases as n increases.[9] An increase in the

[9]However, when the small divisor problem occurs, high order terms can dominate, which leads to chaotic

order of the approximation, however, has a significant impact on the time complexity. We must, therefore, consider the trade-off between the error of the analytical approximation and the cost of re-simulation. It is possible for the computation requirements to grow exponentially with the order of the approximation.

We note, however, that the actual time complexity is problem dependent and so the order of the approximation should be considered at design time. For example, the overall complexity is limited by the complexity of the n-th order derivative of the cost at each stage. Thus, if the n-th order derivative is sparse–few of the coefficients of the monomial are nonzero–then a higher order approximation may be utilized without incurring a significant increase in time complexity. Furthermore, there is little additional information added as we increase the order of the derivative–all that occurs is that the tensor is copied into a higher dimensional space (refer to CorollaryA1 in Section 6.7.1). Intuitively this makes sense as the nonlinearity is the result of a scalar value in the denominator. The implication is that there may exist a more efficient representation of high dimensional tensor terms.

With the cost along a sample path $v_{I_k^{(i)}}$ determined, all that remains is taking the expectation over the set of the sample paths to obtain V. Substituting $b_0^{'} = b_0 + \Delta b_0$ into (6.62), we obtain

$$\begin{aligned} V(b_0^{'}) &= \frac{1}{\mu} \sum_{I_K^{(i)} \in \tilde{\mathcal{I}}_K} v_{I_K^{(i)}}(b_0^{'}) \frac{p(I_K^{(i)}|b_0^{'})}{p(I_K^{(i)}|b_0)} \\ &= \sum_{I_K^{(i)} \in \tilde{\mathcal{I}}_K} v_{I_K^{(i)}}(b_0^{'}) \frac{(\eta^{(i)})^T b_0^{'}}{\eta^T b_0^{'}}, \end{aligned} \tag{6.66}$$

where $(\eta^{(i)})^T = \frac{\mathbf{1}^T \phi_{I_K^{(i)}} b_0^{'}}{\mathbf{1}^T \phi_{I_K^{(i)}} b_0}$, $\eta^T = \sum_{I_K^{(i)}} (\eta^{(i)})^T$, and $\mu = \eta^T b_0^{'}$. We can approximate the importance weight and normalization factor using a n-th order Taylor series approximation of (6.66) via the generalized chain rule, where we take the derivative of $b_0^{'}$ around b_0. This keeps

behavior.

the order of the polynomial approximation fixed. Again, we can collect monomial terms to obtain a compact formulation. An analogous derivation is performed in Section 6.3.2 to obtain a linear approximation. The result is a n-th order polynomial approximation:

$$V(b_0 + \Delta b_0) = V_{sim} + \sum_{n=1}^{n} \frac{1}{n!} \nabla_{b_0}^n v_{I_K} \cdot (\Delta b_0)^n + V_{err}(\Delta b_0),$$

where $b_0^{'} = b_0 + \Delta b_0$.

This compact representation comes at the trade off of exactness. To achieve this we approximate the cost around the generated sample path. More critically, we assume that the same set of actions and observations occur for a perturbed belief. While locally this is likely the case, the question remains to the size of the neighborhood around the sample for which this holds. This issue is somewhat mitigated by the fact that stochastic systems are generally insensitive and perturbations are attenuated over time.

The process of chaining multiple forecasted evolutions together is discussed in Section 6.4. The derivation in Section 6.4 obtains a linear approximation of the chaining process. The main difference with the general non-linear formulation derived in this section is that a higher order approximation (beyond linear) may be derived. This process is analogous to the methodology reproduced throughout this paper, and, in particular, with the derivation of this section. First, the generalized product rule is used to generate a the first through n-th order derivative of the running cost function $V(b_0 + \Delta b_0)$, which is expressed as the already derived Taylor series approximation, i.e. $V_{sim} + \sum_{n=1}^{n} \frac{1}{n!} \nabla_{b_0}^n v_{I_K} \cdot (\Delta b_0)^n + V_{err}(\Delta b_0)$, and the weighting function $\frac{(\eta^{(i)})^T b'}{\eta^T b'}$. Gathering coefficients of the monomial terms, we again obtain a n-th degree Taylor series approximation. In this way, the complexity of the chaining process is irrespective with the number of forecasted evolutions that have been chained together.

Chapter 7

ADAPTIVE SAMPLING-BASED OPTIMIZATION FOR POMDPS

In this chapter, we propose the Adaptive Exploration/Exploitation Sampling-based Optimization for POMDP (AESOP) algorithm. The proposed method, AESOP will be developed in Section 7.2. Examples are provided in Section 7.3. We conclude with some future directions and final remarks Section 7.4.

7.1 Introduction

When described in terms of a system's state space, the evolution of a POMDP is governed by a set of transition probabilities that describe the effects of control actions, and an observation model that specifies uncertainty in the sensing process. If, instead, the system is described in terms of the belief space (i.e., the space of possible a posteriori probability functions on the state space), the evolution of the system can be modeled as a Markov decision process (MDP). This corresponds to lifting the system description from a lower dimensional state space to a higher dimensional belief space.

In general we cannot reach arbitrary points in this space, regardless of the quality of control law. In order to develop a sampling-based planner, we explicitly analyze the sensitivity of both cost and hyperbelief evolution functions with respect to perturbations in the belief. This allows us to adapt a simulated cost and trajectory starting from one hyperbelief state to a cost and trajectory starting at a nearby hyperbelief, without re-simulation of the system.

AESOP applies temporal and spatial abstraction to locally approximate large POMDP

systems. Through a mixed simulation/analytic representation, a directed graph is generated to determine the underlying structure of the POMDP via an anytime algorithm, which enables the refinement of the best optimal policy at each iteration of the technique. The vertices of the directed graph represent beliefs and are generated by sampling. The edges are generated by simulation of the system for multiple stages, e.g. time steps, using simple greedy policies defined on the belief space. They represent transitions from the belief represented by the source vertex to a region near the belief represented by the target vertex. Utilizing our characterization of the effect of perturbations (sensitivity), we can translate a walk through the graph to a connected path through the belief space. These paths correspond to feedback policies for the system. Retaining the sensitivity along edges of the graph also allows reuse of the data structure without re-simulation of the system. These spatial and temporal abstractions are key when planning for systems that not only have increasingly large state, action, and observation spaces, but also require long planning horizons to find sufficiently good policies.

AESOP uses inductive bias to sample temporally abstracted policies to effectively explore the belief space based on previous experience. To leverage information garnered thus far, an exploitation sampling function is alternately applied that too uses inductive bias to predict which policies will perform the best to incrementally improve the current value. After a policy is added to the graph, an iterative optimization algorithm is then used to update the value function over the graph by selectively updating only the beliefs predicted to improve based on the newly added policy. We believe they give our algorithm advantages in terms of both scalability and practicality.

7.2 Methodology: Adaptive Sampling

Given POMDP system, our objective is to determine a policy π that minimizes the expected total cost $V(b_0)$ starting from the initial belief b_0, such that

$$V^*(b_0) \quad = \quad \min_{\pi \in \Pi_{fb}} E\left[\sum_{k=0}^{K-1} c(b_k, \pi(b_k)) + c_K(b_K) \,|\, b_0\right],$$

where policy π in the class of all feedback policies Π_{fb}.[1] The cost function $c : \mathcal{P}_b \times \mathcal{U} \to \mathbb{R}$ may be belief dependent, enabling the evaluation risk-based cost functions. Risk-based cost functions encapsulate costs that are dependent on the uncertainty present in the system, which is ideal for localizing a robot at a goal position.

Our approach seeks to produce a tractable approximation by

- sampling a meaningful and representative portion of the belief space;
- approximating the predicted evolution of the POMDP system though hyper-particle filtering; and
- effectively connecting portions of the belief space to reduce optimization computation cost.

Many methods, as cited in Chapter 3, find approximate solutions by reducing the complexity or dimensionality of the belief space. Our approach is not contrary to such approaches. In fact, it may be possible to combine approaches to achieve even greater results.

To determine the optimal, or nearly optimal, policy, our method generates a hierarchical representation by constructing a directed graph (digraph), $G =< N, E >$. The vertices, N, of the digraph correspond to a sampled belief. The forecasted evolution of the system initiating from one of the vertices in the graph under a given policy is represented by a set of edges. Each policy is represented as a multi-edge set $\pi^{i \to j}$, where each edge, E, corresponds to a multi-stage path terminating at one of the vertices (belief samples) in the digraph

[1] This formulation is for finite/indefinite-time horizon Bolza cost functions. However, we also consider discounted infinite horizon models where the total cost under π is $V(\beta_0) = E\left[\sum_{k=0}^{\infty} \gamma^k c(b_k, \pi(b_k)) \,|\, \beta_0\right]$, and $0 < \gamma < 1$ is a discount factor. The derivation provided throughout this paper is applicable for such cost functions and examples of infinite horizon problems are provided in Section 7.3.

(edges carry a label to indicate the policy that it generated from). Accordingly, each edge has associated with it the probability that policy reaches each such terminal belief. The set of edges from a vertex represent the forecasted evolution of the system from the initial belief under the given policy. Numerous aspects of this representation are shared with point-based POMDP methods discussed in Chapter 3. The key difference is that edges are temporal abstraction of policies over a plurality of stages–not just a single action for a single stage.

At each iteration of our algorithm, we expand the digraph. Graph expansion proceeds by generating a new set of vertices and edges to add to the graph. During the expansion phase, the running cost and perturbation analysis along each of the new edges is determined. After expansion, an iterative graph optimization algorithm is applied to update the current optimal cost-to-go for each vertex in the graph. The process then repeats–starting with the expansion phase again. The proposed method is an anytime algorithm, which can be terminated at any point to provide the best policy determined thus far. The result is a structural representation of the system's behavior in conjunction with nearly optimal policy with respect to the stated objective function. The pseudo code describing these steps is provided in Algorithm 5, where the newly added policy edges are denoted by E'. The set of new edges to the graph E' inform the optimization routine of which beliefs to initiate the backup procedure with. The functions *ExpandGraph* and *OptimizeGraph* are described in detail in the following sections.

When we derive the optimal solution over the graph, we will have to take into consideration the fact that each source belief sample likely will be unable to reach each target hyperbelief sample exactly. To address this issue, we perform perturbation analysis (see Chapter 6 for an in-depth description), which generates a sensitivity function representing a local approximation of the effect of a perturbation has on both the cost and the evolution along each edge. This eliminates the need to re-simulate the system when evaluating the approximate path and cost for perturbed beliefs as well as enabling spatial abstraction by providing a representation of the performance in the region around each of the sampled

hyperbeliefs.

The formulation we provide also enables the chaining of perturbation functions along edges to join the forecasted evolution of the system along multiple edges. In the same way that the sensitivity function eliminates the need to simulate paths stage-to-stage along edges, the composite of multiple edges eliminates the need to simulate the system from edge-to-edge along a composite policy. Representing the sensitivity of the optimal cost-to-go policy from each vertex greatly reduces computational requirements to update the optimal policy each iteration when a new vertex and set of edges are added to the graph. The process will be described in more detail in the following sections.

Algorithm 5: Anytime Graph Optimization

Input: b_0 : initial belief
Output: cost: the optimal cost for each vertex in the graph
$G =< N, E >$: digraph
Γ : mapping of optimal local policy edges to visit for each vertex in G

Insert b_0 into graph G ;
Initialize FIFO queue edge_opt to be empty;
while $s = 0$ *to anytime* **do**
  E', $G \leftarrow$ ExpandGraph(G) ;
  Push each policy edge set in E' into edge_opt ;
  cost, $\Gamma \leftarrow$ OptimizeGraph(G, edge_opt) ;
return *cost,* Γ, G ;

7.2.1 Digraph expansion

As our technique iterates, the digraph G is expanded adding edges and vertices at each iteration. Each vertex $i \in N$ corresponds to a belief $b^{(i)}$. [2] For notational convenience we will label the vertices with the same label as the belief samples. Each forecasted evolution for policy π_s starting from $b^{(i)}$ is captured by a set of edges $\{\pi_s^{i \to j}\}_j$, which represent the evolution to vertex j. We begin with a digraph containing a single vertex 0 representing the

[2] Throughout this paper, we use subscript whenever possible to denote indices. The subscript for beliefs is reserved as a time index, i.e. b_k represents a belief at stage k.

initial belief. An illustration of the process of expanding the digraph is depicted by Figure 7.1, where a total of 5 forecasted evolutions.

