Abstraction for Bayesian Reinforcement Learning in Factored POMDPs

Anonymous authors
Paper under double-blind review

Abstract

Bayesian reinforcement learning provides an elegant solution to addressing the exploration–exploitation trade-off in Partially Observable Markov Decision Processes (POMDPs) when the environment's dynamics and reward function are initially unknown. By maintaining a belief over these unknown components and the state, the agent can effectively learn the environment's dynamics and optimize their policy. However, scaling Bayesian reinforcement learning methods to large problems remains to be a significant challenge. While prior work has leveraged factored models and online sample-based planning to address this issue, these approaches often retain unnecessarily complex models and factors within the belief space that have minimal impact on the optimal policy. While this complexity might be necessary for accurate model learning, in reinforcement learning, the primary objective is not to recover the ground truth model but to optimize the policy for maximizing the expected sum of rewards. Abstraction offers a way to reduce model complexity by removing factors that are less relevant to achieving high rewards. In this work, we propose and analyze the integration of abstraction with online planning in factored POMDPs. Our empirical results demonstrate two key benefits. First, abstraction reduces model size, enabling faster simulations and thus more planning simulations within a fixed runtime. Second, abstraction enhances performance even with a fixed number of simulations due to greater statistical strength. These results underscore the potential of abstraction to improve both the scalability and effectiveness of Bayesian reinforcement learning in factored POMDPs.

1 Introduction

Deep reinforcement learning methods have achieved significant milestones, such as attaining superhuman performance on Atari games with only 100k frames (Ye et al., 2021), solving highly complex games such as Go (Silver et al., 2016), and achieving high performance in simulated control tasks (Haarnoja et al., 2018). Most of these achievements rely on recent function approximation advances made with the work on deep neural networks. However, despite these advances, Reinforcement Learning (RL) still faces critical hurdles that must be addressed to enable its application in diverse real-world scenarios. One of the most pressing challenges is the high sample complexity of deep RL methods, which remains problematic in real-world applications where data collection is expensive, difficult, or dangerous. Fortunately, many such applications offer prior knowledge that can be leveraged to reduce sample complexity. To effectively utilize this knowledge, it is crucial to move away from the tabula rasa approaches of neural networks and incorporate domain-specific prior knowledge into the learning process (Jonschkowski & Brock, 2015; De Bruin et al., 2018; Katt et al., 2022).

Another critical challenge is exploration, which is strongly tied to sample complexity. Effective exploration of unknown and interesting parts of the environment is essential in almost every application. A better exploration strategy means faster learning and improved sample efficiency. Importantly in RL, an agent must balance exploration (i.e. learning) with exploitation (i.e. maximizing reward). Most deep RL methods rely on heuristics for exploration, which can perform poorly in complex domains (Osband et al., 2016). These challenges are particularly pronounced in partially observable environments, where agents must make decisions based on limited information.

Model-based Bayesian RL (BRL) (Ross et al., 2011) offers a principled approach to addressing the exploration-exploitation trade-off by maintaining a belief over both the environment's state and dynamics. This belief enables the agent to balance the exploration–exploitation trade-off effectively. In this work, we build on the Factored Bayes-Adaptive POMDP (FBA-POMDP) framework (Katt et al., 2017; 2019), a model-based BRL approach that combines partial observability and structured factored models. FBA-POMDPs incorporate factorized representations of the environment's dynamics, allowing agents to exploit problem structure for improved scalability. Additionally, thanks to its Bayesian nature, prior knowledge can be incorporated into FBA-POMDPs in a principled way via Bayesian priors, further improving sample efficiency.

Despite its advantages, the FBA-POMDP framework faces significant challenges. While factorization enables better generalization, the inclusion of irrelevant state factors in the model can lead to unnecessarily large model spaces. This increases computational demands during planning and reduces statistical strength by hypothesizing dependencies that are irrelevant to maximizing rewards. For instance, in a cluttered environment, an agent may only need to consider the positions of objects to navigate effectively, while features such as their colors or shapes are irrelevant for the reward. Abstracting away such unnecessary details can simplify the model space, improve computational efficiency, and enhance learning performance. Previous studies have demonstrated that even lossy abstractions can improve performance in planning (Chitnis et al., 2020; He et al., 2020). This is because simplified models can generate more simulations within a fixed runtime, potentially leading to better results in sampling-based online planning. Motivated by this insight, we propose incorporating abstraction into the FBA-POMDP framework to improve scalability and learning efficiency.

In this work, we explore the application of abstraction within FBA-POMDPs to enhance planning efficiency, scalability, and learning performance. To achieve this, we augment Factored BA-POMCP (FBA-POMCP) (Katt et al., 2017; 2019), an established online planning algorithm for FBA-POMDPs, by incorporating multiple levels of abstraction. Our method creates abstractions automatically based on the problem's structure, enabling agents to plan and learn more effectively. This represents a novel step toward combining abstraction with BRL in Factored POMDPs (F-POMDPs). Empirically, we demonstrate that abstraction improves performance in two critical ways: (1) by reducing model size, allowing for more simulations within a given computation time, and (2) by simplifying the learning problem, leading to faster learning and improved performance in fewer episodes. These findings highlight the potential of abstraction to address key challenges in BRL for F-POMDPs and open a promising direction for future research into leveraging abstractions to improve scalability, exploration, and decision-making in complex, real-world environments.

2 Background

In this section, we introduce the rich body of literature that our work builds upon. In particular, section 2.1 introduces the POMDP as the general mathematical model for sequential decision-making. We then describe (model-based) Bayesian reinforcement learning in factored POMDPs in section 2.2, which will turn out to be a (belief-space) Partially-observable Markov decision process itself. Lastly, we will discuss algorithmic approaches for solving these decision problems in section 2.3.

2.1 POMDPs and Factorization

Sequential decision-making in stochastic domains with hidden states can be formalized as a Partially Observable Markov Decision Process (POMDP) (Boutilier & Poole, 1996; Kaelbling et al., 1998). The POMDP is defined by the tuple $(\mathbb{S}, \mathbb{A}, \mathbb{O}, \mathcal{D}, \mathcal{R}, \gamma, H)$, where $\mathbb{S}$, $\mathbb{A}$, and $\mathbb{O}$ are the (discrete) set of states, actions, and observations, respectively. The dynamics $\mathcal{D}$ specify the system's transition probabilities $\mathcal{D} \in \mathbb{D}$: $(\mathbb{S} \times \mathbb{A}) \to \Delta(\mathbb{S} \times \mathbb{O})$, and $\mathcal{R}$: $(\mathbb{S} \times \mathbb{A} \times \mathbb{S}) \to \mathbb{R}$ is the reward function. The (maximum) number of time steps in an episode is the horizon $H \in \mathbb{Z}$, and $\gamma \in [0, 1]$ is the discount factor.

The goal of the agent is to maximize the discounted return, $\sum_t \gamma^t r_t$. To do so, it can use the observable action ($a \in \mathbb{A}$) and observation ($o \in \mathbb{O}$) history $h_t = (a_0, o_1, \ldots, a_{t-1}, o_t)$, or it can use the belief as a sufficient statistic. The belief is the probability distribution over the current state $b \in \mathbb{B}$: $\Delta\mathbb{S}$, which can

be updated with Bayes' rule: $b'(s') = \tau(b, a, o)(s') \propto \sum_s \mathcal{D}(s', o|s, a)b(s)$. However, in most problems, the computation of the belief update is intractable. Section 2.3 will cover how to find a solution.

Running Example As a simple intuitive example, consider the Corridor domain shown in fig. 1a. The agent starts at the "Start" location and its goal is to reach the "Reward" location. As depicted in the figure, there is also a "Boots" location and a "Button" location. To reach the reward the "Door" must be open. There is also always a "Person" present in the environment. There are 8 different persons, with exactly one present during each episode. The probability of the person opening the door at some point during an episode varies depending on which person is present, but this probability is generally very low, ranging from about 1.25% to 10%. The agent cannot interact with the person.

The state consists of the location of the agent, the person present and the binary statuses of the boots, button, and door. The agent has four actions: move *left* or *right*, *put on boots*, *push button*, and *lock pick* the door. Moving left or right succeeds 30% of the time without boots and 95% of the time with boots. The probability of success for both *put on boots* and *push button* is 90%, provided the agent is in the correct location. When the button is pressed, there is a 100% chance that the door will open. The *lock pick* action only has an effect when the agent is standing next to the door and has a 40% chance of success. The agent can also open the door by "bashing" into it, specifically by using the move right action to collide with the door. This method is not very effective and has only a 5% chance of opening the door. The state is not fully observable to the agent; instead, it receives a noisy observation, which will be explained later.

Factorization The dynamics of POMDPs can often be captured efficiently through factorization and graphical models, such as Dynamic Bayes networks (DBNs) (Boutilier et al., 1999; Murphy, 2002). Graphical models represent random variables by their features (also referred to as factors) and provide the ability to capture independence between these features. The advantage is that this independence allows for generalization and efficiency: fewer dependencies lead to fewer parameters. For instance, in our running example, the conditional probability of the *Door* opening given $X = 2$ and $Door = \text{closed}$ (i.e., $P(Door = \text{open}|X = 2, Door = \text{closed})$) can be estimated much more reliably than the conditional probability that involves additional factors, such as $P(Door = \text{open}|X = 2, Door = \text{closed}, \text{Button} = \text{off})$, because the former occurs more frequently. Here we introduce the (dynamic) Bayes network (BN) as a graphical model.

Formally, a BN is defined by a topology — the structure over the nodes in the graph — and conditional probability tables (CPTs) that describe the conditional probability distribution of each node. The topology $G \in \mathbb{G}$ describes the structure of the graph and denotes for each possible edge (between nodes) whether there is a dependency, where $\mathbb{G}$ is the set of all possible edge configurations. These directed edges define the *parents* $Pa(x'_i; G)$ of each node x'_i as the set of incoming nodes (where we typically drop the dependency on the topology G in the notation). The CPTs $\theta \in \Theta$ govern the conditional probability distribution of an output node x'_i given input nodes x : $p(x'_i|x) = p(x'_i|Pa(x'; G); \theta_i)$. In discrete environments, for example, these typically are categorical distributions; one for each parent value combination for each node. The *Dynamic* BN (DBN) restricts the space of graph (topologies) such that the nodes are divided into *input* and *output* sets, and allow only for directed edges from the input to the output set. This is convenient for (Markovian) dynamic systems, where random variables change over time.

The Factored POMDP (F-POMDP) factorizes the state and observation space into nodes and describes the dynamics with a DBN for each action (Boutilier & Poole, 1996). The state and observation factors make up the DBN and their probabilities are represented with CPTs, a probability distribution for each unique set of *parent values* for a given action. The incoming edges of a factor encode the dependencies (parents) of that factor.

For example, fig. 1b shows the DBN of the running example for the action move *right*.[1] In the running example, the state space is factored into the x position and the binary factors *Boots*, *Button*, and *Door*. The factors x, *Boots*, and *Door* influence the movement of the agent, as can be seen by the incoming edges of x. In fig. 1, we do not explicitly show the observation factors, but in this problem, every state factor has a corresponding observation factor, which provides a noisy representation of the state factor. There is a 70%

[1]For simplicity we have omitted the observation factors in fig. 1 and instead describe the observation function in the text.

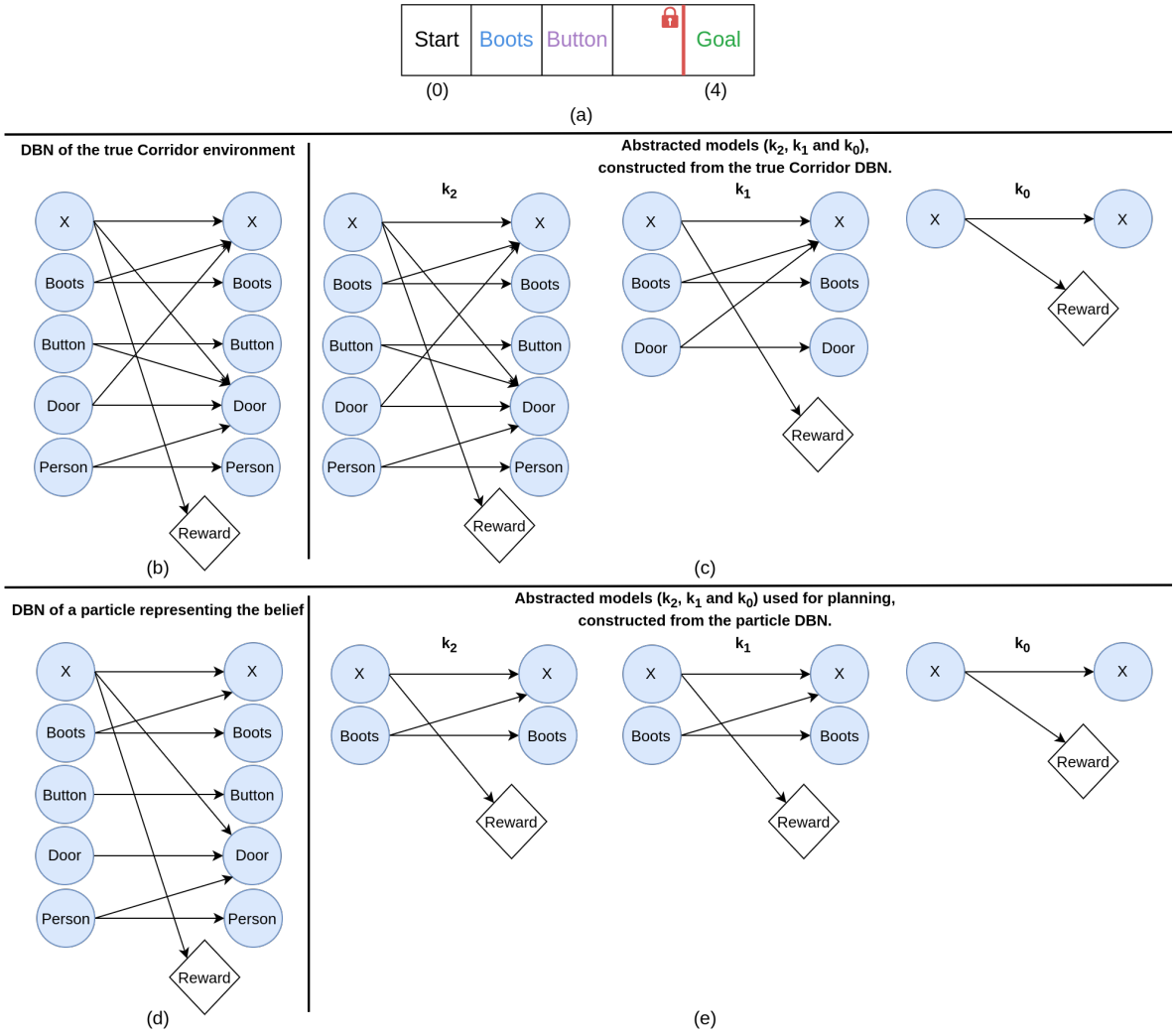


Figure 1: a) The Corridor domain. b) ground truth graph representing the dynamics of the Corridor domain for the action *right*. All the factors are partially observable, the agent gets an independent but noisy observation of each state factor. c) The abstract models for different levels of abstraction (denoted by k_0, k_1, k_2), constructed from the true Corridor DBN for the action *right*. d) We use particle filters to represent the belief, each particle contains a DBN of the corridor, this is an example of such a DBN, and e) the abstract models constructed from this example DBN.

