Backward Monte Carlo Tree Search: Charting Unsafe Regions in the Belief-Space

ANIL YILDIZ*, Stanford Intelligent Systems Laboratory, Stanford University, USA
ESEN YEL, Reliable Intelligent Systems Laboratory, Rensselaer Polytechnic Institute, USA
MARCELL VAZQUEZ-CHANLATTE, Nissan Advanced Technology Center, Silicon Valley, USA
KYLE WRAY, Khoury College of Computer Sciences, Northeastern University, USA
MYKEL J. KOCHENDERFER, Stanford Intelligent Systems Laboratory, Stanford University, USA
STEFAN J. WITWICKI, Nissan Advanced Technology Center, Silicon Valley, USA

Safety-critical systems often operate in partially observable environments, where assessing the safety of the underlying policy remains a fundamental challenge. This study focuses on evaluating policies by identifying regions of the belief-space that can lead the system's policy to an undesirable state with a non-negligible probability. In this paper, we introduce Backward Monte Carlo Tree Search, the first Monte Carlo tree search framework that expands *backward in time* within the belief-space. The tree search begins from an undesired terminal belief and recursively explores its possible predecessors, constructing a tree of belief transitions that could lead to an unsafe outcome within a given horizon. Evaluations in gridworld and autonomous driving domains show that identifying beliefs from which failures may occur enables runtime risk forecasting and targeted policy retraining, marking a conceptual shift in how safety is validated under uncertainty.

JAIR Associate Editor: Prof. Akshat Kumar

JAIR Reference Format:
Anil Yildiz, Esen Yel, Marcell Vazquez-Chanlatte, Kyle Wray, Mykel J. Kochenderfer, and Stefan J. Witwicki. 2026. Backward Monte Carlo Tree Search: Charting Unsafe Regions in the Belief-Space. *Journal of Artificial Intelligence Research* 85, Article 1 (January 2026), 37 pages. DOI: 10.1613/jair.1.18011

1 Introduction

Safety-critical systems typically require rigorous validation of their behavior before deployment. These systems must provide robustness in environments that are often partially observable, making the safety assessment of the underlying policy challenging. Proper evaluation of these systems generally requires a comprehensive assessment of performance across the full spectrum of possible situations. While STAMP (System-Theoretic Accident Model and Processes) based guides (Leveson 2016), such as STPA and CAST, can serve as valuable

*Corresponding Author.

Authors' Contact Information: Anil Yildiz, ORCID: 0000-0001-8194-4895, yildiz@alumni.stanford.edu, Stanford Intelligent Systems Laboratory, Stanford University, Stanford, California, USA; Esen Yel, ORCID: 0000-0002-0463-3601, yele@rpi.edu, Reliable Intelligent Systems Laboratory, Rensselaer Polytechnic Institute, Troy, New York, USA; Marcell Vazquez-Chanlatte, ORCID: 0000-0002-1248-0000, marcell.vazquezchanlatte@nissan-usa.com, Nissan Advanced Technology Center, Silicon Valley, Santa Clara, California, USA; Kyle Wray, ORCID: 0000-0001-6986-9941, k.wray@northeastern.edu, Khoury College of Computer Sciences, Northeastern University, Boston, Massachusetts, USA; Mykel J. Kochenderfer, ORCID: 0000-0002-7238-9663, mykel@stanford.edu, Stanford Intelligent Systems Laboratory, Stanford University, Stanford, California, USA; Stefan J. Witwicki, ORCID: 0009-0005-3224-9198, stefan.witwicki@nissan-usa.com, Nissan Advanced Technology Center, Silicon Valley, Santa Clara, California, USA.

DOI: 10.1613/jair.1.18011

resources for exhaustively enumerating potential causes leading to unsafe states, they do not offer numerical representations or probabilities associated with these scenarios.

In the analysis of a safety critical system, the ability to pre-compute the conditions that can lead the system to unsafe states enable monitoring for undesired situations. Examples of an undesirable state for different domains include an autonomous car colliding with another vehicle or a pedestrian, a robot getting lost during exploration, a nuclear plant reaching a critical heat flux, and leakage during toxic waste clean-up (Cassandra 1998). Sequential decision-making problems for a safety-critical system with a known model in a fully observable environment can