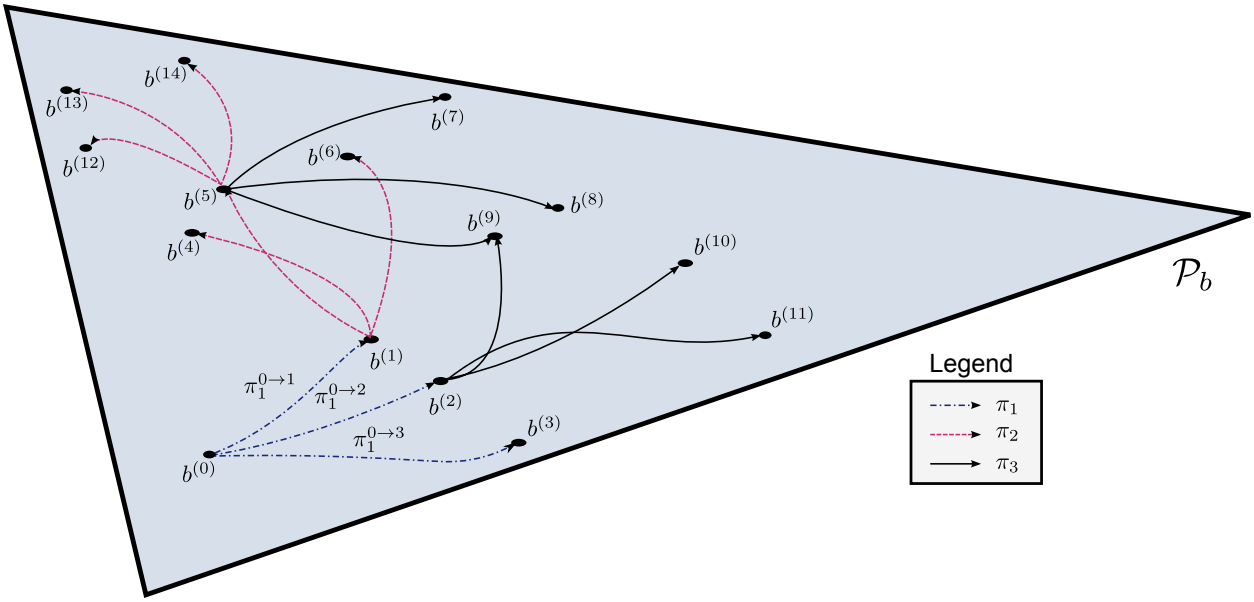


Figure 7.1: An illustration of the digraph G, which includes multi-edge grouping per r and corresponding vertices in digraph G

There are a total of three different policies (π_1,π_2, and π_3) used to generate the forecasted evolutions. Each of the forecasted evolutions are represented by a set of edges, e.g., π_1 from $b^{(0)}$ comprises three edges, which terminate at $b^{(1)}$, $b^{(2)}$, and $b^{(3)}$. Multiple edges may terminate at the same vertex as illustrated by vertex $b^{(8)}$, which is reached from both vertices $b^{(3)}$ and $b^{(6)}$.

In this hierarchical approach, we restrict the lower level to be a local policy within the set of feedback policies Π_{fb}. The class of policies chosen for analysis in Section 7.3 are belief target policies where a gradient is provided that directs the policy towards a specified goal belief. However alternate policy functions may be used. For instance, Candido et al. in [48] suggest local policies can be provided by the system designer inform the optimization algorithm to hasten convergence. The upper level policy $\gamma(\cdot)$ selects the local policy based which edge in G is being traversed and on the current stage to produce a closed-loop policy. The complete policy is a switching-based controller (see [103] for an overview

of hybrid/switching-based methods), whereby the local policy being executed is determined based on some termination condition. This condition is variable, for instance it may depend on policy executing for a maximum number of stages or policy no longer being able to make progress via some value function. The set of all possible switching policies is denoted as Γ. We therefore can represent the value function for a given system as

$$V^{(i)*}(b_0) \approx \min_{\gamma \in \Gamma} E\left[\sum_{k=0}^{K-1} c(b_k, \pi_\gamma) + c_K(b_K) \,|\, b_0\right].$$

We denote the value of the digraph at each vertex as $V^{(i)}(\cdot)$, where i is the vertex so that the optimal value at the initial belief is given by $V^{(0)}(b^{(0)})$. The set of policies that have already been evaluated for a vertex $b^{(i)}$ is denoted by the set $\Pi^{(i)} \subseteq \Pi_{fb}$. Each vertex also has the the cost-to-go function for each policy $\pi \in \Pi^{(i)}$, denoted as $V_\pi^{(i)}(b)$, which holds for any belief b. The value function is just defined as $V^{(i)*}(b^{(i)}) = V_{\pi_i^*}{}^{(i)}(b^{(i)})$, where $\pi_i^* = arg\min_{\pi \in \Pi^{(i)}} V_\pi^{(i)}(b^{(i)})$.

Expansion occurs by either exploration-based sampling or by exploitation-based sampling. During an exploration phase, the system to expands into the unexplored space by attempting to anticipate which policies will make the most progress. Otherwise, during an exploitation expansion, a vertex policy pair is selected based on the predicted impact on the optimal value of the system. The policy is simulated and the policy edges are added to the digraph. The sampling function is then updated based on the results of the simulation. The sampling method is selected randomly from a binomial distribution that is static learning rate q, so that exploration is chosen randomly, on average, q percent of the time, with exploitation selected the remaining $1 - q$ percent of the time. Adaptively selecting when to explore and when to exploit remains an area of future research.

For efficiency, we retain a set of the explored belief set B_t^{exp} , which comprises all the beliefs in the graph at iteration t. The explored belief set is stored in a K-d tree [104], where the high dimensional beliefs are projected into much lower dimensional space by approximating

the belief by a small number of moments. This ensures quick distance comparisons can be performed without having to perform brute force comparison with all possible beliefs. Each vertex also retains a list of the n-nearest neighbors $B_{near}^{(i)}$. This list is iteratively updated as the graph is expanded to keep the computational burden minimized. This neighbor set $B_{near}^{(i)}$ also contains all values needed to perform the sampling in the following sections. The iterative update of the neighbor set ensures that it is unnecessary to regenerate the sampling distribution for every belief at each iteration.

7.2.2 Exploration sampling

The purpose of exploration-based sampling is 1) to expand the into the reachable portion of the belief space and 2) to build a sparse representation of the POMDP system's structure. This proceeds in two steps. First we sample a vertex belief to expand. Next, we select which of the policies to simulate starting at the sampled vertex. Policy sampling seeks to eliminate local policies that are similar to one another and to preferentially select policies for which accumulated evidence in the local region around the reference belief suggest are likely to provide satisfactory exploration. This inductive bias is generated by seeking the behavior of the policies of the n-nearest neighbors. Policy sampling is biased by the likelihood of the policy to make gains, weighted by the inverse distance from the reference belief sample.

Effective means to preferentially sampling beliefs include adaptive sampling. A naive approach to adaptively sample would be to just to take a frequency distribution over the neighboring beliefs of each vertex. Then one could just sample policies that are used less frequently. This, however, is no more than averaging the policy sampling over the belief space. A simple frequency distribution of the sampled policies the neighborhood of a belief is likely not capable as more nuanced sampling methods. In particular, we seek to incorporate the effectiveness of the the policies when simulated: both a policy's success and failure should be weighed when sampling.

We define the chance of a policy succeeding if sampled with the binomial random variable $\mathbf{S}_{exp}$, where S_{exp} indicates success and $\neg S_{exp}$ represents failure. If a policy π has not been simulated at $b^{(i)}$ we extrapolate the likelihood a policy succeeding from neighborhood around $b^{(i)}$. The simple indicator variable $\mathbf{S}_{sim}$ denotes if a policy has been simulated (S_{sim} implies it has and $\neg S_{sim}$ indicates that it has not been simulated) :

$$p(S_{sim}|b^{(i)}, \pi_m) = \begin{cases} 1 & \text{if simulated at } b^{(i)} \\ 0 & else \end{cases}.$$

Initially, $p(S_{exp}|b^{(i)}, \pi, \neg S_{sim}) = \frac{1}{\epsilon_{exp}}$ for all $b^{(i)}$ and π. This value, ϵ_{exp}, can vary between zero and one. If near zero, the sampling method will avoid sampling policies that have not been simulated previously. Conversely, if initialized to near one, the sampling method will prefer to sample unexplored policies. After simulating policy π, the probability of π successfully exploring starting at $b^{(i)}$, updated to become

$$p(S_{exp}|b^{(i)}, \pi, S_{sim}) = \frac{d_{\beta_K^{(i)}}}{d_{max}}.$$

The ratio $\frac{d_{\beta_K^{(i)}}}{d_{max}}$ is a measure of how much progress policy π made in exploring a new portion of the space. The denominator d_{max} is the theoretical maximum distance between beliefs.

We obtain $d_{\beta_K^{(i)}}$ after simulating the policy π from $b^{(i)}$. It is the expected distance from the explored space is generated for its terminal hyperbelief of the forecasted evolution.

The expected distance is computed by weighting the probability of each belief $b \in \beta_K^{(i)}$ in the terminal hyperbelief by the distance to the closest belief b^{exp} in the explored belief set B_t^{exp}, so that

$$d_{\beta_K}^{(i)} = \sum_{b \in \beta_K^{(i)}} \min_{b^{exp} \in B_t^{exp}} d(b||b^{exp}) p(b),$$

where $d(\cdot||\cdot)$ is a belief distance function that measures the similarity between probability

functions.[3]

Our objective is to use neighboring information to update $p(S_{exp}|b^{(i)}, \pi_m, \neg S_{sim})$ based on the actual performance of the POMDP system. In particular, we are interested in is the probability that a policy will be successful in exploring the space given it has not been simulated yet. We capture this concept in the policy sampling probability function:

$$p(S_{exp}|b^{(i)}, \pi_m, \neg S_{sim}) = \frac{1}{\mu} \sum_{b \in B^{neigh}(i)} \sum_{S_{sim}} \frac{p(S_{exp}|b, \pi_m, S_{sim})}{d(b^{(i)}||b)}, \tag{7.1}$$

where

$$\mu = \sum_{b \in B^{neigh}(i)} \sum_{S_{sim}} \frac{p(S_{exp}|b, \pi_s, S_{sim})}{d(b^{(i)}||b)}$$

is a normalizing factor.

The formulation of the policy sampling function above in (7.1) leverages local policy information to extrapolate which policies at belief $b^{(i)}$ are most likely to achieve greatest exploration. However, before we sample a policy, we need to sample the belief $b^{(i)}$ that will be expanded. After a belief sample is generated, we sample a policy from (7.1) conditioned on $b^{(i)}$. We derived the policy sampling probability function first because we will use the marginalized probability function over all policies as a weighting factor when sampling beliefs.

When sampling beliefs to expand the graph, there are three major factors we weigh:

- The likelihood of the remaining policies starting from $b^{(i)}$ expanding into an explored region of the belief space: $P(S_{exm}|b^{(i)}, \neg S_{sim})$;
- The reachability of the the belief: $r(b^{(i)})$; and

[3]There are various measures than can be employed to determine the distance between beliefs. For instance, if there is an implied measure on the state space, then metrics such as Mallows distance (or earth mover's distance) [77] and the probability metric [105] may be applied. Alternatively, when no state space measure is applicable, probability distribution measures such as such as Jenson-Shannon divergence, Hellinger distance, or the 1-norm are often used (refer to [76] for a catalog of probability distance functions).

- The discounted factor α raised by the minimum number of stages, K, it takes to get from $b^{(i)}$ from the initial belief $b^{(0)}$: α^K, where $0 < \alpha \leq 1$.