chance of correctly observing the *Person*, while for each other factor, there is a 90% probability of correct observation.

Bayesian inference over DBNs This work considers the problem of learning the (DBN) dynamics of the environment. When the parameters of a model are not given, the Bayesian approach is to assume a prior (to compute posteriors) instead. The DBN is defined by — and thus a prior must describe a probability distribution over — its *topology* $G \in \mathbb{G}$ and CPTs $\theta \in \Theta$.

Regarding the topology, we specify a prior over what state factors form the set of parents for each state and observation factor in the DBN. In the running example, for instance, the prior could assign a probability greater than 0 to a structure where the factor *Door* does not influence the x position and where the factor *Button* does not influence the door, as shown in fig. 1d. The CPTs are categorical distributions and, hence, the Dirichlet distribution is a natural prior (Frigyik et al., 2010). Dirichlet distributions are parameterized

by a collection of conditional count tables (CCCT), with one conditional count table (CCT) $\chi \in X$ for each unique set of parent values for each node.

The initial counts are specified by the prior, and the agent learns by updating the counts of the CCCT. In particular, given a topology G, prior CCCT and a new data point (x, x'), the Bayesian posterior is computed by incrementing the count $\chi_{x'_i, x[Pa(x'_i)]}$ of each node x'_i that is associated with its parent's values $x[Pa(x')]$. We denote the operation for incrementing the counts of factored POMDP DBN CCTs as $\mathcal{U}$: $(X \times \mathbb{S} \times \mathbb{A} \times \mathbb{S} \times \mathbb{O}) \to X$. Note that there is no closed-form solution to the posterior over topologies.

2.2 Factored BA-POMDPs

If the state transitions were *not hidden*, one could simply maintain a set of the counts CCTs associated with each transition and over time converge to the true dynamics. This is the case under full observability (MDPs), and is called the Bayes-Adaptive MDP (BA-MDP) (Duff, 2002). Unfortunately, this is not the case in partially observable environments[2], and hence, there is uncertainty over these counts χ.

The Factored Bayes-Adaptive POMDP (FBA-POMDP) (Katt et al., 2019) captures this uncertainty by using the POMDP formalism. In particular, this Bayes-adaptive model *is* a POMDP whose state space consists of both the state and the dynamics of the original POMDP (Ross et al., 2011). Formally, we denote the FBA-POMDP with $(\dot{\mathbb{S}}, \mathbb{A}, \mathbb{O}, \dot{\mathcal{D}}, \dot{\mathcal{R}}, \gamma, H)$, where $\dot{\mathbb{S}}$ is the augmented state space: $\dot{\mathbb{S}} = \mathbb{S} \times \mathbb{G} \times X$. I.e., each (hyper-)state $\dot{s} \in \dot{\mathbb{S}}$ contains a domain state s, a topology G, and a CCCT χ: $\dot{s} = \langle s, G, \chi \rangle$. The action and observation spaces, the horizon, and the discount factor are taken directly from the original POMDP. Similarly, the reward function relies on the underlying system: $\dot{\mathcal{R}}(\dot{s}, a, \dot{s}') = \mathcal{R}(s, a, s')$. The dynamics $\dot{\mathcal{D}}$ dictate how augmented states transition:

$$\dot{\mathcal{D}} = p(s', o, G', \chi' | s, a, G, \chi) \tag{1}$$

$$= p(s', o | s, a; G, \chi) \mathbb{1}_G(G') \mathbb{1}_{\chi'}(\mathcal{U}(\chi, s, a, s', o)), \tag{2}$$

where $p(s', o | s, a; G, \chi)$ can be written as:

$$p(s', o | s, a; G, \chi) = p(s' | s, a; G, \chi) p(o | a, s'; G, \chi). \tag{3}$$

In equation 2 the term $p(s', o | s, a; G, \chi)$ shows that the model (G, χ) in state $\dot{s}$ determines the probabilities of the next state s' and observation o. The $\mathbb{1}.(\cdot)$ is the indicator function, e.g., $\mathbb{1}_G(G')$ is non-zero (one) if-and-only-if the "next topology" G' equals the previous: $G' = G$. Similarly, the only possible CCCT χ' is the one that results from updating the corresponding parameters through $\mathcal{U}$. Since the POMDP is fully specified, the original learning problem is cast to a planning problem *with known* dynamics, given a prior $p_{\mathcal{D}}$. Most importantly, the exact solution to this planning problem yields the optimal policy, in terms of exploration-exploitation, with respect to the prior (Ross et al., 2011). Now we can apply our standard POMDP planning tools (e.g., particle filtering, planning) to FBA-POMDPs, as anytime solvers provide good approximations which converge to the exact solution in the limit of infinite compute (Silver & Veness, 2010; Katt et al., 2017; 2019).

2.3 Solving FBA-POMDPs

Unfortunately, FBA-POMDPs are very large, and naive applications of planning techniques will fail. We give a high-level description of how solutions for FBA-POMDPs can be found, and refer to the original work (Katt et al., 2017; 2019) for details. Just like in any other POMDP, a planning solution requires two components: belief tracking and action selection.[3]

Belief tracking in the FBA-POMDP The belief, the posterior over the current state, is a probability distribution over the POMDP state and its distribution $b \in \Delta(\mathbb{S} \times \mathbb{D})$ given the observed history $h_t =$

[2]In an MDP we see the whole transition (s,a,s'), so we can update the count $\chi(s, a, s')$. In contrast, both s and s' are hidden in the partially observable case. So we have to update $\chi(s, a, s')$ based on our belief resulting from the action-observation history rather than the real transitions.

[3]The original POMCP (Silver & Veness, 2010) implementation combined these steps to some extent, but we separate them out.

Algorithm 1 Sequential Importance Sampling

Input: $\{\dot{s}_i, w_i\}_{i=0}^n$: current (weighted) filter
a, o: action and observation
for $i \in 0, \ldots, n$ **do**
  $s'_i \sim p(\cdot|s_i, a; G_i, \chi_i)$
  $w'_i \leftarrow w_i \times p(o|s'_i, a; G_i, \chi_i)$
  $\chi'_i \leftarrow \mathcal{U}(\chi_i, s_i, a, s'_i, o)$
end for
return $\{s'_i, G_i, \chi'_i, w'_i\}_{i=0}^n$ // Normalize & re-sample

Algorithm 2 Initialize Particle Filter

Input: $p_{\dot{s}_0}$: prior over initial state
n: number of desired particles
for $i \in 0, \ldots, n$ **do**
  $\langle s_i, G_i, \chi_i \rangle \sim p_{\dot{s}_0}$
  $w_i \leftarrow \frac{1}{n}$
end for
return $\{s_i, G_i, \chi_i, w_i\}_{i=0}^n$

$(a_0, o_1, \ldots, a_{t-1}, o_t)$. Unfortunately, the computation of the belief update is intractable in most problems. Thus, the belief is often approximated with particles instead (Thrun, 1999). A particle filter represents a distribution through *particles*, which in this case represent FBA-POMDP states, specifically each particle is a weighted FBA-POMDP state (w, s, G, χ) with (unnormalized) weight $w \in \mathbb{R}^+$.

There are numerous sampling mechanisms for updating the belief given a new action-observation pair (Thrun, 1999). Here, we focus on sequential importance sampling (SIS). SIS consists of two operations: propagation and re-weighting (see algorithm 1). First, the *proposal distribution* propagates a particle by sampling its next value from the FBA-POMDP transition function $p(\dot{s}'|\dot{s}, a)$. Then, the likelihood of the particle generating the received observation $p(o|\dot{s}, a, \dot{s}')$ is used to re-weight the particle (recall equation 3). The initial belief (particle filter) is initialized by sampling from the priors over the POMDP state and the DBNs describing the dynamics (see algorithm 2).

Specific to the FBA-POMDP, the belief over the topologies can deteriorate: the number of unique graph structures is determined (and limited) by the initial particle filter, as topologies do not get updated during the belief updates. When that happens, Katt et al. (2019) re-invigorate the belief with a Metropolis-Hastings-within-Gibbs sampling procedure (Murphy, 2012).

Action selection in the FBA-POMDP Even with approximated belief updates, the belief space can be very large, especially in high-dimensional problems. Thus it is often infeasible to compute the action that maximizes the discounted return for every possible belief the agent could end up in. As a result, literature addresses this issue by turning to *online* planners, such as Monte-Carlo tree search (MCTS) (Coulom, 2006) and Partially Observable Monte-Carlo Planning (POMCP) (Silver & Veness, 2010), because they can quickly provide good approximations. Importantly, these methods are anytime algorithms, meaning they can be stopped at any time and still provide a solution that improves with more computation. We use an extension of the POMCP algorithm, called Factored BA-POMCP (FBA-POMCP) (Katt et al., 2019), to pick actions. Like any MCTS method, this method incrementally builds a look-ahead tree of simulated interactions in the (FBA-POMDP) environment, as shown in algorithm 8. Each simulation receives as input a sampled hyper-state $(s, G, \chi) \sim b$, with *root sampled expected* CCCT parameters $\theta = \mathbb{E}[\chi]$, and the history of actions and observations. The simulation traverses to a leaf by picking actions according to UCB (Auer et al., 2002) and, unique to this particular algorithm, observations according to the CCCT θ. The tree is expanded and values are back-propagated as usually done in MCTS. The FBA-POMCP and simulate algorithms are shown in appendix A. For more details, see (Browne et al., 2012) for a survey on MCTS, (Silver & Veness, 2010) for MCTS in POMDPs, and (Katt et al., 2017; 2019) for MCTS in Bayes-Adaptive POMDPs (BA-POMDPs).

3 Abstraction for FBA-POMCP

FBA-POMCP is able to learn and exploit the structure in POMDPs in a Bayesian way. However, it struggles when the number of factors grows large. On the one hand, the presence of many state factors itself slows down sampling, potentially leading to insufficient simulations to derive adequate actions. On the other hand, the prior belief over models with many state factors will typically have low probability for models in which

all factors have a small number of parents. As a consequence, the particle filter typically will contain models that have at least a few factors with many parents. This leads to slow learning (low statistical strength) and possibly to exploration of factors with little or no effect on the rewards. These issues are problematic because our primary focus is on the task performance, rather than on learning the correct model itself.

A natural idea, therefore, is to explore in how far abstraction can address these two issues. While abstractions could lead to inaccurate models, in the regular (non-Bayes adaptive) planning case, it has been demonstrated that abstracting away factors with a weak influence can still improve the performance of online planning (He et al., 2020). Further, without abstraction, the agent could waste time exploring the dynamics of factors with little impact on the performance, if they are falsely believed to be influential. By removing these factors, abstraction can reduce unnecessary exploration and focus on the factors relevant for performance. As such, we propose to explore the impact that abstraction of state factors can have when learning in partially observable settings, formalized as FBA-POMDPs.

Specifically, we propose to perform the POMCP simulations with an abstracted FBA-POMDP model. We hypothesize that such abstraction can improve performance by 1) increasing the number of simulations that can be done thus improving performance in online planning, and 2) reducing unnecessary exploration of factors with little impact on the performance, and allowing to focus exploration on the relevant factors.

We cover the combination of abstraction with FBA-POMCP in four parts. First, we give a high-level overview of the abstraction method and how it is added to FBA-POMCP. Second, we define abstractions on different levels, denoted by $k_0, k_1, ...$, where k_0 represents the coarsest abstraction. We define these through *subsets of state factors* and show how to generate a subset of state factors from a graph structure for a particular level of abstraction. Third, we show how to use the subset of state factors to construct the abstract structure and counts. Finally, we provide theoretical support for the combination of FBA-POMCP with abstraction.