The first term is precisely the marginal of the policy sampling probability function:

$$P(S_{exp}|b^{(i)}, \neg S_{sim}) = \sum_{\pi} P(S_{exm}|b^{(i)}, \pi, \neg S_{sim}) P(S_{\pi}).$$

We assume a uniform prior over the policies $S_{\pi} \sim Uniform(\Pi)$. This is essentially equivalent to assuming that there is no a priori knowledge about which policies will perform better throughout the belief space. Reachability is determined by the maximum of the probability of a belief $b^{(j)}$ occurring starting from $b^{(i)}$, or $r(b^{(i)}) = \max_{b^{(J)} \in N} p(b^{(i)}|b^{(l)})$. Each time a vertex is reached, the reachability is updated–this includes when the vertex is first added to the graph. The probability of a belief being reached is the weighted sample path probability $p(b^{(i)}|b^{(l)}) \propto P(I_{b^{(l)}\pi}|b^{(l)})$, which we derive in Section 6.3.1.

These three factors are combined to generate belief sampling probability function:

$$p(S_{exp,b}) = \xi_{exp} P(S_{exp}|b^{(i)}, \neg S_{sim}) + \xi_r r(b^{(i)}) + \xi_{\gamma} \gamma^K.$$

The above is a convex combination of the three independent factors, where $\xi_{exp} + \xi_r + \xi_{\alpha} = 1$. At each iteration, both the policy sampling probability function and the belief sampling probability function are updated. Then a belief $b^{(i)}$ is sampled from $p(S_{exp,b})$ and for that belief a policy π is sampled from $p(S_{exp}|b^{(i)}, \pi_m, \neg S_{sim}) p(S_{\pi})$, which is weighted by the prior over the policies. Evaluation of the candidate local policy set is simulated using hyper-particle filtering (refer to Chapter 4), which continues until either the policy is unable to make any more progress or when a maximum number of stages has elapsed. We assume for the duration of this paper that each local policy selects actions by choosing the best next stage action relative to a policy specific value function. For instance, the value function may specify a gradient directing the system to a specific belief. Once the policy π terminates,

the resulting forecasted evolution is added to the graph as a new set of vertices and edges. Pseudo code for the exploration sampling procedure is outlined in Algorithm 6. The process of simulating a policy and then adding the resulting forecasted evolution to the graph, as encapsulated by update_graph, is described below in Section 7.2.3

After the new vertices and edges are added to the graph, the nearest neighbors and all associated weights are updated based on the newly added vertices to the graph. It is interesting to note that we use a fixed number of n-nearest neighbors around each vertex. Because of this, when the process starts out, the neighborhood approximation starts as coarse representation. But as the number of beliefs increase the size of the neighborhood shrinks, a finer grained analysis is obtained. This automatic tuning of the resolution of a adaptive sampling has the benefit of allowing greater variance in the policy sampling early on, which will converge to essentially the the infinitesimal neighborhood around a sample belief as the number of samples goes to infinity.

7.2.2.1 Exploitation sampling

When exploitation-based sampling is selected, the objective is to increase the connectivity in an area predicted to decrease the cost of the system. This is necessary as our optimization algorithm does not perform a full backup; only a subset of beliefs are updated at each iteration. Increasing the connectivity ensures that low cost vertices are maximally leveraged. Exploitation sampling serves the purpose of biasing policy sampling to select policies that have a high potential of decreasing the cost of the system. As with exploration sampling, spatial cues are used to both select a vertex $b^{(i)}$ to be expanded as well as the policy to be simulated. The selection process is similar to the method presented above in Section 7.2.2. First we derive a distribution over the policies that predicts the potential of a policy to decrease the value at a given vertex.

To derive a good predictor for impact of a policy, we want a measure of the region's value to the theoretical lower value bound. This gap defines the value range that the policy

Algorithm 6: ExploreSample

Input: $G =< N, E >$: current digraph
Output: E_{set} : edge policy set

foreach *vertex* $i \in N$ **do**
  foreach $\pi \in \Pi$ **do**
    foreach *neighbor* $j \in B^{(i)}_{near}$ **do**
      if $\pi \in \Pi^{(j)}$ **then**
        $p(S_{exp}|b^{(i)}, \pi, \neg S_{sim}) \leftarrow \frac{p(S_{exp}|b^{(j)}, \pi, S_{sim})}{d(b^{(i)}||b^{(j)})}$;
      else
        $p(S_{exp}|b^{(i)}, \pi, \neg S_{sim}) \leftarrow \frac{p(S_{exp}|b^{(i)}, \pi, \neg S_{sim})}{d(b^{(i)}||b^{(j)})}$;
  $\mu \leftarrow \sum_{\pi} p(S_{exp}|b^{(i)}, \pi, \neg S_{sim})$;
  Normalize each $p(S_{exp}|b^{(i)}, \pi, \neg S_{sim})$ by μ ;
$p(S_{exp}|b^{(i)}, \neg S_{sim}) \leftarrow \sum_{\pi} P(S_{exm}|b^{(i)}, \pi, \neg S_{sim})P(S_{\pi})$
$p(S_b) \leftarrow \xi_{exp} p(S_{exp}|b^{(i)}, \neg S_{sim}) + \xi_r r^{(i)} + \xi_{\gamma}\gamma^K$;
$\mu \leftarrow \sum p(S_{exp,b})$;
Normalize each $p(S_{exp,b})$ by μ ;
sample $b^{(l)}$ from $p(S_{exp,b})$;
sample π from $p(S_{exp}, b^{(l)}, \neg S_{sim})$;
$\beta_K \leftarrow$ simulate π from $b^{(l)}$ using hyper-particle filtering ;
$E_{set} \leftarrow$ update_graph(β_K,π) ;
return E_{set}

potentially can reduce. The larger the gap, the greater the possibility to decrease the value. We obtain the theoretical lower bound from the MDP solution to the POMDP. Because of additional uncertainty and partial observability, POMDPs are lower bounded by the MDP value function. This observation is a key component in many contemporary POMDP approximation techniques including [7, 8, 11]. Each of these methods use the value gap to bias their sampling. Kurniawati et al. in [11], in particular, use the gap to selectively perform depth-based sampling. Our approach does not hard code any bias but instead seeks to discover it through the adapting optimization process. Gap-based sampling preferentially selects policies that make incremental progress towards the lower bound optimal value over incremental progress over the nominal value.

We define the value gap as the difference between $V^{(l)}_{\pi}$, the lowest value function in the

neighborhood around $b^{(l)}$ using π evaluated at $b^{(l)}$, and $V^{(l)}_{MDP}$, the expected MDP solution at $b^{(i)}$:

$$\Delta V^{(i)}_{\pi} = V^{(i)}_{\pi} - V^{(i)}_{MDP}.$$

The expected MDP value is

$$V^{(i)}_{MDP} = V_{MDP}(x) b^{(i)}(x),$$

where V_{MDP} is obtained by solving the equivalent MDP version of the POMDP where the system is fully observable. Efficient methods exist to find exact or approximate solutions to MDPs. Refer to [3] for an overview of such approaches. Even the naive exact optimization has a time complexity $O(KX^2)$, where K is the time horizon and X is the number of states.

The approximate value for $b^{(i)}$ relative to the value at $b^{(j)}$ for policy π is bounded above by minimum the value over all the neighbors:

$$V^{(i)}_{\pi} \lessapprox \min_{b^{(j)} \in B^{(i)}_{near}} V^{(j)}_{\pi_m}(b^{(i)}).$$

This follows directly from the fact that $V^{(i)}_{\pi}(b^{(i)}) \lessapprox V^{(j)}_{\pi}(b^{(i)})$ for all $b^{(J)}$ because the optimal value function is piecewise linear and convex. The linearity assumption of the value function holds for the exact single stage backup, but our perturbation analysis will suffer increased error the further a neighbor $b^{(j)}$ is from the reference belief $b^{(i)}$. Furthermore, we want a direct way to weight when the exploitation sampling procedure has an unsatisfactory outcome. For this reason we take the average value for $b^{(i)}$ under π between neighbors weighted by the inverse distance to each neighbor. This is analogous to our derivation above for the exploration bias. We, therefore, define the lowest value function as

$$V^{(i)}_{\pi} = \sum_{b^{(j)} \in B^{(i)}_{near}} \frac{V^{(j)}_{\pi}(b^{(i)})}{d(b^{(i)} || d^{(j)})}.$$

To create the policy sampling probability function $\mathbf{S}_{MDP}$ (where S_{MDP} and $\neg S_{MDP}$ indicate

success and failure, respectively), we just normalize the result over all policies:

$$p(S_{val}|b^{(i)}, \pi, S_{sim}) = \frac{\Delta V_{\pi}^{(i)}}{\mu},$$

where $\mu = \sum_{\pi \in \Pi} \Delta V_{\pi}^{(i)}$. Unlike the exploration sampling, exploitation sampling only considers policies that have executed. Therefore, $p(S_{MDP}|b^{(i)}, \pi, \neg S_{sim}) = 0$ for all $b^{(i)} \in N$ and $\pi \in \Pi$.

Again, as was the case with exploration sampling, the result is a probability function over policies for each belief, which cannot be used to sample beliefs directly. To obtain a belief sampling probability function, we marginalize the policy value over the set of policies to obtain a prediction of the likelihood a given belief will produce tangible gains in the value function. The marginalized probability function is

$$p(S_{val}|b^{(i)}, S_{sim}) = \sum_{\pi \in \Pi} p(S_{val}|b^{(i)}, \pi, S_{sim}) p(S_{\pi}).$$

The prior over the set of policies S_{π} is the same as what was used in Section 7.2.2, which assumes a uniform distribution over the set of policies.

Beyond the adaptive exploitation sampling technique, we also want to further bias the sampling of beliefs based on the potential impact the belief will have on the value function. For instance, we know that if the reachability is low, i.e. the belief has a low probability of occurring, then it will likely not contribute as much as a higher probability neighboring belief. Similarly, if there is a high discount factor, then there is little impetus to sample long time horizons, as the discounting will quickly attenuate the value function the greater the number of stages. To address these concerns, we combine the exploitation sampling method with both additional terms used for the exploration sampling: a reachability bias $r(b^{(i)})$ and discount bias α^K. These three factors are combined to generate belief sampling probability

function:

$$p(S_{val,b}) = \xi_{val} P(S_{val}|b^{(i)}, \neg S_{sim}) + \xi_r r(b^{(i)}) + \xi_\gamma \gamma^K,$$

where $\xi_{val} + \xi_r + \xi_\gamma = 1$. The weighting functions *i.e. ξ_{val}, ξ_r, and ξ_γ) are just heuristic values used to ensure that a balance between sampling methods is obtained. The algorithm for performing exploitation sampling is substantially similar to the method described in Algorithm 6, with the only difference being the weighting of S_{val} instead of S_{exp}.

In addition to exploitation sampling probability function, every new vertex added to the graph, a greedy cost policy, which selects the one stage look-ahead action minimizing the cost function $c(\cdot)$ at each stage, is also executed. This local exploitation policy provides local value information for each new vertex in the graph, which is initially a terminal leaf vertex until it is expanded. This method is also useful for systems that are nearly fully observable as the solution to the underlying MDP can be used as the greedy value function. We will demonstrate later in Section 7.3 that this representation can speed up convergence for nearly fully observed system. For practical reasons we determine the local diversity value of each vertex at each stage of the optimization algorithm. Because the optimization algorithm searches the n-nearest neighbors during the backup of a vertex, we can easily adjust the local diversity value during this operation.

7.2.3 Generating edge and vertex information

Once a policy π is simulated starting from $b^{(i)}$ the complete hyperbelief is split into a set of sample paths: $\{I^{(s)}_{b^{(i)},\pi}\}_s = \tilde{\mathcal{I}}_{b^{(i)},\pi}$. Each sample path $I^{(s)}_{b^{(i)},\pi}$ corresponds to an information vector starting from $b^{(i)}$. The terminal belief $b^{(s)}_K$of each sample path $I^{(s)}_{b^{(i)},\pi}$ is added as a new vertex $b^{(j)}$ to the graph unless a terminal belief is near an existing vertex. In which case, no new vertex is added. An edge $\pi^{i\to j}$ is created for each sample path with the source $b^{(i)}$ and target $b^{(j)}$. When a terminal belief is near an existing vertex $b^{(l)}$, a new edge is added from $b^{(i)}$ to $b^{(l)}$. In this way, we are able to capture the forecasted evolution under policy π

starting from $b^{(i)}$.

The set of sample paths simulated, and hence the set of edges added to the graph, is a subset of the complete set of sample paths: $\tilde{\mathcal{I}}_{\pi,b^{(i)}} \subset \mathcal{I}_{\pi,b^{(i)}}$. The set of edges then represent the likely sample paths the system follows under the policy π, which we use to estimate the sensitivity of the evolution of the system under the given policy due to perturbations in the initial starting belief. Each edge $\pi^{i\to j}$ has associated to it a transition sensitivity function, which is is a chained representation of the effect of a perturbation in the initial belief $\Delta b_0^{(i)}$ on the target belief of an edge $\Delta b^{(j)}$. From (6.7) in Section 6.3.1, the perturbation along an edge is be approximated by

$$b_0^{(j)} + \Delta b_0^{(j)} \approx b_0^{(i)} + \frac{\phi_{I_K}}{\mathbf{1}^T \phi_{I_K} b_0^{(i)}} \left(\mathbf{I} - \frac{b_k^{(i)} \mathbf{1}^T \phi_{I_K}}{\mathbf{1}^T \phi_{I_K} b_0^{(i)}} \right) \Delta b_0^{(i)},$$

where $\Delta b^{(i)}$ is a perturbation to the to $b^{(i)}$.

Each vertex has associated with it the running cost sensitivity function for each policy and the optimal cost-to-go sensitivity function. The running cost sensitivity function represents the change in value along the edge due to a perturbation $\Delta b^{(i)}$ from the initial belief $b^{(i)}$. Using a linear representation via a first order Taylor series, the running cost sensitivity for a policy is approximated as (6.26) from Section 6.3.2:

$$V_\pi^{(i)}(b^{(i)} + \Delta b^{(i)}) \approx V_{\pi,sim}^{(i)} + \alpha_{b^{(i)},\pi}^T \Delta b^{(i)}.$$

Each vertex $b^{(i)}$ of the graph comprises the the set of edges for each policy π in the set of simulated policies $\Pi^{(i)}$ from vertex $b^{(i)}$. The optimal policy is identified as $\pi^{(i)*}$ and the optimal chained cost sensitivity function is retained as $V^{(i)*}$. Initially each vertex simulates a greedy policy and assigns the initial optimal path and cost functions the resulting cost sensitivity and path sensitivity. As the optimal policy is updated, the vertex information is updated so that each vertex contains the optimal policy, cost, and chained value functions

found so far.

The advantages of this sensitivity-based formulation are two fold. First, we achieve temporal abstraction. The chained representation of policies allows us to avoid re-simulation of multi-stage policies as well as use previously simulated policies to predict performance in one step during exploitation-based sampling. Secondly, we achieve spatial abstraction by extrapolating performance of the local policy in the neighborhood around $b^{(i)}$. However, these benefits are not free: additional space complexity is required for each edge and vertex to store the sensitivity cost function as well as the transition sensitivity functions. Both the running cost sensitivity function and the belief transition sensitivity function are approximations of the exact effect a perturbation $\Delta b^{(i)}$ has on the running cost of the policy and the terminal belief, respectively. Refer to Chapter 6 for additional information this perturbation analysis technique.

7.2.4 Updating the approximately optimal policy

After expanding the digraph G and determining the sensitivity functions and cost for each new edge, we can perform digraph optimization on the directed graph to update the optimal path for each vertex in G. The resulting minimum path imposes a switching order for the local policies. The set of local policies along with the switching order describes the approximately optimal policy. The optimal order of edges $\Gamma : \mathcal{G} \rightarrow \Pi$ (and, thus, local policies) from any initial hyperbelief defines the higher level policy, which we denote as $\gamma \in \Gamma$. The local policy between edges $\pi \in \Pi$ defines the lower policy.

The policy (with associated edges and new vertices) added to the graph at each stage has the potential to affect the optimal cost-go-to of its adjacent edges in the graph. We can avoid full graph optimization by recursively evaluating only the parents of each affected policy edge set and their neighbors. This pseudo code for this process is provided in Algorithm 7.

Each new policy edge set$\{\pi^{i \rightarrow j}\}_j$ that is generated during the expansion phase is push

Algorithm 7: OptimizeGraph

Input: G : digraph
$edge\_opt =$: set of initial edges to update
Output: G : digraph with the updated vertex information

while *edge_ opt not empty or a maximum number of iterations have occurred* **do**
  $E \leftarrow$ pop front of edge_opt ;
  $\pi \leftarrow$ policy of E ;
  foreach *target vertex* $j \in E$ *not already optimized* **do**
    $N_{local} \leftarrow$ find n nearest neighbors of j ;
    foreach $l \in N_{local}$ **do**
      val$[l] \leftarrow V^{l*}(\beta^j)$;
    best_val$[j] \leftarrow \min_l\{val[l]\}$;
  $Q^i(\cdot, \pi) \leftarrow$ Chain$(v_\pi^{i \to j}$, best_val) ;
  if $V^i(\beta^i, \pi) < V^{i*}(\beta^i)$ **then**
    $\pi_i^* \leftarrow \pi$;
    $V^{i*}(\cdot) \leftarrow V^i(\cdot, \pi)$;
    foreach E' *parent policy edge set of* i **do**
      push E' and its n-nearest neighbors onto edge_opt ;
return G;

onto a first-in-first-out queue *edge_ opt* of edges to be evaluated. Then, for each entry in edge_opt the cost of the parent vertices of $b^{(i)}$ (any incoming edge to $b^{(i)}$) are to be evaluated along $\pi^{i \to j}$ to the best value at target vertex $b^{(j)}$ and the chained cost-to-go sensitivity function determined. If the new cost reduces the cost the parent vertex $b^{(i)}$ then the parent's vertex is pushed onto the queue edge_opt and the optimal cost-to-go sensitivity function is assigned the updated composite function. The next edge in the queue edge_opt is selected and the process repeats. At most each of the policy edge sets in the graph being updated at each iteration, which has the same cost as Dijkstra's algorithm (refer to [79, Ch. 24]) over the graph. One appealing aspect of this approach is that often only a small subset of the edges are affected and the subsequent optimization at each iteration. For reward-based systems, where negative weight cycles are present, cycle detection is preformed to avoid infinite looping.

7.3 Results

POMDPs with an expected state cost have been studied extensively. This class of cost function imparts a simple and elegant structure on the value function in the belief space; namely, the optimal value function is piecewise-linear and concave (it is convex when considering a reward function instead of a cost function). Traditional value-iteration methods exploit this structure when determining approximately optimal solutions (e.g., [6, 8, 11]). This piecewise-linear and concave form in the value function arises from the fact that the cost function is linear in the belief space and that the backup from one stage to the next is a linear function for each action. The optimization function is then the minimum over the set of linear functions in the belief space, which is a piecewise linear function.

The unnormalized cost function is a ratio of linear function given by

$$c_u\left(\bar{b}\right) = \frac{\mathrm{c}_u^T \bar{b}}{\mathbf{1}^T \bar{b}},$$

for each $u \in \mathcal{U}$. We use a first order approximation of the cost:

$$\begin{aligned} \nabla c_u &= \frac{\mathrm{c}_u^T}{\mathbf{1}^T \bar{b}} - \frac{\mathrm{c}_u^T \bar{b} \mathbf{1}^T}{\left(\mathbf{1}^T \bar{b}\right)^2} \\ &= \frac{\mathrm{c}_u^T}{\mathbf{1}^T \bar{b}} \left(\mathbf{I} - \frac{\bar{b}\mathbf{1}^T}{\mathbf{1}^T \bar{b}}\right). \end{aligned}$$

This implies that we retain a first order approximation throughout. In this way, we retain some of the original structure of piecewise linearity. The nonlinearity is a result of the limited set of sample paths that are considered via the temporal abstraction. In single stage techniques, the effect of every observation is considered so the normalization term disappears and, thus, the nonlinearity of the normalization disappears as well.

To verify the proposed technique, we applied it to several of the benchmark problems found in the literature: maze20, hallway2, and CIT (from [2] and [11]). These methods are

Table 7.1: Verification via comparison to benchmark problems

				Expected Total Cost		Time (s)
	$\|\mathcal{X}\|$	$\|\mathcal{Y}\|$	$\|\mathcal{U}\|$	SARSOP	AESOP	
Maze20	20	8	6	-47.27	-48.965	50
Hallway2	92	17	5	-0.465	-0.435	100
CIT	284	28	4	-0.831	-0.664	200

discounted infinite horizon expected state reward systems. Just as with generating a bound for the cost, we generate a bound on the number of iterations that elapse while traversing edges of the digraph. This way we can propagate the discount from one event to the next via the number of iterations that elapse between events. The result of the the simulated results of the presented method are presented in Table 7.1. The proposed method is compared against SARSOP [11]. SARSOP is not a temporal/spatial abstraction method. However, it is the leading POMDP approximation method for the benchmark problems presented. We can see that the proposed method is competitive with SARSOP for the small systems and the proposed method exceeds SARSOP for several of the large systems. For each example, we ran each method 100 times to evaluate the effectiveness of the method. We note that there are several future improvements to the proposed method that will enable further improvements that should allow the method to expand to the larger systems. Specifically, approximation the sensitivity function is crucial to scale further.

7.4 Conclusion

A method for finding nearly optimal policies for POMDPs was presented. The proposed method is a sampling-based technique using a two-level hierarchical planner, whereby the lower level planner executes local, greedy feedback policies and the higher level planner coordinates the which of the local policies to execute. This method attempts to capture the connectivity of the POMDP system, while simultaneously learning the nearly optimal policy for the stated objective function. The method presented explicitly weighs the explo-

ration/exploitation trade-off through two sampling algorithms. One algorithm uses inductive bias to preferentially sample vertices, as represented by beliefs, and policies that are most likely to explore the reachable belief space most effectively. The other algorithm selects vertices and local policies for the sole purpose of decreasing the value function over the graph. The effectiveness is demonstrated on a set of benchmark problems.

The proposed method demonstrates the utility in including inductive bias in sampling-based methods. However, there are many opportunities to improve the performance of the algorithm. One promising method is to bootstrap the results of a similar system when optimizing a new system. Both the bias in exploration and exploitation used as a prior should drastically improve convergence of the proposed method. Further research could also focus on tuning the inductive bias algorithms to maximize their performance.

Chapter 8

DISCUSSION

We catalog potential future research directions in Section 8.1. In Section 8.2, we conclude a summary of contributions of the proposed research.

8.1 Future Research

8.1.1 Improved sampling methods

While sampling may be an effective method to approximate the set of feasible hyperbeliefs, the set of generated hyperbeliefs may not be indicative of sensitive regions in the hyperbelief space where the local policy changes and the value of fundamentally shifts. An adaptive sampling approach therefore may be beneficial. Numerous researchers have approached sampling from an adaptive framework (refer to [39, Section 7.1.3]). In fact, RRTs [72] are in some sense adaptive sampling methods. Many of these techniques either implicitly or explicitly search for points analogous to such sensitive regions. Searching for a path narrow gap [106], sampling near obstacles [107], or sampling based on manipulation constraints [108] are analogous in some ways to searching for a fundamental shift in a policy.

Adaptive sampling may performed either in the digraph generation stage or during the graph optimization stage. If performed during the digraph generation stage, numerous possible options exist. If one hyperbelief sample is incapable of simulating near another hyperbelief sample, a set of intermediate sample between the two hyperbeliefs can be sampled to determine if a better strategy can be found. Another possibility is to expand, for a limited

time horizon, the set of all possible actions, to determine if a better policy exists to transition one hyperbelief to the next. Adaptive sampling can also be used to connect under sampled portions of the hyperbelief space in a manner similar to that of PRMs.