3.1 Adding Abstraction to FBA-POMCP

We propose a method to enable the FBA-POMDP framework to benefit from abstraction. Specifically, we will use abstract models for online planning with a variant of FBA-POMCP. To operationalize this, we will cover the following steps:

1. We expand the representations to include abstract states.

2. While we will not exploit abstraction in the belief update, the abstracted belief state still needs to be updated. We cover the necessary modifications to the belief update process.

3. Finally, we explain how FBA-POMCP can use abstracted states.

Expanding the Belief Representation When initializing the weighted particle filter in FBA-POMCP, each particle is a hyper-state $\dot{s} = \langle s, G, \chi \rangle$ associated with a weight w. The hyper-state $\dot{s}$ contains a ground state s, a graph structure G, and a set of counts χ. For the initialization we require a probability distribution over the possible starting states, over the possible structures, and a probability distribution over the counts given a structure. For the running example, fig. 1d shows a possible structure of a hyper-state, in this case the factor *Door* is not believed to influence the x factor, and the factor *Button* is not believed to influence the *Door* factor. When combining FBA-POMCP with abstraction, we abstract G and χ and add the resulting abstracted structure $\bar{G}$ and counts $\bar{\chi}$ to each hyper-state $\dot{s}$. This leads to an abstract hyper-state: $\bar{s} = \langle \dot{s}, \bar{G}, \bar{\chi} \rangle$. The particle filter thus stores both the original hyper-state $\dot{s}$ and the abstracted structure $\bar{G}$ and counts $\bar{\chi}$. The construction of the abstract hyper-states is done during the initialization of the particle filter, as shown in algorithm 3.

For our running example, fig. 1e illustrates the structure $\bar{G}$ for different levels of abstraction, corresponding to the original structure G in fig. 1d. In algorithm 3, the function Abstract creates the abstract model from a hyper-state $\dot{s}$, as detailed in algorithm 4. The methods for selecting a subset based on the level of abstraction k and for creating the abstract $\bar{G}$ and $\bar{\chi}$ will be elaborated upon in the following sections.

Algorithm 3 Initialize Abstract Particle Filter

Input: $p_{\dot{s}_0}$: prior over initial state
n: number of desired particles
k: abstraction level
for $i \in 0, \ldots, n$ **do**
  $\langle s_i, G_i, \chi_i \rangle \sim p_{\dot{s}_0}$
  $\langle \bar{s}_i, \bar{G}_i, \bar{\chi}_i \rangle \leftarrow \text{Abstract}(k, \langle s_i, G_i, \chi_i \rangle)$
  $w_i \leftarrow \frac{1}{n}$
end for
return $\{\bar{s}_i, \bar{G}_i, \bar{\chi}_i, w_i\}_{i=0}^{n}$

Algorithm 4 Abstract

Input: k: abstraction level
$\dot{s} = \langle s, G, \chi \rangle$: hyper-state
$Q \leftarrow \text{GetSubsetK}(k, G)$
$\bar{G} \leftarrow G$
$\bar{\chi} \leftarrow \chi$
for $(q, a) \in Q \times A$ **do**
  $\bar{G}, \bar{\chi} \leftarrow q.\text{MarginalizeCounts}(\bar{G}, \bar{\chi}, Q, a)$
end for
for $x \in G - Q$ **do**
  $\bar{G}.\text{remove}(x)$ // Remove factor
  $\bar{\chi}.\text{remove}(x)$ // Remove factor
end for
return $\langle \dot{s}, \bar{G}, \bar{\chi} \rangle$

The Belief Update Process To ensure consistency of the belief, we use the full model (G and χ) during the belief update (Russell & Norvig, 2016; Katt et al., 2019), as described in sections 2.2 and 2.3. This means that the belief update process remains largely the same. The main difference is that we now track and update both the full and abstract models, as shown in algorithm 5. As in section 2.2, the graph structures of both the full and abstract models stay the same during the update. The counts χ are updated via $\mathcal{U}$. Since each state maps to exactly one abstract state, the abstract counts $\bar{\chi}$ can be updated through the update of the counts χ. Essentially, the (abstract) graphs remain unchanged, while the (abstract) counts are updated.

Using Abstracted States in FBA-POMCP The planning process exclusively uses abstract models. Specifically, in Algorithm 8, the abstract representation is used to perform the environment *Step* function, which is utilized during simulations and roll-outs in the look-ahead tree search. The *Step* function uses the hyper-state with the abstract model and an action to sample a next abstract state by iteratively sampling the factors. It is shown in Appendix A.

The benefit of using an abstract model is that it speeds up planning since it contains fewer state factors, allowing for fasting sampling of the next state. When there is limited time for planning, this is one way in which abstraction can improve performance. It is important to note that the abstract models are constructed at the beginning of the agent's lifetime, when its belief is initialized. As a result, these abstract models are always available.

3.2 Abstraction Via Subsets of State Factors

We introduce a method for performing abstraction in the BRL context. Since we are interested in understanding how abstraction impacts the learning process, we base our approach on a relatively simple planning method for factored Markov decision processs (MDPs) (Dearden & Boutilier, 1997), which is easy to understand and analyze. We extend it to 1) make it applicable to partially observable problems, and 2) handle the counts required in BRL. Our approach introduces a level of abstraction that determines the factors to include based on the graph structure.

We define different levels of abstraction based on their connection to the reward in the graph structure. Each abstraction level is defined by a set of state factors that is included in the model, observation factors are always kept in the model. After abstraction, this can lead to observation factors without parents, we explain how we deal with this in section 3.3.4. We start building abstractions from the factors directly influencing the reward, the immediately relevant factors (IR). We first give a formal definition and then illustrate it with an example.

Algorithm 5 SIS with Abstraction

Input: $\{\dot{s}, \bar{G}, \bar{\chi}, w\}_{i=0}^{n}$: current (weighted) filter
a, o: action and observation
for $i \in 0, \ldots, n$ **do**
  $s'_i \sim p(\cdot|s_i, a; G_i, \chi_i)$
  $w'_i \leftarrow w_i \times p(o|s'_i, a; G_i, \chi_i)$
  $\chi'_i \leftarrow \mathcal{U}(\chi_i, s_i, a, s'_i, o)$
  $\bar{\chi}'_i \leftarrow \bar{\mathcal{U}}(\bar{\chi}_i, s_i, a, s'_i, o)$
end for
// Normalize & re-sample
return $\{s'_i, G_i, \chi'_i, \bar{G}_i, \bar{\chi}'_i w'_i\}_{i=0}^{n}$

Algorithm 6 GetSubsetK

Input: k: abstraction level
G: Graph structure.
$Q \leftarrow$ GetMinimumSet() // k_0, the IR factors
if $k == 0$ **then**
  return Q
end if
$Q' \leftarrow Q$
for L = 1; L $\leq$ k; L++ **do**
  for $(q, a) \in Q \times A$ **do**
    $Q' \leftarrow Q' \cup G$.getNode(q, a).parents()
  end for
  $Q \leftarrow Q'$
end for
return Q

Definition 1. *The set of immediately relevant factors (IR) contains only the factors $q \in G$ that directly influence the reward. Specifically, these are the factors that are parents of the reward, denoted as $Pa(\text{reward})$. The smallest subset of state factors k_0 is equal to IR, $k_0 = IR$. The set k_i is the smallest set such that the following holds:*

1. *$k_{i-1} \subseteq k_i$.*
2. *If $q \in k_{i-1}$ then $Pa(q) \in k_i$.*

The set k_{inf} refers to the full model.

In the running example, we see an example of a structure in fig. 1d. In this problem, the agent only receives a reward when it is in the goal location, i.e., $x = 4$. This is reflected in the graph structure where the only parent factor of the reward is x. In this case, x is the only IR factor and the abstraction k_0 only contains x as shown in fig. 1e. To construct the subset of state factors for k_n, we add the parents of the factors in k_{n-1}. So to see which factors to include in k_1 in this example, we check in fig. 1d which factors are parents of x. In this case, that is only *Boots*. Finally, for k_2, no new factors are added since in the structure in fig. 1d the parents of *Boots* do not include any factors not yet in the set of k_1.

The procedure to get the subset of state factors for a given level of abstraction k and a particle (or hyper-state) $\dot{s}$ is shown in algorithm 6. First, it initializes a set Q with the set IR, retrieved with *GetMinimumSet*. For abstraction level k_0, this is what is returned immediately. For higher levels, it then builds the subset incrementally by adding the parents of the factors in Q. That is, to construct the subset of state factors k_n, it starts with k_{n-1} and then adds the parents of these factors.

When the reward function is known, the function GetMinimumSubset directly returns the set IR. When the reward function is unknown, the reward itself is also modeled as a state factor that takes the same value as the reward. Uncertainty about the reward function can then be incorporated in the belief, and hyper-states may end up with different graph structures for the reward. The IR can then be retrieved by finding the parents of the reward state factor. We demonstrate that our method can deal with uncertainty about the reward and IR in section 4.4.

3.3 Abstract Model Construction

After retrieving a subset of state factors, we have to construct the abstract model. Since the abstraction uses only a subset of the state factors, this involves removing factors, and therefore we need to decide how to treat factors that will have missing parents as a result of this. To illustrate, when we abstract a full model

such as the one in fig. 1d to create the abstract model on level k_0 (fig. 1e), we remove *Boots* and create a distribution for x with only x itself as the parent factor. The question then is how we can define a CCT that does not depend on 'Boots' from the original one that does.

For probability distributions in known models it is logical to resort to marginalization. However, in section 3.3.1 we show that the problem is more deeply rooted: marginalization leaves us with a term that is difficult to specify since it depends not only on the values of its parents but also on the policy, and it can change over time. As such there is no fundamentally right approach to do this form of abstraction. Instead, novel ideas and approximate approaches are needed. We explore 2 first ideas. In section 3.3.2 we make an assumption on the abstraction and show that we can simply aggregate the counts in that case. In section 3.3.3, we motivate using approximate abstraction and discuss potential issues that arise with the approximation.

3.3.1 For Probability Distributions

Before considering the case of CCTs, we treat the case of probability distributions. For a probability distribution, given a factor X with a set of parents Parents(X), we can marginalize out a parent Y or a set of parents. For ease of notation, we show the marginalization for one parent, multiple parents can be removed by repeating this:

$$P(X|\text{Parents}(X) \setminus Y) = \sum_Y P(X, Y|\text{Parents}(X) \setminus Y) \tag{4}$$

$$= \sum_Y P(X|\text{Parents}(X) \setminus Y, Y) P(Y|\text{Parents}(X) \setminus Y). \tag{5}$$

The term $P(Y|\text{Parents}(X) \setminus Y)$ gives a weight to the contribution of $P(X|\text{Parents}(X) \setminus Y, Y)$ to $P(X|\text{Parents}(X) \setminus Y)$. However, estimating this term for a DBN is nontrivial. In the running example, consider the probability of moving from $x = 1$ to $x' = 2$ after taking the action *right*. The sampled model in fig. 1d only has the factors x and *Boots* as parents of x, and the abstract model k_0 (fig. 1e) does not include *Boots*. Following equation 5, we can obtain the marginal distribution for x by summing over the separate values of *Boots*:

$$P^{right}(x'|x) = \sum_{b \in Boots} P^{right}(x'|x, b) P(b|x). \tag{6}$$

However, while the probabilities $P^{right}(x'|x, b)$ are well defined (e.g., $P^{right}(x' = 2|1, b = \text{On}) = 0.95$), the value of $P(b|x)$ is not as clear. This probability represents the likelihood that the boots are on given a specific location x. However, it does not depend solely on x itself. For instance, we might know that the probability of the boots being on is 0 at the start of the episode, but this probability generally depends on the history of actions and observations. In general, we can make the following observation:

Observation 1. *Accurately estimating $P(Y|Parents(X) \setminus Y)$ without additional information is generally not possible. This is because Y can depend on other variables, including itself, and on the policy that can change over time.*

The view of $P(Y|\text{Parents}(X) \setminus Y)$ as a weight in equation 5 is related to the concept of a weighting function in work on state abstraction (Dearden & Boutilier, 1997; Li et al., 2006). Theoretical work shows that, for some abstractions, a policy based on the abstract model (with any weighting function) can perform well in the real problem in planning (Li et al., 2006; Abel et al., 2016; Congeduti & Oliehoek, 2022) and in RL (Starre et al., 2023). For probability distributions, Dearden & Boutilier (1997) use a sort of average of the probabilities but also remark this can lead to suboptimal solutions. The best way to approach estimating $P(Y|\text{Parents}(X) \setminus Y)$ for the optimal performance is still an open problem.

3.3.2 Beliefs and Aggregating Counts for Exact Abstractions

In the previous section we discussed the problem of dealing with conditional probability tables (CPTs) where parents are abstracted, leading to dependence on the policy and history for estimating $P(Y|\text{Parents}(X) \setminus Y)$

in equation 5. In this section we will first make an assumption which allows us to get around estimating of $P(Y|\text{Parents}(X) \setminus Y)$ and then switch to the case of the CCCT under this assumption. In the next section we will discuss what happens when we do not make this assumption.