8.1.2 Improved local planners

The techniques presented presume no information or guidance in selection of the local policy set. However, as we demonstrated in [109], often limited operator input in the local policy selection process can drastically speed convergence of the optimization algorithm when learning sufficient and effective policies. In particular, we observed that in a distributed multi-agent POMDP system, a semi-supervised controller that was endowed with only a limited number of local policies was capable of achieving non-intuitive and satisfying results. In this system, the local policies were informed by the problem description as well as the presumed optimal local behavior of each agents.

It has been observed that often the MDP policy is nearly optimal for POMDP systems when there is little uncertainty in the observation model. In fact both [8] and [11] explicitly take advantage of this fact. Interestingly, the observed behavior is a mixture between uncertainty minimization (exploration) and following the MDP trajectory (exploitation). Further streamlined by leveraging a limited set of local policies that can exploit the MDP solution while simultaneously reduce the uncertainty in the system is one potential method to select and improve local planners.

8.1.3 Sensitivity analysis for variations in the model description: evaluation for an entire class of systems

By analyzing the sensitivity of the system model to variations we wish to find an efficient and compact representation of the optimal policies over the entire space of system models. Just as the neighborhood around optimal policies is often nearly optimal, an optimal policy

for a given system is often nearly optimal for a similar system. This is a concept we wish to investigate further as future research. However, just as there are sensitive regions in the hyperbelief space where the policy changes, it stands to reason that so to there exist regions in the space defining the class of systems whereby the policy fundamentally changes as well. By investigating the efficient representations of the configuration space, Leven and Hutchinson in [110] found an efficient compression of the configuration space based on critical regions. In a similar fashion, we wish to derive a method based on the hyperbelief sampling to determine an entire set of approximately optimal policies, for an entire canonical class of systems. To do this, we propose defining a parametrized class of transition and observation probability functions. We would then seed the space with a representative set of parameterizations and explore the space around the solution until a sensitive region is found. We would then make sure there is a sufficient number of vertices in the digraph to capture the insensitive regions between this sensitive region.

We are also interested in exploring the sensitivity to see if we can derive the relationship between the value of the optimal policy and the interplay between the process model, the observation model, and the cost function. In particular we are interested in the impact of the relative uncertainty in the process model to the uncertainty in the observation model has on achieving some given objective. For instance, if the process model is subject to a great amount of uncertainty and the observation model is nearly uninformative, then it is likely that the evolution of the system spirals to the center of the belief space. However if the observation is informative and, thus, subject to a small amount of uncertainty, the system will be pushed to the boundary of the belief space. These two scenarios are drastically different and will likely lead drastically different values. The questions are: How does the value change between these two extremes? and Are there sensitive regions where the value changes suddenly? If so, is there anything we can derive from these regions that tells us something fundamental about the system in general, such as the maximal amount of uncertainty that can be tolerated by a system to still be able to perform a task satisfactorily? Evaluating

such insights into POMDP system remains as a topic of future research.

8.1.4 Extension to continuous, parametrized spaces

Unfortunately, it may not be possible to directly apply hyper-particle filtering to continuous state spaces. The hyperbelief space for finite state systems is itself an infinite dimensional space. The set of probability functions over a continuous space, on the other hand, resides in an infinite dimensional space. The space of probability functions defined on that infinite dimensional space may be poorly defined. Because of this current limitation of hyper-particle filtering to discrete state spaces, hyper-particle filtering fails to be applicable to significant portion of robotics problems. Specifically, motion planning in real world environments reside in continuous state spaces. Approximating the continuous state space by discretizing the space into a finite grid is often done in robotics. However, the number of cells required grows exponentially with the dimension of the state space. And the quality of the solution depends on the resolution of the cells. We are interested in better approximations than simply cutting the world into hypercubes of constant volume–that give a better approximation with fewer parameters.

To address this issue we propose turning to parametrized transition and observation probability functions, whereby the parameter space is finite dimensional. Thus, we can then look at the space of probability functions over the parameter space much in the same way we analyzed the space of probability functions defined over the belief simplex. Ideally, we wish to devise a formulation that has certain, beneficial properties: 1) the description of the beliefs requires only a finite number of parameters at each stage, 2) the number of parameters is either a constant from stage to stage or only increases by a linear to sub-linear amount from one stage to the next and 3) the parametrization is rich enough to approximate most practical systems of interest in robotics.

8.2 Conclusions

The development of techniques to learn nearly optimal policies for finite-time horizon and infinite-time horizon POMDPs was presented. The proposed methods use a bi-level hierarchical planner, whereby the lower level planner executes local feedback policies and the higher level planner coordinates the order of hyperbelief samples that are visited. They attempt to capture the structure of the POMDP system. The first method is independent of the starting hyperbelief (or belief) and the cost function, so that an efficient multi-query technique can be utilized for any initial hyperbelief or cost function for a given POMDP system. The second technique is derived from the first. As an anytime algorithm, it applies a multi-edge formulation and implements perturbation analysis to achieve both spatial and temporal abstraction. Finally, a method using a mixed information representation via belief and direct sensing data is presented. This method demonstrates the utility in a mixed representation as well exposing limitations of direct sensing implementations.

The resulting methods and analysis support the notion that POMDPs are inherently insensitive. When coupled to feedback policies the inherent sensitivity is amplified even further. The proposed techniques are not only are comparable in performance to existing approximation methods, but exceed existing methods for long time-horizon systems and for systems subject to a greater uncertainty in the observation model. Further, the structure-based formulation should enable a principled analysis of the effect changes in the system description have on the system's performance. Exploiting this should enable the development of a compact policy representation for a variety of system parameters.

Appendix A

STOCHASTIC FILTERING

For systems subject to uncertainty, *filtering* describes the process whereby an estimate of the state and its uncertainty are propagated from one stage to the next. Filtering is a sequential or recursive method whereby an estimate from the previous stage is used to determine an estimate for the current stage. *Hyperfiltering* is also a sequential method, but, while filtering evolves an estimate of the state and its uncertainty from previous stages to the current stage, hyperfiltering propagates the uncertainty and estimate of a system forward into future stages. The concept of filtering is presented as well as the relevant background material needed to motivate the development of hyperfiltering. Filtering is formulated in Section A.1 and is followed, in Section A.2, by a taxonomy of filtering approximation methods. The particle filter, the approximation technique on which the hyper-particle filter is based is presented in Section A.3.

A.1 Formulation

Filtering is a general term for processing sequential systems that are either causal or non-causal, whereby the likelihood of the system being in a particular state is estimated or "filtered" from one stage to the next. In this way the unlikely states are "filtered" out. For stochastic systems, filtering refers to a sequential technique that generates an estimate of the state and uncertainty from one stage to the next. Conveniently, filtering minimizes the amount of information that must be retained from previous stages because only the previous stage is used to estimate the current stage. Having an estimate of the state and a repre-

sentation of the uncertainty from one stage to the next is immensely useful in finding and implementing robust and even optimal control policies.

When dealing with partially observable systems, a construct is needed to encapsulate the known information. Unfortunately, the state is only indirectly known through the information state I_k. The *information state* is an accumulation of all of the information about a system that is directly known. This includes the set of actions performed and the set of observations collected up to time k. The question is how to incorporate all of the information into an estimate of the state of the system and its uncertainty. While some filtering methods take advantage of certain properties of r.v.'s, the essential nature of filtering Markovian systems is best described by the precise method known as the Bayesian filter.

A.1.1 Bayesian filter

Because of Markov properties, Bayes rule can be applied to perform sequential filtering. This method, known as Bayesian filtering was first introduced in [13]. Bayesian filtering estimates the belief at the current stage k from the previous belief at stage $k-1$. The evolution of the belief can be split into two stages: prediction and update. The prediction stage takes the previous belief, $p_{\mathbf{x}_{k-1}|I_{k-1}}$, and pushes it through the transition probability function to obtain the predicted current belief, $p_{\mathbf{x}_k|I_{k-1},u_k}$. The prediction step evaluates the effect of an action on the belief of the system. This is achieved by simply marginalizing $p(x_k|I_{k-1},u_{k-1})$ on x_{k-1}, which becomes

$$p(x_k|I_{k-1},u_{k-1}) = \sum_{x_{k-1}\in\mathcal{X}} p(x_k|x_{k-1},I_{k-1},u_{k-1})p(x_{k-1}|I_{k-1},u_{k-1}) \quad \text{(A.1)}$$

$$= \sum_{x_{k-1}\in\mathcal{X}} p(x_k|x_{k-1},u_{k-1})p(x_{k-1}|I_{k-1}). \quad \text{(A.2)}$$

In (A.1) the current belief is marginalized on x_{k-1}. The transition probability function is independent of I_{k-1} given x_{k-1}. Therefore, (A.1) reduces to (A.2). Likewise, because the previous belief is conditionally independent of u_{k-1}, $p(x_{k-1}|I_{k-1}, u_{k-1})$ becomes $p(x_{k-1}|I_{k-1})$ as shown in (A.2). The current belief is now represented in terms of the previous belief and the transition probability function.

After the prediction stage, the update stage is executed. The update stage incorporates an observation to condition the belief on new information to generate $p_{\mathbf{x}_k|I_k}$. After applying Bayes rule, the system of interest becomes

$$\begin{aligned} p(x_k|I_k) &= p(x_k|y_k, I_{k-1}, u_{k-1}) && \text{(A.3)} \\ &= \frac{p(y_k|x_k, I_{k-1}, u_{k-1})p(x_k|I_{k-1}, u_{k-1})}{p(y_k|I_{k-1}, u_{k-1})} && \text{(A.4)} \\ &= \eta_k p(y_k|x_k)p(x_k|I_{k-1}, u_{k-1}) && \text{(A.5)} \end{aligned}$$

where η_k is the normalizing constant

$$\frac{1}{\eta_k} = \sum_{x_k \in \mathcal{X}} p(y_k|x_k)p(x_k|I_{k-1}, u_{k-1}). \tag{A.6}$$

In (A.3), $p(x_k|I_k)$ is rewritten so that the y_k and u_{k-1} are pulled out of I_k. As shown in (A.4), Bayes rule is applied so that the probability of y_k is conditioned on x_k. The probability of y_k is independent of other terms when conditioned on x_k, thus the other terms are eliminated in (A.5). Also in (A.5), $p(y_k|u_{k-1}, I_{k-1})$ is a normalizing constant that can be determined as the sum of the probability of y_k over all possible x_k as is shown in (A.6).

By combining both the prediction (A.2) and update (A.5), the Bayesian filter is obtained:

$$p(x_k|I_k) = \eta_k p(y_k|x_k) \sum_{x_{k-1} \in \mathcal{X}} p(x_k|x_{k-1}, u_{k-1})p(x_{k-1}|I_{k-1}), \tag{A.7}$$

where η_k is as defined by (A.6). From (A.7), it is shown that $p_{\mathbf{x}_k|I_k}$ can be evaluated directly

from $p_{\mathbf{x}_{k-1}|I_{k-1}}$ and u_{k-1} using both the transition and observation probability functions. Likewise, if Bayes rule is applied to $p_{\mathbf{x}_{k-1}|I_{k-1}}$, the belief at stage $k-1$ requires only the belief and action at stage $k-2$. By continuing to expand (A.7), it becomes apparent that the Bayesian filter can be recursively applied up to the initial belief. This inductive step is crucial in understanding that the Bayesian filter can be sequentially applied to continuously evaluate the current belief given the previous belief.

Interestingly, by analyzing filtering from the belief perspective, the problem reduces to a deterministic function that transitions one belief into another. Thus, the belief at stage k can be determined from the belief at stage $k-1$.

Definition A.1. *The belief transition function $B(\cdot)$, where $B : \mathcal{P}_b \times \mathcal{U} \times \mathcal{Y} \to \mathcal{P}_b$, transfers one belief b_{k-1} into another b_k given some particular action u_{k-1} and observation y_k*, or

$$b_k = B(b_{k-1}, u_{k-1}, y_k),$$

where, with $x_k(i) \in \mathcal{X}$ *for* $i = 1, \ldots, |\mathcal{X}|$ *representing the set of states,*

$$B(b_{k-1}, u_{k-1}, y_k) \triangleq \begin{bmatrix} \eta_k p(y_k|x_k(1)) \sum_{x_{k-1}\in\mathcal{X}} p(x(1)|x_{k-1}, u_{k-1}) b_{k-1}(x_{k-1}) \\ \eta_k p(y_k|x_k(2)) \sum_{x_{k-1}\in\mathcal{X}} p(x_k(2)|x_{k-1}, u_{k-1}) b_{k-1}(x_{k-1}) \\ \vdots \\ \eta_k p(y_k|x_k(|\mathcal{X}|)) \sum_{x_{k-1}\in\mathcal{X}} p(x_k(|\mathcal{X}|)|x_{k-1}, u_{k-1}) b_{k-1}(x_{k-1}) \end{bmatrix}$$

and

$$\frac{1}{\eta_k} = \sum_{x_k\in\mathcal{X}} \sum_{x_{k-1}\in\mathcal{X}} p(y_k|x_k) p(x_k|x_{k-1}, u_{k-1}) b_{k-1}(x_{k-1}).$$

The Bayesian filtering description derived above was for discrete space systems. The continuous state analog is generated by replacing the summations with integrals. Many robotic systems evolve over continuous spaces. The difficulty with extending from discrete

to continuous spaces is the lack of closed form solutions to the integral equivalent of (A.2). It becomes necessary to find tractable approximations to the Bayesian filter to proceed. Fortunately, there exist a plethora of techniques focused on approximating the Bayesian filter for both continuous and discrete systems.

A.2 Filtering Approximation Methods

Most general filtering problems have no known analytical solution or an unacceptable running time, also known as computational time complexity. Because of inherent difficulties in filtering for general systems, approximation methods are typically the only feasible choice. For discrete systems difficulties arise from the $O(|\mathcal{X}|^2)$ computational time complexity in the evolution from one stage to the next, where $|\mathcal{X}|$ is the number of states. This means the worst case running time grows quadratically in the size of the state space. Often for realistic systems, there can be millions of states, and the square of this quantity can make the exact derivation of the next belief state prohibitively expensive. Reducing the running time to linear or sublinear computational time complexity is often desired for such systems.

While computational time complexity can be an issue with continuous time systems, it is more often the lack of a known closed form solution that makes approximation techniques necessary. As an exception, the Kalman filter (KF) [16], the most popular method for continuous state systems, is an exact filter for linear Gaussian systems. Taking advantage of the property that a random variable that is the superposition of jointly Gaussian random variables is itself a Gaussian random variable, Kalman derived a formulation to evolve the mean and covariance describing the probability function. This method's popularity holds even to this day.

The extended Kalman filter (EKF) was developed in an attempt to expand the KF method to general nonlinear systems (refer to [17]). The EKF linearizes the system around the current estimate and then applies the KF on the linearized system to update the esti-

mated parameters. The EKF has become especially popular in SLAM applications.

Numerous alternative methods, based directly on Bayesian filtering, have been developed. To expand past the Gaussian limitation of the KF, the Gaussian mixture method [18] was developed for linear systems subject to non-Gaussian noise. The mixture method works by approximating a non-Gaussian probability function by a sum of Gaussian probability functions. A major drawback to this method is the possible exponential growth, in the time horizon, of the number of Gaussians representing the belief.

Researchers in [19] tackle the problem of nonlinear Gaussian systems with the unscented filter. The unscented filter samples a Gaussian to estimate the belief and then passes the samples through the transition probability function. Once completed, a new Gaussian probability function is generated to fit the newly evolved samples. Set-theoretic methods are used to filter systems when no model besides the support of the noise is known. These systems are evaluated using forward projection techniques [111]. The approach of Hanebeck [20] and Stump et al. [21] is to employ sequential elliptical approximations to evaluate the uncertainty sets.

While some research has occurred for the class of problems listed above, the majority of work has focused on solutions to general nonlinear, non-Gaussian systems. The vast majority of this work has been for parametrized family solutions, including [22–24, 82]. Several researchers have researched mixture methods for parametrized families (e.g., [112, 113]) that meld the parametrized solution with the Gaussian mixture method.

More recently, sampling-based methods have become popular. Sampling methods take a computational approach to solving the filtering problem, whereby at each stage a finite set of points is used to evaluate an approximation to the exact filtering outcome. A grid-based method introduced in [114] deterministically samples the sample space by using a grid-based approximation over the state space. For continuous systems, using this approach makes it possible to reduce a possibly unknown analytical solution to an approximate computational solution. For discrete systems, sampling can reduce the computational time complexity sig-

nificantly by considering only a limited number of the states in the state space. A major drawback of this deterministic approach is that the computational burden grows exponentially with the dimension of the state space. To alleviate this drawback, motivated by the convergence properties of random sampling, an alternative random sampling method known as particle filtering has become the defacto standard when filtering nonlinear, non-Gaussian systems. Particle filtering is used as a basis for an approximation for the hyperfilter and, for this reason, particle filtering is described in detail below.

A.3 The Particle Filter

Unlike deterministic sampling, e.g. grid based methods, the particle filter [58–69] randomly samples the space. Particle filtering is a sequential method that is simple to implement and has many desirable properties. Particle filtering is based on Monte Carlo Markov chain (MCMC) simulation, which is influenced by Monte Carlo integration. Unlike particle filtering, MCMC is an iterative method requiring the re-evaluation of all information at every stage [58]. At each stage k, MCMC methods sample a series of states from the initial stage to the current stage. The probability of the sample is then evaluated by simulating the system forward from the initial stage until the current stage k. At the next stage $k+1$, the MCMC methods, again, generate a new set of state samples for each stage from the initial stage to stage $k+1$. This process is an iterative technique whereby no information from the previous stage is retained. Iterative techniques have worst-case running times that are geometric in the number of states at each stage. Because of this, the entire computational time complexity burden for estimating the system from an initial stage to a given time horizon results in a computational time complexity that is exponential in the time horizon. Particle filtering, on the other hand, is sequential in nature. At each stage the approximation of the current belief is evaluated from the approximation of the previous belief.

Particle filtering approximates the probability function by a finite set of samples instead

of performing exact filtering. Each sample consists of a two elements: a weight (probability) and a point in the state space. Because the representation is discrete, the evolution through the Bayesian filtering equations becomes computational in nature. Hence, particle filtering allows for the evaluation of a broad class of nonlinear, non-Gaussian systems.

Particle filtering is known as bootstrap filtering [60], condensation [65], sequential Monte Carlo [66], interacting particle approximations [67], and survival of the fittest [68]. Regardless of the name, one of the benefits of particle filtering is that, under general conditions, the convergence or the error is defined in the number of samples, not the dimension of the state space. Furthermore, as was shown in [69] (and subsequently in [58]), particle filtering converges under fairly weak assumptions.

Unlike most derivations, the formulation of the particle filtering algorithm, to be presented in this section, is specifically separated into a prediction and an update step. The reason for this is to facilitate the adaptation of the particle filtering method into the hyper-particle filtering approach (see Chapter 4). Particle filtering can also be applied to discrete state systems as will be done when exploring the hyper-particle filter. The following derivation is based on the sequential importance resampling method (SIR) (see [57] for further description). There are numerous other adaptations to particle filtering that can be applied; however, the fundamental approach in all of these methods is the same.

Particle filtering approximates the probability function of a system with a finite set of particles $\mathcal{S} = \{s^i\}$. Each particle $s^i = (x_k^i, w_k^i)$ comprises as a point x^i in the state space $\mathcal{X}$ and a scalar weight w^i, where $0 < w^i \leq 1$ and $\sum_i w^i = 1$. At any given stage, k, the belief of the system is approximated by the set of particles $\mathcal{S}_k = \{s_k^i\}$ as

$$p(x_k|I_k) \approx \sum_{i=1}^{|\mathcal{S}_k|} w_k^i \delta(x_k - x_k^i).$$

Particle filtering begins by taking m random samples of the initial belief and giving them equal weight of $1/m$. The primary portion of the method, repeated at each iteration,

performs the approximated Bayesian filtering on the set of particles $\mathcal{S}_k$ for each stage k. Instead of calculating each possible future state from each current state, particle filtering works by randomly sampling a set of next states using an importance sampling function $q(\cdot)$. Because state and observation spaces are finite and the observation y_k is already known, it is possible to sample from the optimal choice of importance sampling function, which is

$$q(x_k|x^i_{k-1}, y_k, u_{k-1}) \triangleq p(x_k|x^i_{k-1}, y_k, u_{k-1}), \tag{A.8}$$

as was shown in [66]. The importance sampling function (A.8) can be expanded by applying Bayes rule to become

$$\begin{aligned} q(x_k|x^i_{k-1}, y_k, u_{k-1}) &= p(x_k|x^i_{k-1}, y_k, u_{k-1}) \\ &= \frac{p(y_k|x_k, x^i_{k-1}, u_{k-1})p(x_k|x^i_{k-1}, u_{k-1})}{p(y_k|x^i_{k-1}, u_{k-1})} \\ &= \frac{p(y_k|x_k)p(x_k|x^i_{k-1}, u_{k-1})}{p(y_k|x^i_{k-1}, u_{k-1})}, \end{aligned}$$

where the later equation follows from the conditional independence of y_k on previous states (i.e., x^i_{k-1}). Often $q(x_k|x_{k-1,}u_{k-1})$ is chosen as the importance sampling function because it is often simple to generate samples from x_{k-1} and u_{k-1}.

From the previous set of particle samples, $\mathcal{S}_{k-1} = \{(x^i_{k-1}, w^i_{k-1})\}^m_{i=1}$, a set of new state samples $\{x^j_k\}^m_{j=1}$ are randomly generated using $q(x_k|x_{k-1,}u_{k-1})$ as the importance sampling function, where one sample x^j_k is generated for each x^i_{k-1} in $\mathcal{S}_{k-1}$. The weight, $\hat{w}^j_k$, representing the probability of each new sampled state, x^j_k, is then determined to generate the new particle set $\mathcal{S}_k = \{(x^j_k, \hat{w}^j_k)\}^m_{j=1}$. The probability of *any* state $x_k \in \mathcal{X}$ at time k is obtained

from the previous belief at stage $k-1$ and the transition probability function as

$$p(x_k|I_{k-1}, u_{k-1}) = \sum_{x_{k-1}\in\mathcal{X}} p(x_k|x_{k-1}, u_{k-1})p(x_{k-1}|I_{k-1}) \quad (A.9)$$

$$\approx \sum_{i=1}^{m} p(x_k|x_{k-1}^i, u_{k-1})w_{k-1}^i. \quad (A.10)$$

In (A.11), the exact predicted belief is approximated from the particle set at stage $k-1$. The weight w_{k-1}^i is the approximated probability of $p(x_{k-1}^i|I_{k-1})$ for each x_{k-1}^i in $\mathcal{S}_{k-1}$.

To simplify the analysis, particle filtering methods assume that the transition probability of each particle x_k^j is nonzero for only the sample x_{k-1}^i used to generate it. For discrete systems this approximation reduces the computational time complexity significantly. Taking this approximation into account, (A.11) reduces to

$$p(x_k^j|I_{k-1}, u_{k-1}) \approx \eta_p p(x_k^j|x_{k-1}^i, u_{k-1})w_{k-1}^i, \quad (A.11)$$

where η_p is a normalizing constant. Because each sample x_k^j was sampled randomly from an importance sampling function, an adverse effect is introduced. Because the samples are generated from an importance sampling function and not the actual probability transition function, the random samples are not representative of the random samples that would be generated if the probability transition function was sampled directly and therefore the representation of the posterior probability function is skewed. Without taking into account this effect, the result can quickly become erroneous.

The issue is that, on one hand, if one samples from the transition function the weight should be identical for each sample, as the sampled set will approximate the probability function itself. However, if one sampled the space uniformly, each sample should be weighted according to the probability of each sample. The question is then how to take into account the adverse effect when performing quasi-random or random sampling from a probability function other than the transition probability function. If each sample is just given equal

weight, the approximation becomes that of the importance sampling function and not the transition probability function. However, if each sample is weighted only according to the transition probability function, the approximated probability function becomes erroneous. Imagine sampling from an importance sampling function that is a Gaussian centered in the state space. If the transition probability function has a low probability of occurring around the center of the state space so that each sample receives a low weight, the samples get assigned a higher weight when normalized and the set of samples are focused in the center of the space. The result is an inaccurate representation of the true probability function.