One situation where abstraction makes sense is when the model is overspecified; it contains links that are unnecessary. This is the case when the abstract model probabilistically behaves in the same way as the full model, which is what we assume here:

Assumption 1. *The abstraction is exact. That is, let Z be the set of removed parents for a factor X, then we have*

$$\forall_{Y \in Z}, \forall_{y_1, y_2 \in Y} : P(X|Parents(X) \setminus Y) = P(X|Parents(X) \setminus Y, y_1) \tag{7}$$
$$= P(X|Parents(X) \setminus Y, y_2). \tag{8}$$

This assumption implies that the links between the removed parents and the corresponding child node were obsolete. For instance, consider a change in the running example where the boots would have no effect on x, then this would mathematically mean that $P(x'|x, \text{Boots} = \text{On}) = P(x'|x, \text{Boots} = \text{Off})$. We observe:

Observation 2. *Under assumption 1, $P(Y|Parents(X) \setminus Y)$ has no influence. That is, since*

$$P(X|Parents(X) \setminus Y) = \sum_Y P(X|Parents(X) \setminus Y, Y) P(Y|Parents(X) \setminus Y) \qquad (equation\ 5) \tag{9}$$
$$= \forall_{Y \in Z}, \forall_{y \in Y} : P(X|Parents(X) \setminus Y, y). \tag{10}$$

Thus, if boots had no influence, data collected with boots on and off can be used to estimate $P(x'|x)$. For example, consider the conditional counts $\chi(x'|x = 1, \text{Boots})$ in table 1.

Observation 2 means that, to marginalize in the CCCT, we no longer have to be concerned about $P(Y|\text{Parents}(X) \setminus Y)$. Which means that in table 1 we can aggregate the counts in the columns, formally:

$$\bar{\chi}(X|\text{Parents}(X) \setminus Y) = \sum_Y \chi(X|\text{Parents}(X) \setminus Y, Y). \tag{11}$$

For example, to determine the conditional counts in table 1, we apply equation 11 to construct the abstracted

Table 1: Initial conditional count table, for $x = 1$ and action *right*.

Boots	$x' = 1$	$x' = 2$
On	2	8
Off	6	4

Table 2: Count table after aggregation, for $x = 1$ and action *right*.

$x' = 1$	$x' = 2$
2+6 = 8	8+4 = 12

Table 3: Count table after aggregation and normalization, for $x = 1$ and action *right*.

$x' = 1$	$x' = 2$
$1/2 * 8 = 4$	$1/2 * 12 = 6$

(or marginalized) counts $\bar{\chi}$ from the original counts χ. This is done for every action by aggregating the counts as follows:

$$\bar{\chi}^{right}(x'|x) = \sum_{b \in Boots} \chi^{right}(x'|x, b). \tag{12}$$

Writing out equation 12 we obtain the counts $\bar{\chi}^{right}(x'|x = 1)$ after aggregation:

$$\bar{\chi}^{right}(x' = 1|x = 1) = \sum_{b \in Boots} \chi^{right}(x' = 1|x = 1, b) = 2 + 6 = 8, \tag{13}$$

$$\text{and } \bar{\chi}^{right}(x' = 2|x = 1) = \sum_{b \in Boots} \chi^{right}(x' = 2|x = 1, b) = 8 + 4 = 12. \tag{14}$$

The resulting conditional counts are shown in table 2. Note that now the resulting row has counts $(\{8, 12\})$ which are higher than the individual previous rows for Boots *On* $(\{2, 8\})$ and *Off* $(\{6, 4\})$. This implies that after abstraction we are (relatively) more confident about these transitions than before. Under assumption 1 this does not have a large influence when the prior is close to the true distribution, as in that case these estimates should be close together. However, this could be different when the abstraction is not exact or if the prior is not close to the true distribution.

3.3.3 Aggregating Counts for Approximate Abstractions

Previously, we assumed that the abstraction was exact. Of course, this may not always be the case, or it may not be necessary to make this assumption. There exist scenarios where one could argue for the use of *approximate* abstractions (Dearden & Boutilier, 1997; Abel et al., 2016; Starre et al., 2023). For example, in cases where a parent only has a small influence, abstracting these parents away can lead to faster learning (Starre et al., 2023). Additionally, reducing the size of the model through abstraction can enhance performance by facilitating faster planning (He et al., 2020).

However, when the abstraction is not exact, observation 2 no longer holds. Specifically, with an approximate abstraction, $P(X|\text{Parents}(X) \setminus Y, Y)$ will generally vary for different instantiations of Y. Consequently, $P(Y|\text{Parents}(X) \setminus Y)$ does influence the result. In this case, abstracting a candidate model could result in a probability distribution that deviates significantly from the behavior of the candidate model.

As an example, consider again the running example where with *Boots* = *On* we have a probability of moving of 95% and with *Boots* = *Off* only 30%. Table 1 shows are initial estimates where *Boots* is still included, counts of $\{2, 8\}$ and $\{6, 4\}$ for *Boots* = *On* and *Boots* = *Off*, respectively. These are reasonably accurate with, if we translate the counts to probabilities, an expected 80% and 40% chance of moving, respectively. However, when *Boots* is removed we see in table 2 this leads to counts of $\{8, 12\}$, or an expected probability of moving of 60%. With *Boots* being removed from the model the agent is highly likely to be in a state with *Boots* = *Off*, and thus this leads to an overestimation of the probability of moving for the agent.

As alluded to in the previous section, the abstraction also increases the confidence in the resulting counts. When the abstraction is not exact, we could say that the increased confidence in the resulting counts is not warranted, there is overconfidence. This overconfidence can slow down learning since it will take more experience to change the belief. For example, the change in table 1 of adding an extra observation to the row with *Boots* = *Off* has a relatively larger effect than adding one extra observation after aggregation in table 2.

This means that the proposed aggregation in equation 11 does not work as well when assumption 1 does not hold, since it can lead to incorrect estimations with a higher confidence. As such, we want to adapt the aggregation method. **It is still an open question what the best way to aggregate when using approximate abstraction.**

We propose a way to reduce the overconfidence in the resulting dynamics through a normalization scheme, which should lead to quicker learning in cases where the prior and abstraction are biased. Let $Z = Y_1, Y_2, \ldots, Y_n$ denote the set of removed parents. To normalize, we multiply each entry by

$$\frac{1}{\prod_{Y \in Z} |\text{dom}(Y)|}, \tag{15}$$

where $|\text{dom}(Y)|$ represents the number of values that the parent Y can take. For example, in the case of the position x from the running example, we have $|\text{dom}(x)| = 5$.

Using $\tilde{\chi}$ to represent normalized counts, applying the normalization factor equation 15 to equation 11 results in:

$$\tilde{\chi}(X|\text{Parents}(X) \setminus Z) = \frac{1}{\prod_{Y \in Z} |\text{dom}(Y)|} \sum_{(y_1, y_2, \ldots, y_n) \in Y_1 \times Y_2 \times \cdots \times Y_n} \chi(X|\text{Parents}(X) \setminus Z, y_1, y_2, \ldots, y_n). \tag{16}$$

Intuitively, the proposed normalization scheme reduces the counts proportionally to the amount of rows that is removed during aggregation. Applying equation 16 to the example where we remove *Boots* this leads to:

$$\tilde{\chi}^{right}(x'=1|x=1) = \frac{1}{|\text{dom}(Boots)|} \sum_{b \in Boots} \chi^{right}(x'=1|x=1,b) = \frac{1}{2}(2+6) = 4, \tag{17}$$

$$\text{and } \tilde{\chi}^{right}(x'=2|x=1) = \frac{1}{|\text{dom}(Boots)|} \sum_{b \in Boots} \chi^{right}(x'=2|x=1,b) = \frac{1}{2}(8+4) = 6, \tag{18}$$

also shown in table 3.

By applying the normalization of equation 15 the amount of counts in the table after aggregation is equal to the average amount of counts in the initial prior. For example, in table 1 the counts in the rows both sum up to 10, and the counts after the aggregation and normalization also sum up to 10 (table 3).

The proposed normalization provides a robustness against mistakes in the prior and approximate abstraction, by lowering the impact that the prior has on the learning. In our experiments, we investigate this normalization scheme, showing that this can significantly speed up learning.

3.3.4 The Observation Space for Abstract Models

Since we remove state factors from the model in the abstraction, a natural question is how we deal with the observation factors. We can consider two cases, 1) where a part of the parents is removed, and 2) when all the parents are removed.

In the first case, we simply perform the aggregation in the same way as for the state factors. It is the second state that poses a problem, as it leaves us with no parents for the observation factor. Since in this case the observation would provide no actual information about the underlying state, we enhance the observation space of the abstract model by including an observation "not observed" for all observation factors. For observation factors where all parents are removed the observation function will simply return "not observed".

3.4 Theoretical Support

Here we will show how the combination of FBA-POMCP with the abstraction method can lead to near-optimal performance when the abstraction is good. We first show that transforming the original FBA-POMDP with the abstraction method results in another FBA-POMDP. Because of this, the theoretical guarantees of FBA-POMCP apply to the abstracted problem. Then we give a definition for the quality of the abstraction that gives a guarantee on the performance in the original FBA-POMDP. Together this shows that abstractions leads to near-optimal solutions with FBA-POMCP, when a good abstraction is used.

First, we note that the abstraction results in another FBA-POMDP:

Lemma 1. *The result of applying the abstraction method, as described in algorithm 4 and section 3.3, to the original FBA-POMDP results in another FBA-POMDP, the abstract FBA-POMDP.*

In appendix B, we present a constructive proof. In essence, abstraction reduces the state space and marginalization produces a dynamics function for the resulting state space. Lemma 1 implies that we can use POMCP to find a near-optimal solution with respect to the belief in the abstract FBA-POMDP, due to the following result:

Theorem 1 (Katt et al., 2019)**.** *Given a belief $b(s, G, \chi)$, FBA-POMCP converges to an ϵ-optimal value function of a FBA-POMDP: $V(b,a) \xrightarrow{p} V^*(b,a) - \epsilon$.*

The bias of the value function, ϵ, can be made arbitrarily small by increasing the maximum search depth. If there is no limit on the search depth, the bias ϵ is $O(\frac{\log(n)}{n})$ in the limit of the number of simulations n starting from the belief b (Silver & Veness, 2010).

The question that remains is, how good is the solution for the abstract FBA-POMDP in the original FBA-POMDP? In general, the quality of a solution when using abstraction depends on the type and quality of

the abstraction (Dearden & Boutilier, 1997; Abel et al., 2016). We define the quality η of the abstraction as the upper bound of the difference between optimal value function of the original FBA-POMDP and the value function of the original FBA-POMDP under a ϵ-optimal policy for the abstracted FBA-POMDP:

Definition 2. *An abstraction has a quality η, s.t. every ϵ-optimal solution π of the abstract FBA-POMDP applied to the original FBA-POMDP has suboptimality bounded by $\epsilon + \eta$:*

$$\forall (b, a) \in \mathbb{B} \times \mathbb{A} : \ |V^*(b, a) - V^\pi(b, a)| \leq \epsilon + \eta. \tag{19}$$

This definition is based on existing suboptimality bounds for abstract models (Dearden & Boutilier, 1997; Abel et al., 2016). More informative bounds can be derived for specific abstractions, for instance, by assessing how well the abstract model can approximate the original model (Dearden & Boutilier, 1997; Starre et al., 2023). This definition is based on existing suboptimality bounds for abstract models (Dearden & Boutilier, 1997; Abel et al., 2016). More informative bounds can be derived for specific abstractions by assessing how well the abstract model can approximate the original model (Dearden & Boutilier, 1997; Starre et al., 2023). For instance, in the context of learning with abstraction in MDPs, it has been shown that approximate model-similarity abstractions can yield such bounds by using a measure of the difference in the transition and reward functions of the grouped states (Starre et al., 2023). Extending these findings to the partially observable setting could be feasible by incorporating learning of the observation function and applying a simulation lemma for POMDPs (Lee et al., 2023).

When combined with Theorem 1, definition 2 implies that combining FBA-POMCP with abstraction leads to a near-optimal solution for the original FBA-POMDP, particularly when the abstraction effectively captures the dynamics of the original problem (i.e., when η is small):

Corollary 1. *Given a belief $b(s, G, \chi)$ and an abstraction, assuming there exists an η for which equation 19 holds, FBA-POMCP combined with abstraction converges to an $\epsilon + \eta$-optimal value function of the original FBA-POMDP: $V^\pi(b, a) \xrightarrow{p} V^*(b, a) - (\epsilon + \eta)$.*

Proof. By lemma 1 the problem after abstraction is still an FBA-POMDP. By theorem 1, we can use FBA-POMCP to get a policy π within ϵ of the optimal solution of this abstract FBA-POMDP. Then, by definition 2 this leads to a solution within $\epsilon + \eta$ of the optimal solution of the original FBA-POMDP. □

This result shows that we can reach near-optimal performance in the original problem with good abstractions, but that better performance can be achieved in theory without abstraction when $\eta > 0$. However, in practice abstraction could perform better through faster simulations, and it could get to a good performance more quickly through greater statistical strength due to aggregation.

4 Experiments

We aim to investigate three questions; 1) does abstraction lead to faster simulations and enable scaling to more complex problems, 2) does abstraction lead to faster learning by reducing unnecessary exploration, and 3) does abstraction provide these benefits when the reward function is not known? We empirically evaluated our approach on four domains to answer these questions. The first domain is the Corridor problem from the running example, a simple problem where the trade-off of using abstraction is shown. The second domain is Cracky Pavement Gridworld where we show the advantage of abstraction in a very large problem, with a state space of up to size $|\mathbb{S}| = 6 \times 10^{25}$. In addition, in this domain we show the effectiveness of the proposed normalization step for abstraction, equation 15, in the k_0 model. The third domain is an adjusted version of Collision Avoidance (Katt et al., 2019; Luo et al., 2019), made more complex to allow for more abstraction. The final domain is Room Configuration, where the agent must learn the reward function.