Particle filtering researchers rely on insights from Monte Carlo integration to deal with this issue. When approximating the expectation of some bounded function $c(\cdot)$ relative to some probability function $p(\cdot)$ by a randomly generated, finite set of samples, the set of samples generated can adversely effect the result. It turns out that this adverse effect can be eliminated by weighting each sample by the ratio of the probability of the sample being generated by the transition probability function divided by the probability of the sample being generated by the importance sampling function. More precisely, as observed in Monte Carlo integration, for some function $c(\cdot)$,

$$E[c(x_k)] = \sum_{x \in \mathcal{X}} c(x)p(x) = \sum_{x \in \mathcal{X}} c(x)\frac{p(x)}{q(x)}q(x).$$

The expectation of a r.v. with a probability function $p(x)$ can be represented as the expectation of another r.v. with the probability function $q(x)$ by weighting $c(x)$ by the ratio of $p(x)$ and $q(x)$ for each $x \in \mathcal{X}$. Thus, as can be seen, the adverse effect of the importance sampling on the expected value of any bounded function $c(\cdot)$ is eliminated. As the effect of the bias relative to any $c(\cdot)$ is attenuated, any measure over the probability function has the effect of the bias attenuated. By dividing by $p_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1}}$ by $q_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1},y_k}$, the expected adverse effect in (A.11) relative to any bounded function $c(\cdot)$ is therefore attenuated and the

weight $\hat{w}_k^j$ associated with x_k^j becomes

$$p(x_k^j|I_{k-1}, u_{k-1}) \approx \eta_p \frac{p(x_k^j|x_{k-1}^i, u_{k-1})}{q(x_k^j|x_{k-1}^i, u_{k-1}, y_k)} w_{k-1}^i \tag{A.12}$$

$$= \hat{w}_k^j. \tag{A.13}$$

When the transition probability function is selected as the importance sampling function and is substituted into (A.12), the updated weight becomes $\hat{w}_k^i = w_{k-1}^i$. Thus the adverse effect is eliminated so that weight for each new particle is just the weight with the previous particle. The precise particle filtering prediction algorithm is given in Algorithm 9. Because the systems of interest are discrete, it is always possible to sample from $p_{x_k|x_{k-1},u_{k-1}}$ directly. Thus, the effect of sampling from an importance sampling function becomes moot.

Algorithm 9: Particle filter prediction

Input: $\mathcal{S} = \{w^i, x^i\}_{i=1}^m$: Set of particles and weights,
u : Applied control action ,
y : Observation$n$
Output: $\hat{\mathcal{S}}$: Updated particle set

for $i = 1, \cdots, m$ **do**
  Sample $\hat{x}^i$ from $q(\cdot \mid x^i, u, y)$;
  $\hat{w}^i \leftarrow w^i \frac{p(\hat{x}^i|x^i,u,y)}{q(\hat{x}^i|x^i,u,y)}$;
$\frac{1}{\eta_p} \leftarrow \sum_{i=1}^m \hat{w}^i$;
for $i = 1, \cdots, m$ **do**
  $\hat{w}^i \leftarrow \eta_p \hat{w}^i$;
$\hat{S} \leftarrow \{\hat{w}^i, \hat{x}^i\}_{i=1}^m$;
$\hat{S} \leftarrow$PF_update($\hat{S}$,y) ;
return $\hat{S}$

The update procedure weights of each particles according to the probability of each

sample given the observation y_k. With η_u, a normalizing constant, the new weight becomes

$$\begin{aligned} p(x_k^i|I_k) &= \frac{p(y_k|x_k^i)p(x_k^i|I_{k-1},u_{k-1})}{p(y_k|I_{k-1},u_{k-1})} \\ &\approx \eta_u p(y_k|x_k^i)\hat{w}_k^i \\ &= w_k^i \end{aligned}$$

by using (A.13) as an approximation of $p(x_k^i|I_{k-1},u_{k-1})$. The particle location does not change in the update step of the particle filtering algorithm. Instead, the weight is only amplified or attenuated. The update algorithm is described in Algorithm 10.

Algorithm 10: Particle filter update (PF_update)

Input: $\hat{\mathcal{S}} = \{\hat{w}^i, \hat{x}^i\}_{i=1}^m$: Predicted particle set,
y : Observation
Output: S : Updated particle set

for $i = 1, \cdots, m$ **do**
  $w^i \leftarrow \hat{w}^i p(y \mid \hat{x}^i)$;
$\frac{1}{\eta_u} \leftarrow \sum_{i=1}^m w^i$;
for $i = 1, \cdots, m$ **do**
  $w^i \leftarrow \eta_u w^i$;
$\mathcal{S} \leftarrow \{w^i, x^i\}_{i=1}^m$;
Estimate particle divergence of S ;
if *particle divergence greater than threshold* **then**
  $\mathcal{S} \leftarrow$ PF_resample(S) ;
return S

Once the new weight is generated, a resampling algorithm is executed. The resampling algorithm selects a set of m random samples from the belief approximated by the particle set. Resampling avoids particle degeneracy, which occurs when low weight, or low probability, particles continue to be utilized. Particle degeneracy can, and usually does, cause problems because higher probability regions are undersampled and lower probability regions are oversampled, resulting in a poor representation of the actual probability function. However, resampling has the potential side effect of causing particle impoverishment, whereby particles with high weights are selected many times. This is especially a concern in cases when

the system is subject to a small process noise [57]. The resampling algorithm is described in Algorithm 11.

Algorithm 11: Particle filter resample (PF_resample)

Input: $\mathcal{S} = \{w^i, x^i\}_{i=1}^m$: Particle set before resample,
y : Observation
Output: $\bar{S}$: Resampled particle set

for $j = 1, \cdots, m$ **do**
  $c_{j+1} \leftarrow c_j + w^j$;
Draw initial sample u_0 from uniform density over $[0, \frac{1}{m}]$;
$i \leftarrow 2$;
for $j = 1, \cdots, m$ **do**
  $u_j \leftarrow u_j + \frac{j-1}{m}$;
  while $u_j > c_i$ **do**
    $i \leftarrow i + 1$;
  $\bar{w}^j \leftarrow \frac{1}{m}$;
  $\bar{x}^j \leftarrow x^{i-1}$;
$\bar{S} \leftarrow \{\bar{w}^j, \bar{x}^j\}_{j=1}^m$;
return $\bar{S}$

The prediction stage has an $O(lm)$ computational time complexity, where m is the number of samples, and l is the computational time complexity of drawing a random sample from $q_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1},y_k}$. The update stage has a computational time complexity of $O(m)$, which includes the $O(m)$ computational time complexity of the resampling procedure. The algorithmic complexity of the particle filter up to the time horizon K is therefore $O(Kml)$. This computational time complexity includes all stages including the prediction, update, and resampling. When samples are generated from $p_{\mathbf{x}_k|\mathbf{x}_{k-1},u_{k-1}}$, the computational time complexity of the sampling procedure is $O(|\mathcal{X}|)$. This occurs because the sampling procedure requires that a sample be generated from an uniform probability function. The sample is then indexed into the cumulative distribution function generated from the transition probability function, which has $|\mathcal{X}|$ possibilities. This process is similar to the resampling procedure outlined in Algorithm 11. Thus, when the importance sampling function is chosen as the transition probability function, particle filtering has a computational time complexity of

$O(Km|\mathcal{X}|)$. While particle filtering was originally derived for continuous space systems, it can be applied as an approximation to discrete systems to reduce the computational burden of finding the exact solution. The exact solution for the discrete case is $O(K|\mathcal{X}|^2)$. In many robotics applications there can be tens of millions of states. Evaluation of such a system is not practical with today's computational means and the $O(Kml)$ time complexity of the particle filter is preferable to the exact solution.

REFERENCES

[1] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 4, pp. 31–43, 2010.

[2] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, Jan 1998.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: The MIT Press, 2005.

[4] R. Kaplow, "Point-based pomdp solvers: Survey and comparative analysis," Ph.D. dissertation, McGill University, 2011.

[5] S. Thrun, "Monte Carlo POMDPs," in *Advances in Neural Information Processing Systems*, 2000, pp. 1064–1070.

[6] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003, pp. 1025 – 1032.

[7] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 2004, pp. 520–527.

[8] T. Smith and R. Simmons, "Point-based POMDP algorithms: Improved analysis and implementation," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2005.

[9] M. T. Spaan and N. Vlassis, "A point-based POMDP algorithm for robot planning," in *IEEE International Conference on Robotics and Automation*, 2004, pp. 2399–2404.

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

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

[12] R. He, A. Bachrach, and N. Roy, "Efficient planning under uncertainty for a target-tracking micro-aerial vehicle," in *IEEE International Conference on Robotics and Automation*, 2010.

[13] K. J. Astrom, "Optimal control of Markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[14] D. P. Bertsekas, *Dynamic Programming and Optimal Control.* Cambridge, MA: Athena Scientific, 2007.

[15] G. Casella and R. L. Berger, *Statistical Inference* (2nd Edition). Pacific Grove, CA: Thomson Learning, Inc., 2002.

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

[17] H. W. Sorenson, *Kalman Filtering: Theory and Applications.* New York, NY: IEEE Press, 1985.

[18] D. L. Alspach and H. W. Sorenson, "Nonlinear Bayesian estimation using Gaussian sum approximation," *IEEE Transactions on Automatic Control*, vol. 17, no. 4, pp. 439–448, 1972. [Online]. Available: http://www.cs.rpi.edu/~isler/new/pub/pubs/tr-04-13.pdf

[19] S. Julier and J. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, March 2004.

[20] U. Hanebeck, "Recursive nonlinear set-theoretic estimation based on pseudo ellipsoids," in *International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2001, pp. 159–164.

[21] E. Stump, B. Grocholsky, and V. Kumar, "Extensive representations and algorithms for nonlinear filtering and estimation," in *International Workshop on the Algorithmic Foundations of Robotics*, 2006.

[22] U. D. Hanebeck, K. Briechle, and A. Rauh, "Progressive Bayes: A new framework for nonlinear state estimation," in *Proceedings of SPIE*, 2003, pp. 256–267.

[23] V. M. Klumpp, D. Brunn, and U. D. Hanebeck, "Approximate nonlinear Bayesian estimation based on lower and upper densities," in *The 9th International Conference on Information Fusion*, 2006, pp. 1–8.

[24] X. Boyen and D. Koller, "Tractable inference for complex stochastic processes," in *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 33–42.

[25] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable Markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, Sep. 1973.

[26] E. Sondik, "The optimal control of partially observable Markov processes," Ph.D. dissertation, Stanford University, 1971.

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

[28] G. E. Monahan, "A survey of partially observable Markov decision processes: Theory, models, and algorithms," *Management Science*, vol. 28, no. 1, pp. 1–16, Jan. 1982.

[29] H. T. Cheng, "Algorithms for partially observable Markov decision processes," Ph.D. dissertation, University of British Columbia, Vancouver, BC, Canada, 1988.

[30] N. L. Zhang and W. Liu, "Planning in stochastic domains: Problem characteristics and approximation," Department of Computer Science, Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS96-31, 1996.

[31] A. Cassandra, M. L. Littman, and N. L. Zhang, "Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes," in *Proceedings of Uncertainty in Artificial Intelligence*, 1997, pp. 54–61.

[32] C. Papadimitriou and J. Tsitsiklis, "The complexity of Markov decision processes," *Mathematics of operations research*, pp. 441–450, 1987.

[33] P. Poupart and C. Boutilier, "Value directed compression of POMDPs," in *Advances in Neural Information Processing Systems*, 2003.

[34] N. Roy and G. Gordon, "Exponential family PCA for belief compression in POMDPs," in *Advances in Neural Information Processing Systems*, 2002, pp. 1–8.

[35] X. Li, W. Cheung, and J. Liu, "Improving POMDP tractability via belief compression and clustering," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 40, no. 1, pp. 125–136, 2010.