Experimental Setup In the experiments, we investigated various levels of abstraction across different domains and considered both a fixed amount of simulations and a fixed amount of computation time. For each level of abstraction (including the full model) and each of the different settings, we ran a separate experiment. Due to the stochasticity in the runs, we conducted up to 10000 runs for each experiment. In

Table 4: Fixed experiment settings.

Parameter	Corridor	Cracky Pavement	Collision	Room Conf
γ	0.95	0.95	0.95	0.95
# of particles in belief	500	500	500	10000
# of episodes	100	500	500	50
# of runs	100	100	10000	1000
Horizon (H)	20	12	20	13
UCB constant	5	1	500	10
Reinvigoration	Yes	No	No	Yes
Log-likelihood threshold	-1500	N/A	N/A	-500
State factors	5	$[23, 83]$	7	15
$\lvert\mathbb{S}\rvert$	320	$[5 \times 10^7, 6 \times 10^{25}]$	6000	196608

the figures, we report the moving average of the returns over a window of 10 ($\frac{x_n + \cdots + x_{n+9}}{10}$), with the shaded areas indicating the 95% confidence interval. To avoid cluttering the figures with markers, we placed only 5 markers per line, spaced evenly along the x-axis.

The settings of the experiments, and some specifics of the environments, are detailed in table 4. In the table, γ denotes the discount factor, the UCB constant is the exploration constant, and the log-likelihood threshold is the threshold below which reinvigoration is triggered. The log-likelihood is obtained during the belief update, and a low log-likelihood can indicate the belief does not adequately represent the observed data. The parameters were chosen to maintain a reasonable total run time. Because of this, we ran the experiments in the Cracky Pavement Gridworld and the Collision avoidance domains without the invigoration step. We show in appendix C that the effect of abstraction is orthogonal to the effect of reinvigoration.

We performed the experiments with a fixed number of simulations on three different machines: Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz with 384GB RAM, Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz with 190GB RAM, and AMD EPYC 7452 32-Core Processor CPU @ 2.0GHz with 256GB RAM. For the experiments with a fixed amount of computation time, we used (2 cores of) an AMD EPYC 7452 32-Core Processor CPU @ 1.5GHz with 512GB RAM. The software is written in C++.

4.1 Corridor

The Corridor domain is the domain described in the running example in section 2.1, where there are 8 different *Persons* that can be present. Each person has a slightly different probability of opening the door, such that the probability that the person opens the door during an episode is approximately between 0.0125 and 0.1. In this domain, we assume prior knowledge of all observation functions and the locations of the start, boots, button, door, and goal. In addition, we assume the structure of the transition functions of the factors *Button*, *Boots* and *Person* are known, in other words, the prior includes only models where the parents of these factors are correctly specified.

What is unknown is the structure of the transition function of the x-position, for the actions left and right. For this transition, the prior includes the following combinations of factors as parents for the x-position transitions, 1) x, 2) x and *Boots*, 3) x, *Boots*, and *Door*, and 4) x, *Boots*, *Door*, and *Button*. Each of these combinations is assigned the same probability in the prior. The transition function of the door is also not fully known, with the prior specifying a 50% chance of *Button* being present as a parent. The prior is initialized optimistically, and so the agent will initially overestimate its chances of success and the probability that the person that is present will open the door. It has to learn the correct structure and transition probabilities.

We consider the abstractions k_0, k_1, and the full model k_{inf}. The abstraction k_0 only includes the factor x. Abstraction k_1 includes x and, depending on the structures included in the belief, can also include the factors *Boots*, *Button* and *Door*.

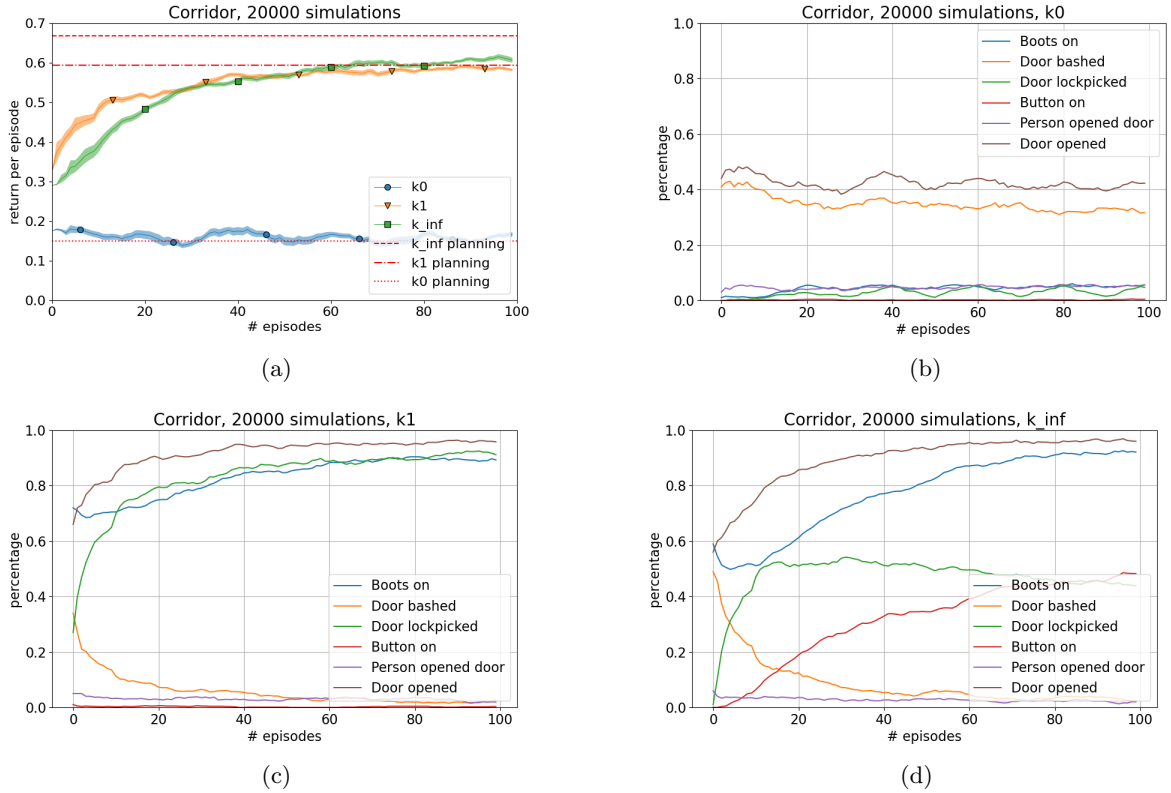


Figure 2: a) Performance in the Corridor domain. The learning behavior in the Corridor domain is shown for different models: b) k_0, c) k_1, and d) k_{inf}. The y-axis shows the percentage of runs in which a certain behavior or occurrence happened during each episode.

Results We test the effectiveness of two levels of abstraction under a fixed number of simulations. Figure 2a shows the simple moving average of the return per episode, and the shaded areas show the 95% confidence interval. Figures 2b to 2d illustrate the behavior of the different models across episodes. Specifically, these figures display how often the agent put on the boots, how often the door was opened, and through which means the door was opened. For example, in fig. 2d, it can be seen that in the first episode, the agent rarely lock picked the door (close to 0%), whereas after 20 episodes, the agent lock picked the door in more than 50% of the runs.

First, there is a large difference in the performances between k_0 and the other two models. This difference occurs because the full model and model k_1 can learn to open the door through more effective means than simply bashing against it, while k_0 cannot. This is due to the fact that the model k_0 only keeps the x factor and does not include *Door*, *Boots*, and *Button*. Consequently, it does not recognize that the actions to open the door, push the button, and put on the boots have any effect. In addition, since these actions have no direct effect on the x-position, k_0 is unable to learn interactions with the environment beyond moving left and right. It ultimately learns a strategy of just moving to the right. This strategy can still lead to reaching the goal state, as there is a chance that the door opens when the agent bashes into it by moving right, or when the person opens the door. This can be observed in fig. 2b, which shows that most of the time the agent successfully opened the door by bashing into it. While this shows the agent never learns to open the door by pushing the button, it does occasionally open the door through the *lock pick* action. This can happen near the end of the episode, with just one step remaining when the agent is at location $x = 3$ and the door is still closed. In such situations, the values for the different actions in the tree search will all be zero. Since the action is then randomly chosen among those with the same value, this can lead to selecting the *lock pick* action.

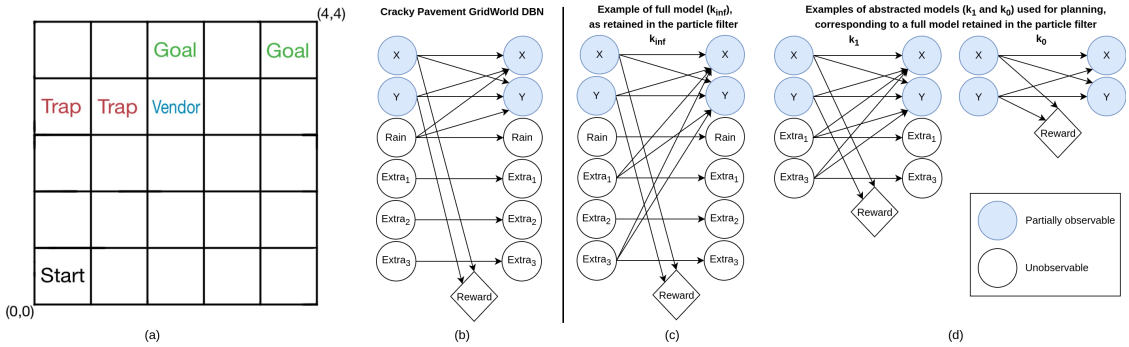


Figure 3: a) The Cracky Pavement Gridworld, b) ground truth graph representing the dynamics of the Cracky Pavement Gridworld problem with 3 extra binary factors, c) example of a full model in the particle filter, d) examples of two abstract models.

The k_1 model can effectively learn to interact with a part of the environment because it keeps not only the x factor but also the factors that influence x. This means that if the believed model is the correct model, the k_1 model will also contain *Boots* and *Door*. Consequently, k_1 learns to *put on boots* and to *lock pick* the door, leading to a much better performance than k_0. Due to greater statistical strength from aggregation, k_1 learns to use lock pick more quickly than the full model.

Technically, it would be possible for k_1 to also open the door by *pushing the button* if the *Button* factor is also believed to influence x. However, as shown in fig. 2c, this did not occur frequently in the experiments. Instead, k_1 rapidly learns to *put on boots* and *lock pick*.

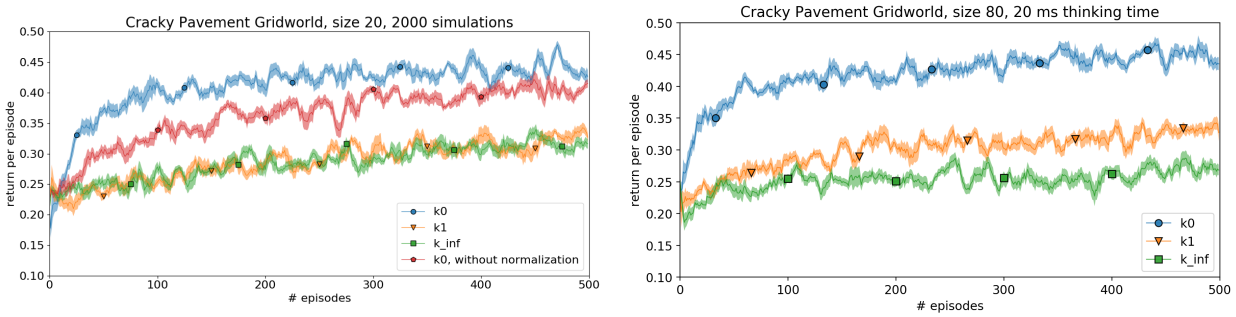
When comparing the performance of k_1 with the full model k_{inf}, we can distinguish three phases. Initially, k_1 learns more quickly than k_{inf} because the full model k_{inf} takes longer to learn to *lock pick*. Then, k_{inf} catches up as it learns to *lock pick*. Finally, k_{inf} starts to surpass k_1 by learning to open the door through pushing the button, as is visible in fig. 2d.

This experiment shows that when the abstraction is not exact, it can initially still lead to better performance because of greater statistical strength due to aggregation. Since in this domain the abstract models cannot learn the optimal behavior, they eventually get outperformed by the full model.

4.2 Cracky Pavement Gridworld

The Cracky Pavement Gridworld is a grid world as shown in fig. 3a. The DBN of the domain is shown in fig. 3b. The state space is factored into the x and y locations, *Rain*, and several extra binary factors. These extra factors could be global (e.g., light conditions) or local (e.g., presence of a chair in a specific location). In reality, only x, y, and *Rain* influence the movement of the agent, as can be seen by the incoming edges of x and y. The "Trap" and "Vendor" tiles depicted in fig. 3 do influence the movement of the agent but are not included as separate factors because their dynamics are already captured by x, y, and *Rain*. Their interaction is described in the next paragraph.

The agent is initially located at "Start" and is running low on battery, so it has to move to a charging station at one of the "Goal" locations. The agent only observes its x and y location and does so with a noisy sensor. For both x and y, the agent makes the correct observation 90% of the time. If an incorrect observation occurs, a randomly selected adjacent location is returned. At the edges, the probability of receiving the correct observation increases to 95%. The agent can move in all four directions, but the movements can fail. The chances of movement failure are influenced by a global factor called *Rain* which represents whether the tiles are dry or wet. The *Rain* factor is initialized randomly, and every timestep there is a 5% chance that the rain condition changes. On normal tiles, actions succeed 95% of the time when there is no rain and 66% of the time when it is raining. However, on the Trap tiles with cracked pavement, moves succeed only 10% of the time. Additionally, when it is not raining, a vendor with a cart occupies the Vendor tile, making it harder to move past with only a 10% success rate for moving. Conversely, during rain, the Vendor tile is



(a) (b)

Figure 4: Performance in the Cracky Pavement Gridworld domain, a) with a fixed thinking time, b) with a fixed number of simulations.

vacant and functions like a normal tile. Therefore, to behave optimally, the agent should traverse the Vendor tile when it is raining and circumvent it when dry.

In this domain, we assume prior knowledge of all observation functions, the transition functions of *Rain* and the extra binary factors, and the start and goal locations. Additionally, we assume that it is known that the x and y factors both (at least) depend on each other. However, there is no prior knowledge of the trap locations and the vendor locations, meaning that none of the possible count tables in the belief space specify different movement probabilities on these locations. Some of the graph structures in the belief space include *Rain* and/or extra binary factors (three in fig. 3b) as parents of x and y, implying the agent does not know whether or not these factors influence its movement. The extra binary factors are initialized randomly and have a 20% of changing at each step. In this problem, the agent has to learn that x, y, and *Rain* are the only relevant factors for its movement. This task is complicated by the interaction between the trap states and several uninformative factors, as the agent may mistakenly attribute its inability to move on trap states to the presence of some of these uninformative factors.

As shown in table 4 on page 15, we run the experiments with two different amounts of extra binary factors: 20 and 80. The total amount of state factors is 23 and 83, respectively, since both settings also have the X, Y, and *Rain* factors. Scalability in the number of factors is very hard and important because the size of the state space, and the possible graph structures, grow exponentially with the number of factors. With 20 and 80 extra binary factors this domain has a state space of approximately 5×10^7 and 6×10^{25} states, respectively. Factored representations are needed to find solutions for problems of such sizes. Flat learning methods like Bayes-Adaptive POMCP (Katt et al., 2017) are not feasible here, even in the simple case of 20 extra binary factors. This is because representing the transition table for just 1 action, a table of size $|S|^2$, requires more than 9 million GB per particle.

We consider the abstractions k_0, k_1, and the full model k_{inf}. The abstraction k_0 includes only the factors x and y since they are the ones that directly influence the reward (fig. 3b). Abstraction k_1 also includes the parents of x and y. We scale the number of extra binary factors to test the speedup and to see how it affects the performance.

Results First, we examine the results for one particular setting, with 80 extra binary factors and 2000 simulations. Figure 4a shows these results, where the lines represent the simple moving average of the return per episode, and the shaded regions indicate the 95% confidence interval. Both k_1 and the full model k_{inf} struggle to learn a good policy, while k_0 outperforms both.

The primary difference between k_0 and the other two models is that k_0 retains only the x and y factor, while k_1 includes any factor it believes influences x or y, such as *Rain* or parts of the additional binary factors, and k_{inf} retains all factors as it does not abstract. Although k_0 sacrifices the ability to account for *Rain*, it performs better because it simplifies learning about the trap states by only considering x and y. This leads to greater statistical strength as it is much easier to learn $P(x'|x, y)$ than $P(x'|x, y, \text{rain, and numerous binary factors})$.

Table 5: Average number of simulations in the Cracky Pavement Gridworld.

	Size 20			Size 80		
Time	k_0	k_1	k_{inf}	k_0	k_1	k_{inf}
5ms	634	550	272	521	429	149
10ms	1324	1263	573	1097	871	240
15ms	1978	1658	660	2044	1726	328
20ms	2571	2145	803	2433	2360	440

Table 6: Average return over the first 500 episodes in the Cracky Pavement Gridworld.

	Size 20			Size 80		
Time	k_0	k_1	k_{inf}	k_0	k_1	k_{inf}
5ms	0.33	0.26	0.23	0.31	0.25	0.18
10ms	0.39	0.29	0.26	0.37	0.28	0.21
15ms	0.41	0.31	0.27	0.41	0.30	0.23
20ms	0.42	0.30	0.28	0.42	0.30	0.25

While k_0 cannot distinguish between rain and no rain and therefore does not learn when the vendor is on the tile, it can learn to navigate around this tile. Although this is generally not optimal, it is optimal when rain and the vendor are not considered. The full model k_{inf} and k_1 can eventually outperform k_0 by learning that it is better to cross the vendor tile when it is raining, as shown in appendix C. However, this takes a considerable amount of time, and during the earlier episodes, k_0 performs much better earlier.

These results shows that the greater statistical strength obtained by removing information, like *Rain* and extra binary factors, can result in a significant increase in performance. Additionally, fig. 4a compares the performance of k_0 with and without the proposed normalization step for abstraction (equation 16), demonstrating that the normalization step can significantly improve learning performance.

In fig. 4b, where we compare the performance with a fixed amount of thinking time instead of a fixed number of simulations, we see that k_1 outperforms the full model k_{inf}. The main difference between k_1 and k_{inf} is that k_1 abstracts away all the factors that, given the graph topology, are not directly or indirectly relevant for the reward. This means that the k_1 model is generally smaller than the k_{inf} model, leading to faster simulations. As shown in table 5, k_1 performs an average of 2360 simulations with 20ms of thinking time, while the full model only reaches 440 simulations. This increase in simulation speed results in the improved performance shown in fig. 4b.

Table 6 shows that an increase in thinking time generally increases performance, most notably when increasing from 5ms to 10ms, with diminishing returns for further increases. The differences between k_1 and k_{inf} are also most pronounced at lower thinking times, especially with 80 additional binary factors, where the abstraction provides the most benefit. These findings demonstrate that augmenting FBA-POMCP with abstraction can increase performance through computational efficiency.

Overall, these results show that abstraction can be beneficial in multiple ways. Increasing the simulation speed leads to better performance, and simplifying the problem leads to faster learning due to greater statistical strength. The improvement in simulation speed is most pronounced when many factors are abstracted away, maximizing the difference in simulation speed between the models with and without abstraction.

4.3 Collision Avoidance

In the Collision Avoidance domain, the agent flies from one side to the other in a 10 (width) x 5 grid. The episode ends when the agent reaches the last column, where it has to avoid colliding with a moving obstacle. This obstacle has a 20% chance to stay stationary and otherwise randomly moves either up or down. The agent can decide to move up, down, or stay level.

We increased the complexity of the original Collision Avoidance (Katt et al., 2019; Luo et al., 2019) by adding additional factors. These additional factors influence those in the original problem. Figure 5 shows the resulting dynamics. We added the factors *Speed* (slow, fast), *Traffic* (low or high amount of traffic), *Time of Day* (day, night), and *Obstacle Type* (3 types, e.g., helicopter, plane). *Obstacle Type* and *Time of Day* are fully observable and do not change during the episode. The agent receives a noisy observation of the obstacle (accurate around 80% of the time). The agent has an 85% chance to move one cell forward. If the *Speed* is high, it has a 15% chance to move forward two columns. If the *Speed* is low, it has a 15% chance to stay in the same column.

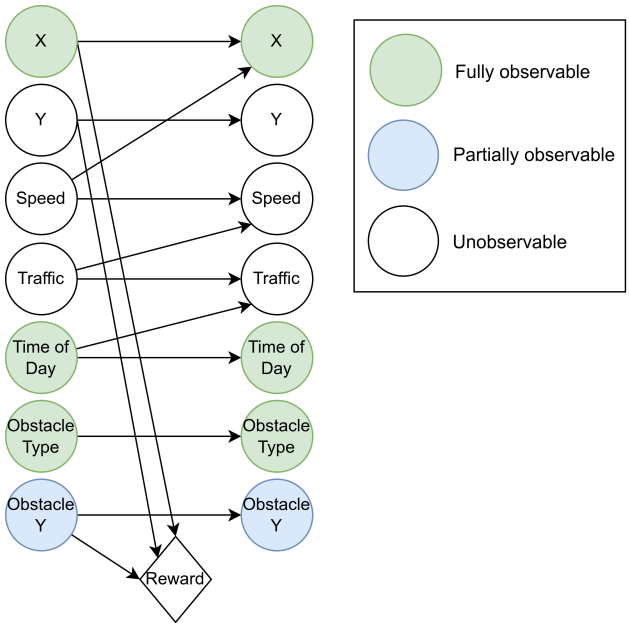


Figure 5: Ground truth graph representing the dynamics of the Collision avoidance problem.

The *Speed* is influenced by the *Traffic*. When the amount of *Traffic* is low, there is a 90% chance that the *Speed* will change to (or stay at) high and a 10% chance that the *Speed* will change to (or stay at) low. This is reversed when the *Traffic* is high. The *Traffic* is influenced by the *Time of Day* in a similar way, when it day there is a 90% chance that the *Traffic* will change to (or stay at) high and a 10% chance that the *Traffic* will change to (or stay at) low. This is reversed when it is night. The Time of Day is randomly chosen at the start of an episode, with a 50% chance of either day or night.

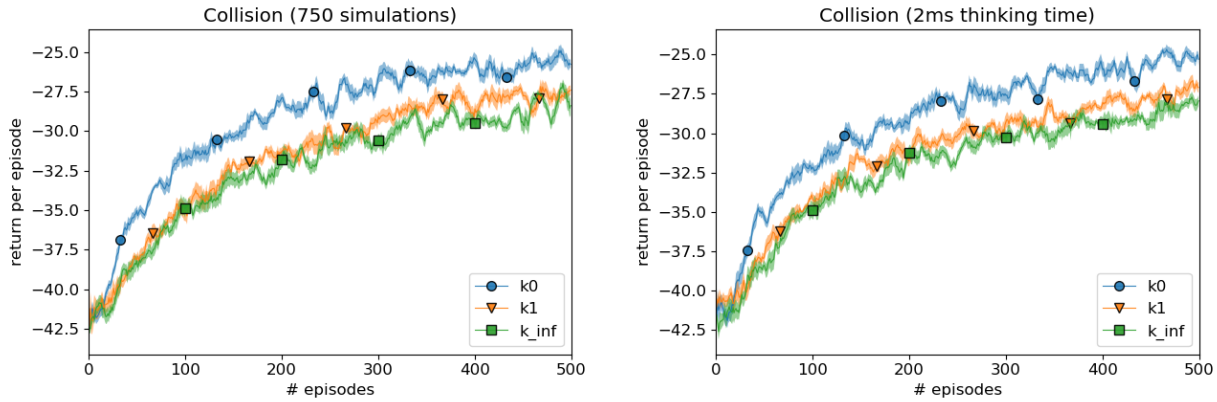
We assume prior knowledge of all transition and observation functions, except for the transition probabilities of the obstacle. There are four different graph structures for the obstacle transition function to which the prior assigns a positive probability. These structures include: 1) the structure where *Obstacle Y* is only influenced by itself; 2 and 3) the structures where it is influenced by itself and either *Speed* or *Obstacle Type*; and 4) the structure where it is influenced by itself, *Speed*, and *Obstacle Type*. Each structure has a 25% probability.

We consider the abstractions k_0, k_1, and k_2, which is equivalent to the full model k_{inf}. The abstraction k_0 includes the factors x, y, and *Obstacle Y*. The abstraction k_1 additionally includes the factors *Speed* and *Obstacle Type*, if *Obstacle Type* has a connection to *Obstacle Y*.

Results We again test the effectiveness of two levels of abstraction under a fixed number of simulations. One of the benefits of abstraction is a smaller and (therefore) faster model. Another benefit is an increase in statistical strength through the removal of factors. The lines in fig. 6 show the simple moving average of the return per episode, and the shaded areas show the 95% confidence interval.

We see that abstraction k_0 learns significantly faster than both k_1 and k_{inf}, both with a fixed number of simulations and with a fixed thinking time. The abstraction k_0 allows for faster learning of the transition function of *Obstacle Y* since it removes the (possible) parents *Speed* and *Obstacle Type*. When *Speed* and *Obstacle Type* are parents of *Obstacle Y* in the sampled graph structure, k_1 retains these connections. As a result, k_1 learns the transition function of *Obstacle Y* at a rate comparable to k_{inf}.

In k_1, *Time of Day* and *Traffic* are removed. However, since their transition functions, along with that of *Speed*, are considered known, this does lead to greater statistical strength in learning *Speed*. The impact of *Time of Day* and *Traffic* is also less pronounced, as they only influence x indirectly through *Speed*. Nevertheless, k_1 tends to perform slightly better than k_{inf}. This is not due to faster simulations; in this



(a) (b)

Figure 6: Performance in the Collision Avoidance domain, a) with a fixed thinking time, b) with a fixed number of simulations.