[36] E. Zhou, M. Fu, and S. Marcus, "Solving continuous-state POMDPs via density projection," *IEEE Transactions on Automatic Control*, vol. 55, no. 5, pp. 1101–1116, 2010.

[37] R. Kaplow, A. Atrash, and J. Pineau, "Variable resolution decomposition for robotic navigation under a pomdp framework," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 369–376.

[38] X. Li, W. Cheung, and J. Liu, "Decomposing large-scale POMDP via belief state analysis," in *IEEE Conference on Intelligent Agent Technology*, 2005, pp. 428–434.

[39] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.

[40] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, June 1996.

[41] J. Kim, R. Pearce, and N. Amato, "Extracting optimal paths from roadmaps for motion planning," *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 2, pp. 2424–2429 vol.2, 14-19 Sept. 2003.

[42] M. Apaydin, D. Brutlag, C. Guestrin, D. Hsu, J. Latombe, and C. Varm, "Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion," *Journal of Computational Biology*, vol. 10, pp. 257–281, 2003.

[43] R. Alterovitz, T. Simeon, and K. Goldberg, "The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty," in *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[44] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in belief space by factoring the covariance," *International Journal of Robotics Research*, vol. 28, no. 11-12, pp. 1448–1465, 2009.

[45] R. He, E. Brunskill, and N. Roy, "Efficient Planning under Uncertainty with Macro-actions," *Journal of Artificial Intelligence Research*, vol. 40, pp. 523–570, 2011.

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

[47] H. Kurniawati, Y. Du, D. Hsu, and W. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," *International Journal of Robotics Research*, vol. 30, no. 3, p. 308, 2011.

[48] S. Candido, J. Davidson, and S. Hutchinson, "Exploiting domain knowledge in planning for uncertain robot systems modeled as POMDPs," in *IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, May 2010, pp. 3596–3603.

[49] R. He, E. Brunskill, and N. Roy, "PUMA: Planning under uncertainty with macro-actions," in *Proceedings of the American Association for Artificial Intelligence*, 2010, pp. 1089–1095.

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

[51] N. Melchior and R. Simmons, "Particle RRT for path planning with uncertainty," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 1617–1624.

[52] E. Hansen and R. Zhou, "Synthesis of hierarchical finite-state controllers for POMDPs," in *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling*, 2003, pp. 113–122.

[53] J. Pineau, G. Gordon, and S. Thrun, "Policy-contingent abstraction for robust robot control," 2003, pp. 477–484.

[54] M. Toussaint, L. Charlin, and P. Poupart, "Hierarchical POMDP controller optimization by likelihood maximization," 2008.

[55] L. Charlin, P. Poupart, and R. Shioda, "Automated hierarchy discovery for planning in partially observable environments," in *Advances in Neural Information Processing Systems*, 2007, pp. 225–232.

[56] J. Davidson and S. Hutchinson, "Hyper-particle filtering for stochastic systems," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 2770–2777.

[57] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[58] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. New York, NY: Springer Verlag, 2001.

[59] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 50, no. 2, pp. 174–188, February 2002.

[60] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *Radar and Signal Processing, IEE Proceedings F*, vol. 140, no. 2, pp. 107–113, 1993.

[61] C. Kwok, D. Fox, and M. Meila, "Real-time particle filters," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 469–484, March 2004.

[62] S. Thrun, "Particle filters in robotics," in *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence*, 2002.

[63] D. Fox, "Adapting the sample size in particle filters through kld-sampling," *International Journal of Robotics Research (IJRR)*, vol. 22, pp. 985–1003, 2003.

[64] J. H. Kotecha and P. M. Djuric, "Gaussian sum particle filtering," *IEEE Transactions on Signal Processing*, vol. 51, pp. 2602–2612, Oct. 2003.

[65] M. Isard and A. Blake, "Condensation – conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998. [Online]. Available: citeseer.ist.psu.edu/isard98condensation.html

[66] A. Doucet, "On sequential simulation-based methods for Bayesian filtering," Department of Engineering, University of Cambridge, Tech. Rep. CUED/F-INFENG, TR. 310, 1998.

[67] D. Crisan, P. D. Moral, and T. Lyons, "Discrete filtering using branching and interacting particle systems," *Markov Processes and Related Fields*, vol. 5, no. 3, pp. 293–318, 1999.

[68] K. Kanazawa, D. Koller, and S. Russell, "Stochastic simulation algorithms for dynamic probabilistic networks," in *Proceedings of the 11th Annual Conference on Uncertainty in Artifical Intelligence*, 1995, pp. 346–351.

[69] D. Crisan and A. Doucet, "Convergence of sequential Monte Carlo methods," Cambridge University, Tech. Rep. CUED/FINFENG, TR381, 2000.

[70] D. Hsu, W. S. Lee, and N. Rong, "What makes some pomdp problems easy to approximate?" in *Advances in Neural Information Processing Systems*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. Cambridge, MA: MIT Press, 2008.

[71] J. Davidson and S. Hutchinson, "A sampling hyperbelief optimization technique for stochastic systems," in *International Workshop on the Algorithmic Foundations of Robotics*, 2009, pp. 217–231.

[72] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *New Directions in Algorithmic and Computational Robotics*, B. R. Donald, K. Lynch, and D. Rus, Eds. AK Peters, 2001, pp. 293–308.

[73] M. Morales, S. Rodriguez, and N. M. Amato, "Improving the connectivity of PRM roadmaps," in *IEEE International Conference on Robotics and Automation*, 2003, pp. 4427–4432.

[74] R. He, S. Prentice, and N. Roy, "Planning in information space for a quadrotor helicopter in a GPS-denied environment," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 1814–1820.

[75] C. Holleman and L. E. Kavraki, "A framework for using the workspace medial axis in PRM planners," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 1408–1413.

[76] A. L. Gibbs and F. E. Su, "On choosing and bounding probability metrics," *International Statistical Review*, vol. 70, pp. 419–435, 2002.

[77] E. Levina and P. Bickel, "The earth mover's distance is the mallows distance: some insights from statistics," *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2, pp. 251–256 vol.2, 2001.

[78] S. Łukaszyk, "A new concept of probability metric and its applications in approximation of scattered data sets," *Computational Mechanics*, vol. 33, no. 4, pp. 299–304, 2004.

[79] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2002.

[80] F. Le Gland and N. Oudjane, "Stability and uniform approximation of nonlinear filters using the Hilbert metric and application to particle filters," *The Annals of Applied Probability*, vol. 14, no. 1, pp. 144–187, 2004.

[81] S. Candido and S. Hutchinson, "Minimum uncertainty robot path planning using a POMDP approach," in *IEEE International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, October 2010, pp. 1408–1413.

[82] X. Boyen and D. Koller, "Exploiting the architecture of dynamic systems," in *Proceedings of the 16th National Conference on Artificial Intelligence*, 1999, pp. 313–320.

[83] N. Berglund, "Perturbation theory of dynamical systems," *Arxiv preprint math. HO/0111178*, 2001.

[84] M. Stone, "The generalized weierstrass approximation theorem," *Mathematics Magazine*, vol. 21, no. 5, pp. 237–254, 1948.

[85] D. Cacuci, *Sensitivity and uncertainty analysis: Theory*. CRC Press, 2003, vol. 1.

[86] C. Meyer, "Sensitivity of the stationary distribution of a Markov chain," *SIAM Journal on Matrix Analysis and Applications*, vol. 15, no. 3, pp. 715–728, 1994.

[87] G. Cho and C. Meyer, "Comparison of perturbation bounds for the stationary distribution of a Markov chain," *Linear Algebra and its Applications*, vol. 335, no. 1-3, pp. 137–150, 2001.

[88] X. Cao, "A unified approach to Markov decision problems and performance sensitivity analysis," *Automatica*, vol. 36, no. 5, pp. 771–774, 2000.

[89] F. Le Gland and L. Mevel, "Exponential forgetting and geometric ergodicity in hidden Markov models," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 13, no. 1, pp. 63–93, 2000.

[90] P. Jacquet, G. Seroussi, and W. Szpankowski, "On the entropy of a hidden Markov process," *Theoretical computer science*, vol. 395, no. 2, pp. 203–219, 2008.

[91] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, no. 22, 2000.

[92] J. Baxter and P. Bartlett, "Infinite-horizon policy-gradient estimation," *J. Artif. Intell. Res. (JAIR)*, vol. 15, pp. 319–350, 2001.

[93] X. Cao, "A basic formula for online policy gradient algorithms," *IEEE Transactions on Automatic Control*, vol. 50, no. 5, p. 697, 2005.

[94] S. Ross, M. Izadi, M. Mercer, and D. Buckeridge, "Sensitivity analysis of pomdp value functions," in *Machine Learning and Applications, 2009. ICMLA'09. International Conference on*. IEEE, 2009, pp. 317–323.

[95] R. B. Ash, *Information Theory*. New York, NY: Dover Publications, 1990.

[96] R. Atar and O. Zeitouni, "Exponential stability for nonlinear filtering," in *Annales de l'Institut Henri Poincare (B) Probability and Statistics*, vol. 33, no. 6. Elsevier, 1997, pp. 697–725.

[97] R. Douc, G. Fort, E. Moulines, and P. Priouret, "Forgetting the initial distribution for hidden Markov models," *Stochastic processes and their applications*, vol. 119, no. 4, pp. 1235–1256, 2009.

[98] M. A. Tanner, *Tools for statistical inference: methods for the exploration of posterior distributions and likelihood functions*. Springer Verlag, 1996.

[99] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of symbolic computation*, vol. 9, no. 3, pp. 251–280, 1990.

[100] P. Drineas and R. Kannan, "Fast monte-carlo algorithms for approximate matrix multiplication," in *Annual Symposium on Foundations of Computer Science*, vol. 42. IEEE Computer Society Press, 2001, pp. 452–459.

[101] R. Bishop and S. Goldberg, *Tensor analysis on manifolds*. New York, NY, USA: Dover Publications, 1980.

[102] T. Ma, "Higher chain formula proved by combinatorics," *the electronic journal of combinatorics*, vol. 16, no. 21, p. 1, 2009.

[103] D. Liberzon, *Switching in Systems and Control*. Boston, MA: Birkhäuser, 2003.

[104] J. Bentley, "Multidimensional divide and conquer," *Communications of the ACM*, vol. 23, no. 4, 1980.

[105] S. Kukaszyk, "A new concept of probability metric and its applications in approximation of scattered data sets," *Computational Mechanics*, vol. 33, no. 4, pp. 299–304, 2003.

[106] D. Hsu, L. E. Kavraki, J. C. Latombe, R. Motwani, and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners," in *Robotics: The Algorithmic Perspective*, e. a. P. Agarwal, Ed. Wellesley, MA: A.K. Peters, 1998, pp. 141–154.

[107] N. M. Amato, B. Bayazit, L. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3d workspaces," in *Robotics: The Algorithmic Perspective*, P. Agarwal, L. E. Kavraki, and M. Mason, Eds. AK Peters, 1998, pp. 156–168.

[108] P. Leven and S. Hutchinson, "Using manipulability to bias sampling during the construction of probabilistic roadmaps," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 6, pp. 1020–1026, Dec. 2003.

[109] S. Candido, J. Davidson, and S. Hutchinson, "Exploiting domain knowledge in planning for uncertain robot systems modeled as POMDPs," in *IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, May 2010, pp. 3596–3603.

[110] P. Leven and S. Hutchinson, "Real-time path planning in changing environments," *International Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, Dec. 2002.

[111] S. M. LaValle, *Planning Algorithms*. Cambridge, MA: Cambridge University Press, 2006.

[112] M. Huber, D. Brunn, and U. D. Hanebeck, "Closed-form prediction of nonlinear dynamic systems by means of Gaussian mixture approximation of the transition density," in *International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2006, pp. 98–103.

[113] J. Li and A. Barron, "Mixture density estimation," in *Advances in Neural Information Processing Systems*, 2000, pp. 279–285.

[114] W. Lovejoy, "Computationally feasible bounds for partially observed Markov decision processes," in *Operations Research*, vol. 39, 1991, pp. 162–175.