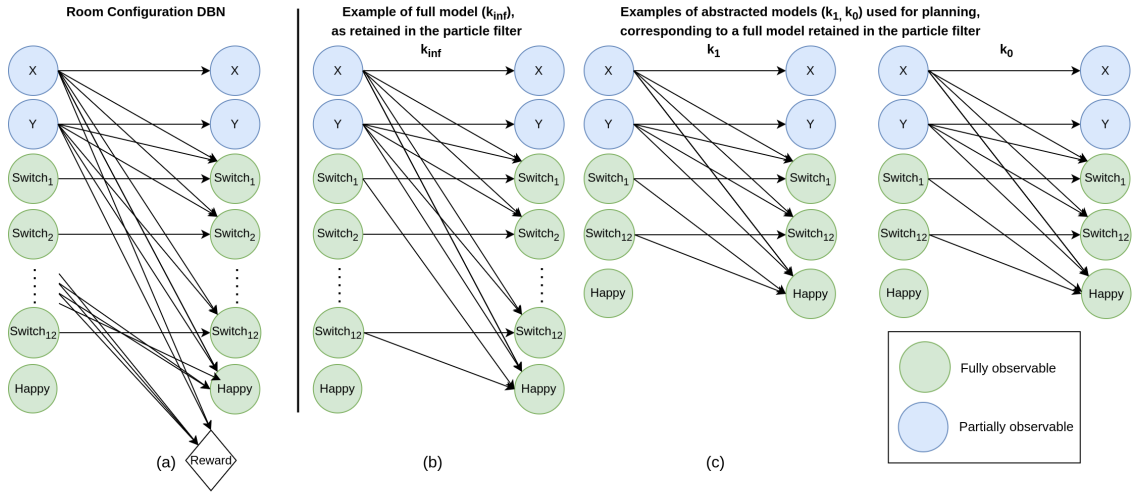


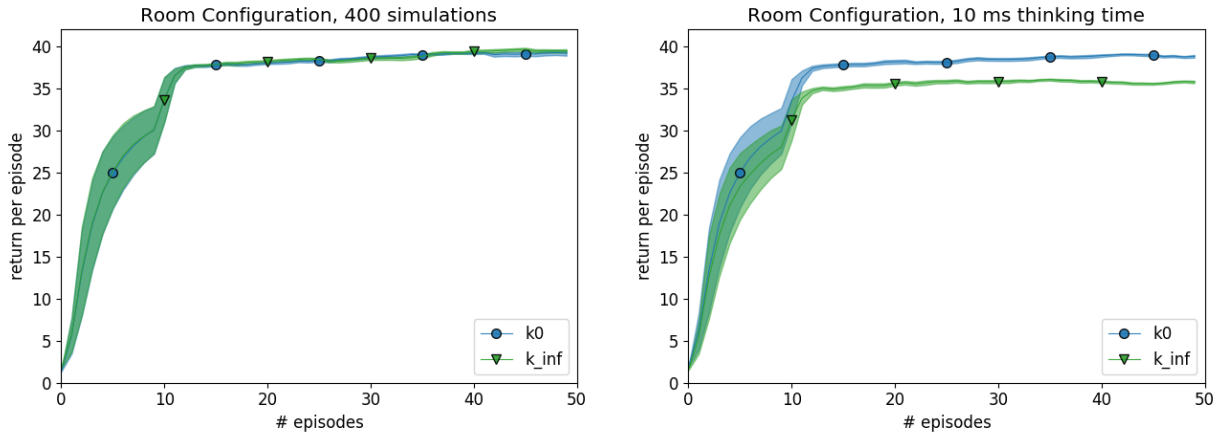
Figure 7: a) Ground truth graph representing the dynamics of the Room Configuration domain for the *switch* action, b) example of a full model in the particle filter, c) examples of two abstract models.

domain, any speed up is minimal, and similar results are observed even with a fixed number of simulations. One potential advantage of removing *Time of Day* and *Traffic* is that it reduces the branching factor in the tree, as there will be no separate observations for these factors. This focuses the tree search by eliminating the need to consider these factors, albeit at a slight cost to model accuracy.

4.4 Room Configuration

In the Room Configuration domain, the task of the agent is to set up a classroom in a desirable way for a teacher. We model this environment as a 4 (width) x 3 grid, where each tile contains a configurable item with two settings. The agent can change the configuration of these items. The teacher is concerned only with the configuration of three specific items and is happy once these are configured correctly.

The reward function is (largely) unknown to the agent in this domain, as the agent does not initially know configuration factors that influence the reward. To address this, we model the reward with a state factor called *Happy*, which takes on the same value as the reward and is fully observable. While *Happy* is fully



(a) (b)

Figure 8: Performance in the Room Configuration domain, a) with a fixed thinking time, b) with a fixed number of simulations.

observable, the agent does not know exactly which configuration factors influence it. That is, the prior belief assigns a non-zero probability to multiple sets of parents of *Happy*. Therefore, the agent must learn which configuration factors are relevant to the reward. The *Happy* factor has four different states: neutral (0 reward), slightly unhappy (-1), slightly happy ($+1$), and very happy (100). The agent receives a reward when it performs the switch action, and this reward depends on its location and the status of the configurable items. Specifically, the agent receives a small reward or penalty (± 1) when it changes the configuration of an item to the correct or incorrect setting, respectively. Once it sets the configuration of all three items the teacher desires, it receives a large reward (100).

While the configurations of the items are fully observable, the agent receives noisy observations of its own location. For both x and y, the agent makes the correct observation 95% of the time. If an incorrect observation occurs, a randomly selected adjacent location is returned. At the edges, the probability of receiving the correct observation increases to 97.5%. Movement actions have a 95% chance of moving in the intended direction, whereas the *switch* action to change the item settings is always successful. The agent has prior knowledge of the observation function and the transition function for switching. Additionally, all the possible initial belief states underestimate the probability that the move action is successful.

We consider the abstraction k_0 and the full model k_{inf}. The abstraction k_0 keeps the factors connected to the reward. The structures in the prior belief always include x, y as parents of *Happy*. The prior belief is defined such that it assigns a non-zero probability only to structures where the abstract models k_1 and k_0 are identical. Thus, in the experiment, we only use k_0.

Results We test learning of the reward function and the advantage of abstraction when only factors that (the agent believes) are irrelevant to the reward are abstracted away. The lines in fig. 8 show the moving average of the return per episode, and the shaded areas show the 95% confidence interval. We can see that the agent learns how to perform well in 10 episodes, after which the performance remains the same.

The abstraction k_0 performs more simulations (455) than the full model k_{inf} (365) within a fixed amount of time (10ms). This computational advantage allows k_0 to outperform the full model, as shown in fig. 8b. Figure 8a further demonstrates that there is no performance difference between the two models when the number of simulations is fixed. These results show that the agent can quickly learn the structure of the reward and that abstraction can increase performance through its increased simulation speed.

5 Related Work

The FBA-POMDP is a factored version of the tabular BA-POMDP (Ross et al., 2011). The infinite-POMDP approach (Doshi-Velez, 2009) is a non-parametric Bayesian approach. This approach requires no knowledge of the state space. However, it assumes a hierarchical Dirichlet Process as prior, for which providing an informative prior can be difficult. BRL in continuous POMDPs typically makes Gaussian assumptions, such as in Dallaire et al. (2009), where Gaussian processes are the model of choice. To extend our approach to continuous POMDPs, DBNs that work with continuous variables (Grzegorczyk & Husmeier, 2011) could be investigated. Alternatively, constructing different levels of abstraction in a continuous domain could done through varying levels of discretization.

The BA-MDP (Duff, 2002), and the respective solution methods (Poupart et al., 2006; Ross & Pineau, 2008; Vien et al., 2013), are the fully observable counter-part to the (F)BA-POMDP. In this setting (BRL for MDPs), work has been done on exploring applications of Deep RL (Zintgraf et al., 2019; Hoang & Vien, 2020). This line of work solves the (easier) fully observable problem, and how to extend these methods to POMDPs is unclear.

Other approaches make use of recurrent networks for dealing with partial observability to generalize deep RL to POMDPs (Dung et al., 2008; Hausknecht & Stone, 2015; Zhu et al., 2017; Meng et al., 2021). However, these networks are general-purpose, requiring many samples. This realization has motivated RL-specific architectures designed to capture history efficiently (Jaderberg et al., 2019; Ma et al., 2019). Deep variational methods are another approach that can be efficient (Igl et al., 2018; Tschiatschek et al., 2018). However, none of these methods allow for encoding prior knowledge or tackling the exploration problem, to which the FBA-POMDP framework provides an elegant solution.

In the last two decades, there have additionally been many different approaches to (not deep or Bayes) learning in POMDPs, e.g., McCallum (1996); Shani et al. (2005); Azizzadenesheli et al. (2016); Liu & Zheng (2019); Bennett & Kallus (2024). However, these also do not allow for encoding prior knowledge or tackling the exploration problem.

Planning with abstractions has been studied before, mostly in the context of MDPs. The method by Dearden & Boutilier (1997), that our approach is based on, applied abstraction to factored problems. We extend this work to the partially observable BRL and planning setting and introduce a mechanism to automatically create multiple levels of abstraction based on the structure of the problem. Another line of work has focused on building abstractions during the tree search (Hostetler et al., 2014; Anand et al., 2016). Hostetler et al. (2014) also provide results for doing the tree search using an abstraction function. However, this means the simulations themselves need to be done using the full model. Chitnis et al. (2020) instead first learn an abstraction, which they then use for planning. Rather than being given an abstraction, their goal is to learn one. We investigate the effects of an abstraction method on learning efficiency and performance. He et al. (2020) do planning in the partially observable setting and construct an abstract model before the tree search. The key difference with our approach is that we do not assume access to a simulator of the real environment. Instead, we *learn* abstract dynamics while acting in the real-world.

6 Discussion

A limitation of the FBA-POMCP method is that it does not plan beyond the current episode. By not planning for future episodes, it does not consider the value that knowledge obtained in the current episode can have for future episodes. Approaches that quantify information gain (Zhang et al., 2020; Ambrogioni, 2021; Liu et al., 2021) and those that plan for long horizons (Grover et al., 2020) could address this limitation, and future work could explore combining these strategies.

A difficulty of applying abstractions within FBA-POMCP lies in marginalizing the belief. As discussed in section 3.3, when aggregating counts for approximate abstractions, it is unclear what the best solution is, since this is generally unknown beforehand and can depend on the policy. Even for a fixed policy, marginalization may lead to a belief that misrepresents the true (abstract) dynamics. For example, in the corridor problem, marginalizing away the *Button* factor leads to misrepresented abstract dynamics because

marginalizing it away combines the situations where the button is in the non-pushed state with those where it is in the pushed state, implicitly resulting in a belief that at each step there is a probability that the button will be in the pushed state. This is a misrepresentation since the button starts in the non-pushed state and, since it is marginalized away, the agent will not learn to push the button, so it will always be in the non-pushed state. Such situations can make it more difficult for the agent to learn. An idea could be to use knowledge about the starting position and the abstracted model in the marginalization. For instance, if the *Button* factor is removed from the model, we could keep the counts for when it is in the non-pushed state and ignore the counts for the pushed state since it is always in the non-pushed state at the beginning of an episode.

As shown in the Corridor experiment, abstraction does not always help, and can in fact be detrimental. When crucial information is removed, it can make learning impossible. In general, the usefulness of abstraction depends on how much value (or information) is lost. If the loss in value is too great, more simulations will not help to achieve a better performance. On the other hand, if there is no loss in value or the loss in value is relatively small, abstraction can help improve performance by making learning easier and by speeding up simulations during planning.

It can be difficult to determine whether abstraction will be beneficial, as in RL the problem is often not fully known. A way to handle this could be with *abstraction selection* (Ortner et al., 2014; Jiang et al., 2015), where the algorithm chooses which abstraction to select during learning. For instance, selection could be done by deriving value loss bounds for specific abstractions (Dearden & Boutilier, 1997) and using these to make a decision. Alternatively, the problem of abstraction selection could be viewed as a non-stationary multi-armed bandit problem, where at each episode we have to select a (abstract) model. The problem is non-stationary since the models the agent learns change each episode with the experience it obtains, and thus the policy and the expected reward can also change. Multi-armed bandit methods that deal with such non-stationarity could be used (Garivier & Moulines, 2011; Cheung et al., 2019; Trovo et al., 2020).

7 Conclusion

We proposed combining learning and online planning for BRL for FBA-POMDPs with abstraction. We empirically showed that this combination significantly improves learning, scalability, and performance, using an intuitive and straightforward form of abstraction. This happens through several effects. First, we have shown that abstraction improves performance by increasing the simulation speed. Moreover, we have shown that abstraction improves performance even with a fixed number of simulations through greater statistical strength. With an abstract model, the agent takes different actions since it does not need to explore the dynamics of less relevant factors. Finally, the abstraction method allows FBA-POMCP to learn in very large problems with a state space up to $|\mathbb{S}| = 6 \times 10^{25}$.

To the best of our knowledge, this is the first work to explore abstraction in BRL for POMDPs, representing an initial step in investigating this combination. In the future, abstraction could also be further incorporated into FBA-POMDP by directly maintaining a belief over which factors should be part of the model.

References

D. Abel, D. Hershkowitz, and M. Littman. Near optimal behavior via approximate state abstraction. In *ICML*, 2016.

Luca Ambrogioni. Knowledge is reward: Learning optimal exploration by predictive reward cashing. *arXiv preprint arXiv:2109.08518*, 2021.

Ankit Anand, Ritesh Noothigattu, Parag Singla, et al. Oga-uct: On-the-go abstractions in uct. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26, 2016.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. Reinforcement learning of pomdps using spectral methods. In *Conference on Learning Theory*, pp. 193–256. PMLR, 2016.

Andrew Bennett and Nathan Kallus. Proximal reinforcement learning: Efficient off-policy evaluation in partially observed markov decision processes. *Operations Research*, 72(3):1071–1086, 2024.

Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 1168–1175. Citeseer, 1996.

Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. 4(1):1–43, 2012.

Wang Chi Cheung, David Simchi-Levi, and Ruihao Zhu. Learning to optimize under non-stationarity. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1079–1087. PMLR, 2019.

Rohan Chitnis, Tom Silver, Beomjoon Kim, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Camps: Learning context-specific abstractions for efficient planning in factored mdps, 2020.

Elena Congeduti and Frans A Oliehoek. A cross-field review of state abstraction for markov decision processes. In *34th Benelux Conference on Artificial Intelligence (BNAIC) and the 30th Belgian Dutch Conference on Machine Learning (Benelearn)*, 2022.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pp. 72–83. Springer, 2006.

Patrick Dallaire, Camille Besse, Stephane Ross, and Brahim Chaib-draa. Bayesian reinforcement learning in continuous pomdps with gaussian processes. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2604–2609. IEEE, 2009.

Tim De Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3):1394–1401, 2018.

Richard Dearden and Craig Boutilier. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1-2):219–283, 1997.

Finale Doshi-Velez. The infinite partially observable markov decision process. *Advances in neural information processing systems*, 22:477–485, 2009.

Michael O'Gordon Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. University of Massachusetts Amherst, 2002.

Le Tien Dung, Takashi Komeda, and Motoki Takagi. Reinforcement learning for pomdp using state classification. *Applied Artificial Intelligence*, 22(7-8):761–779, 2008.

Bela A Frigyik, Amol Kapila, and Maya R Gupta. Introduction to the dirichlet distribution and related processes. *UWEE Technical Report (UWEETR-2010-0006)*, 2010.

Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for switching bandit problems. In *International conference on algorithmic learning theory*, pp. 174–188. Springer, 2011.

Divya Grover, Debabrota Basu, and Christos Dimitrakakis. Bayesian reinforcement learning via deep, sparse sampling. In *International Conference on Artificial Intelligence and Statistics*, pp. 3036–3045. PMLR, 2020.

Marco Grzegorczyk and Dirk Husmeier. Non-homogeneous dynamic bayesian networks for continuous data. *Machine Learning*, 83:355–419, 2011.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

J He, M Suau de Castro, and FA Oliehoek. Influence-augmented online planning for complex environments. *Advances in Neural Information Processing Systems*, 33, 2020.

Tai Hoang and Ngo Anh Vien. Bayes-adaptive deep model-based policy optimisation, 2020.

Jesse Hostetler, Alan Fern, and Tom Dietterich. State aggregation in monte carlo tree search. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning*, pp. 2117–2126. PMLR, 2018.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Nan Jiang, Alex Kulesza, and Satinder Singh. Abstraction selection in model-based reinforcement learning. In *International Conference on Machine Learning*, pp. 179–188, 2015.

Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39:407–428, 2015.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Sammie Katt, Frans A Oliehoek, and Christopher Amato. Learning in pomdps with monte carlo tree search. In *International Conference on Machine Learning*, pp. 1819–1827. PMLR, 2017.

Sammie Katt, Frans A Oliehoek, and Christopher Amato. Bayesian reinforcement learning in factored pomdps. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 7–15, 2019.

Sammie Katt, Hai Nguyen, Frans A Oliehoek, and Christopher Amato. Baddr: Bayes-adaptive deep dropout rl for pomdps. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pp. 723–731, 2022.

Jonathan Lee, Alekh Agarwal, Christoph Dann, and Tong Zhang. Learning in pomdps is sample-efficient with hindsight observability. In *International Conference on Machine Learning*, pp. 18733–18773. PMLR, 2023.

L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

Evan Z Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In *International conference on machine learning*, pp. 6925–6935. PMLR, 2021.

Yunlong Liu and Jianyang Zheng. Online learning and planning in partially observable domains without prior knowledge, 2019.

Yuanfu Luo, Haoyu Bai, David Hsu, and Wee Sun Lee. Importance sampling for online planning under uncertainty. *The International Journal of Robotics Research*, 38(2-3):162–181, 2019.

Xiao Ma, Peter Karkus, David Hsu, Wee Sun Lee, and Nan Ye. Discriminative particle filter reinforcement learning for complex partial observations. In *International Conference on Learning Representations*, 2019.

Andrew W Marshall. The use of multi-stage sampling schemes in monte carlo computations. Technical report, RAND CORP SANTA MONICA CALIF, 1954.

Andrew Kachites McCallum. *Reinforcement learning with selective perception and hidden state*. University of Rochester, 1996.

Lingheng Meng, Rob Gorbet, and Dana Kulić. Memory-based deep reinforcement learning for pomdps. In *2021 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 5619–5626. IEEE, 2021.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Kevin Patrick Murphy. *Dynamic Bayesian networks: representation, inference and learning*. University of California, Berkeley, 2002.

Ronald Ortner, Odalric-Ambrym Maillard, and Daniil Ryabko. Selecting near-optimal approximate state representations in reinforcement learning. In *International Conference on Algorithmic Learning Theory*, pp. 140–154. Springer, 2014.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete bayesian reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 697–704, 2006.

Stéphane Ross and Joelle Pineau. Model-based bayesian reinforcement learning in large structured domains. In *Conference on Uncertainty in Artificial Intelligence*, volume 2008, pp. 476. NIH Public Access, 2008.

Stéphane Ross, Joelle Pineau, Brahim Chaib-draa, and Pierre Kreitmann. A bayesian approach for learning and planning in partially observable markov decision processes. *Journal of Machine Learning Research*, 12(5), 2011.

Stuart J Russell and Peter Norvig. Artificial intelligence: a modern approach. malaysia, 2016.

Guy Shani, Ronen I Brafman, and Solomon E Shimony. Model-based online learning of pomdps. In *Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings 16*, pp. 353–364. Springer, 2005.

David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Neural Information Processing Systems*, 2010.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Rolf AN Starre, Marco Loog, Elena Congeduti, and Frans A Oliehoek. An analysis of model-based reinforcement learning from abstracted observations. *Transactions on Machine Learning Research*, 2023.

Sebastian Thrun. Monte carlo pomdps. In *Advances in Neural Information Processing Systems*, volume 12, pp. 1064–1070, 1999.

Francesco Trovo, Stefano Paladino, Marcello Restelli, and Nicola Gatti. Sliding-window thompson sampling for non-stationary settings. *Journal of Artificial Intelligence Research*, 68:311–364, 2020.

Sebastian Tschiatschek, Kai Arulkumaran, Jan Stühmer, and Katja Hofmann. Variational inference for data-efficient model learning in pomdps, 2018.

Ngo Anh Vien, Wolfgang Ertel, Viet-Hung Dang, and TaeChoong Chung. Monte-carlo tree search for bayesian reinforcement learning. *Applied intelligence*, 39(2):345–353, 2013.

John Von Neumann. Various techniques used in connection with random digits. *Appl. Math Ser*, 12(36-38): 3, 1951.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34, 2021.

Jin Zhang, Jianhao Wang, Hao Hu, Yingfeng Chen, Changjie Fan, and Chongjie Zhang. Learn to effectively explore in context-based meta-rl. *arXiv preprint arXiv:2006.08170*, 2020.

Pengfei Zhu, Xin Li, Pascal Poupart, and Guanghui Miao. On improving deep reinforcement learning for pomdps. *arXiv:1704.07978*, 2017.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning, 2019.

A Algorithms

Algorithm 7 shows the main loop of the FBA-POMCP algorithm (Katt et al., 2019). The *Simulate* that is uses is shown in algorithm 8. The GreedyActionSelection selects the action that has the highest value. In case of a tie, it randomly selects one of the actions with the highest value. Algorithm 9 shows the *Step* function when abstractions are used.

Algorithm 7 FBA-POMCP

Input: B: particle filter with hyper-states $\dot{s}$
num_sims: number of simulations to do.
$h_0 \leftarrow ()$ // The empty history (i.e., now)
for $i \in 1, \ldots,$ num_sims **do**
  // First, we root sample a hyper-state:
  $\dot{s} \sim B$ // Sample from belief
  Simulate$(\dot{s}, 0, h_0)$
end for
$a \leftarrow$ GreedyActionSelection(h_0).
return a

Algorithm 8 Simulate

Input: $\dot{s} = \langle s, G, \chi \rangle$: hyper-state
d: search depth
h: simulated history.
if IsTerminal$(h)||d ==$ max_depth **then**
  return 0
end if
$a \leftarrow$ UCBactionSelection(h)
$R \sim R(\dot{s}, a)$
$\dot{s}', o \leftarrow$ Step$(\dot{s}, a)$
$h' \leftarrow (h, a, o)$
if $h' \in$ Tree **then**
  $r \leftarrow R + \gamma$ Simulate$(\dot{s}', h')$
else
  ConstructNode(h')
  $r \leftarrow R + \gamma$ RollOut$(\dot{s}', h')$
end if
$(\ldots)$ // Update statistics in nodes
return r

Algorithm 9 Step (with abstraction)

Input: $\bar{s}$: abstracted hyper-state
a: simulated action.
// Recall that $\bar{s} = \langle \dot{s}, \bar{G}, \bar{\chi} \rangle$, with $\dot{s}$ containing a current state s.
$s', o \sim p_{\bar{G}, \bar{\chi}}(\cdot|s, a)$ // Sample next state and observation from abstracted counts.
$\dot{s}' \leftarrow \langle s', G, \chi \rangle$
$\bar{s}' \leftarrow \langle \dot{s}, \bar{G}, \bar{\chi} \rangle$
return $\bar{s}', o$

B Proof

We restate the Lemma from section 3.4 and give a proof sketch:

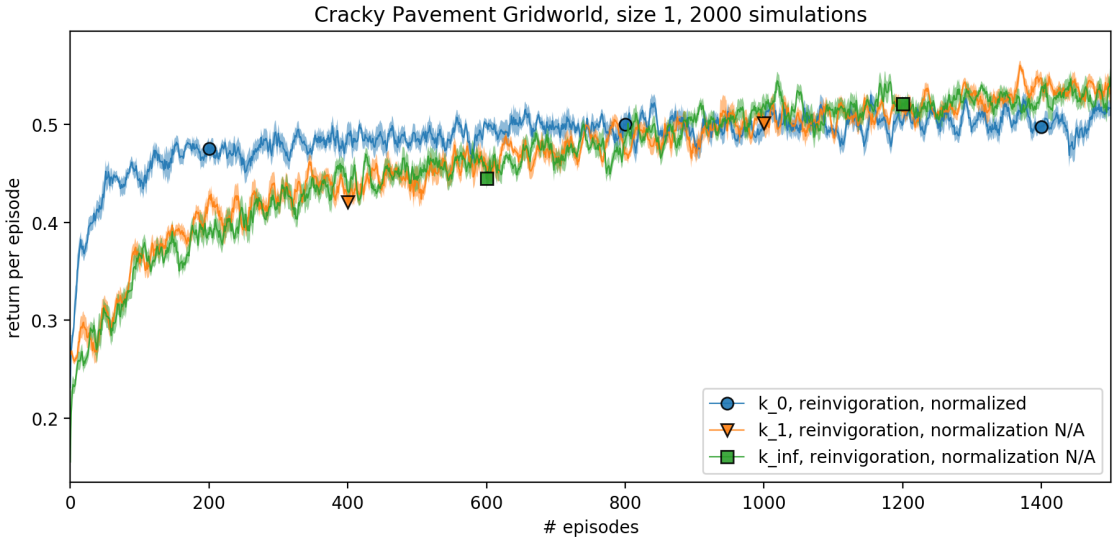


Figure 9: Comparison with invigoration for a longer number of episodes.

Lemma 1. *The result of applying the abstraction method, as described in Algorithm 4 and Section 3.3, to the original FBA-POMDP results in another FBA-POMDP, the abstract FBA-POMDP.*

Proof. First, we define the observation space and function, followed by the state space for different levels of abstraction. The reward function does not require changes since, during abstraction, the models in the particles will always keep the factors that are believed to be part of the IR set.

As detailed in section 3.3.4, the abstract observation space $\bar{\mathbb{O}}$ is enhanced by including a "not observed" option for all observation variables. This addition addresses scenarios where the observation function depends on state factors that are abstracted away; in such cases, the observation function will return "not observed". Otherwise, the observation space and function remain unchanged.

For the state space, we distinguish abstraction level k_0 from other abstraction levels. For k_0, only the factors in the set IR are included, while higher abstraction levels can include additional factors depending on the topology.

For level k_0, the state space $\bar{\mathbb{S}}$ is derived by aggregating states based on the distinct values of the remaining factors. Transition functions are adjusted for the new state space through the marginalization procedure described in section 3.3.

For higher abstraction levels, the belief specifies the factors that are relevant. For example, for abstraction level k_1, factors that directly influence the IR set are also considered relevant. Similar to k_0, factors that are never considered relevant are removed. For state factors included in only some models, the abstract state space is enhanced by adding a "not relevant" value. This value ensures transitions default to "not relevant" when the factor is not present in a model. Transitions are then adjusted for the new state space through the marginalization procedure. The observation function is also adjusted to return "not observed" for factors that return "not relevant".

Combining these transformations, we obtain a fully specified FBA-POMDP after abstraction. □

In practice, implementing the "not relevant" value is unnecessary, as only factors included in the simulated particle affect observations and actions during tree search.

C Further Experiments

Here, we demonstrate that abstraction can improve performance when we use the invigoration step from FBA-POMCP, and that k_1 and the full model k_{inf} can eventually outperform k_0. We conducted 100 runs for each setting and report the simple moving average of the return per episode, with the shaded areas showing the standard error.

We ran an experiment with invigoration over a larger number of episodes, as shown in fig. 9. Initially, k_0 outperforms k_1 and k_{inf}. However, with enough data, the k_1 and k_{inf} models become accurate enough to surpass k_0 in performance. The k_1 model matches the performance of k_{inf} because the number of simulations is fixed.

For larger problem sizes and with a fixed thinking time instead of a fixed number of simulations, k_1 is expected to initially outperform k_{inf}. Additionally, k_0 should show an even greater initial improvement over k_{inf}, and it could take even longer for k_{inf} to surpass the performance of k_0.