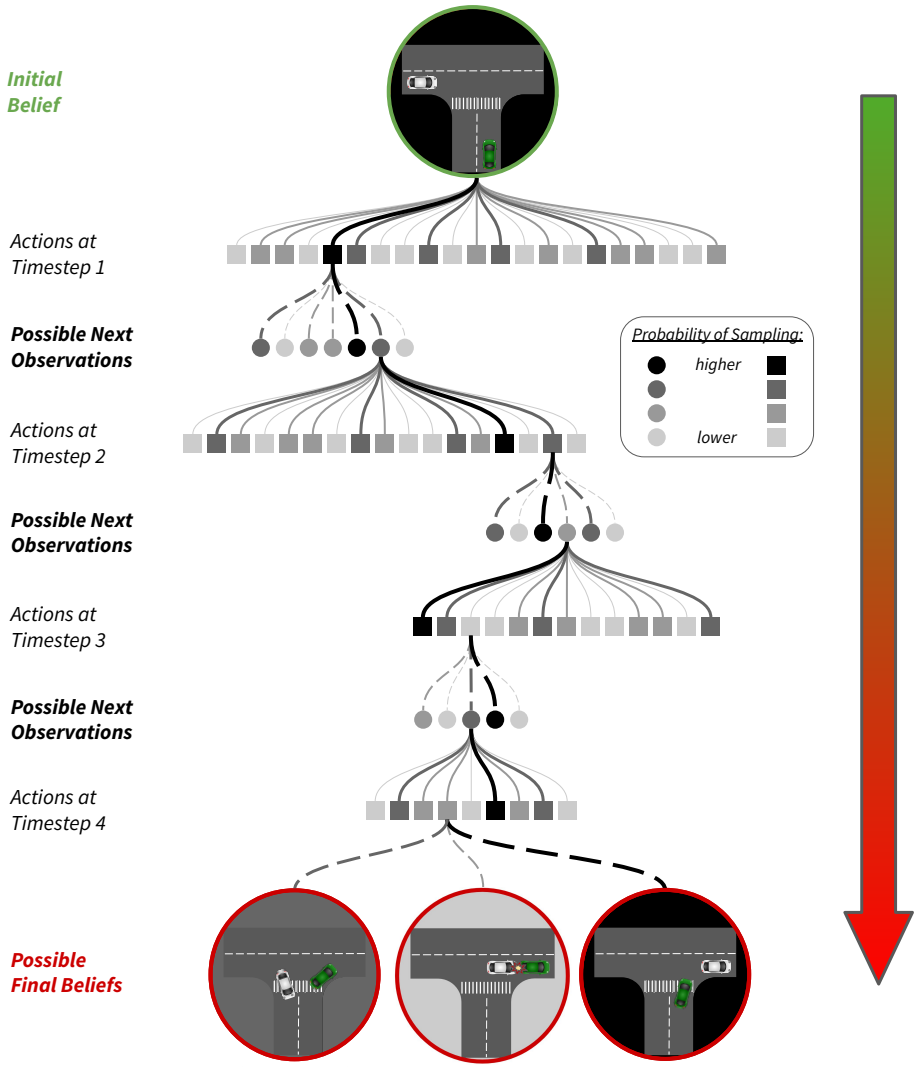


Fig. 1. The typical way of performing a Monte Carlo Tree Search (MCTS) is to start from an initial belief, and sample actions (square nodes) and observations (circle nodes) until a desired depth or a terminating condition is reached. Here, a tree starts from an initial belief and branches out to possible final beliefs, demonstrated for five timesteps forward in time.

be framed as a Markov decision process (MDP). When considering safety in the MDP settings, states that would reach an undesirable state under a given policy can be computed in a tractable way. One technique is to trace the agent's policy backward using Markov chain computations (Lindsten and Schön 2013). Even though these techniques are useful for safety validation, their applicability is limited because safety-critical systems commonly operate in partially observable environments. For example, the intentions of other vehicles and pedestrians, the extent of territory explored, power generation dynamics, and sources of leakage are not directly observable to the system.

In partially observable settings, the system infers the state of the environment indirectly through *observations*. Based on these observations, the agent maintains a *belief* over the state of the world, which is used to generate actions through a policy to achieve its goal. Such sequential decision making problem can be systematically framed as a partially observable Markov decision process (POMDP) (Kochenderfer, Wheeler, and Wray 2022). For the safety validation of such systems, it is useful to quantify the set of beliefs where the system's policy has a non-negligible probability of transitioning to an unsafe state within a certain number of timesteps. Determining this set of beliefs can be used to retrain the policy specifically for those beliefs. Furthermore, such a component in a real-time safety-critical system could warn the users to briefly override the policy and engage in manual control if found in such a belief. The lack of full observability makes it difficult to perform such a safety analysis. Techniques used for policy validation in MDP settings cannot be directly applied to POMDPs for the following three reasons: (1) the lack of complete observability introduces significant computational complexity in decision-making, (2) the belief-space in POMDPs can be continuous and very large, making it computationally challenging to scale to high-dimension state spaces, and (3) value functions and policies for POMDPs need to be functions of beliefs rather than the actual states, leading to increased complexity in value function approximations and policy optimizations.

To address these challenges, we are inspired by Monte Carlo tree search methods (Silver and Veness 2010) that are frequently used in partially observable settings due to their scalability and efficiency in branching. A typical Monte Carlo tree search starts from a single root node, and branches forward in time, top to bottom, into multiple leaf nodes to simulate transitions of the system across multiple timesteps. This process is shown in Figure 1, where actions and observations are selected/sampled at each layer with respect to a policy/distribution. The direction of the tree is from the green root node to the red leaf nodes, as indicated by the arrow. Each of the circular nodes correspond to a belief. In Figure 1, we are considering a self-driving car domain, where the ego vehicle (green) would like to make a right turn at an intersection when another vehicle (white) is present. As the tree branches forward in time, we reach certain leaf nodes for a desired depth. The biggest drawback of a forward search is the unlikeliness of reaching a low probability phenomenon of interest, such as a collision. To remedy this, this paper proposes an approach that specifically focuses on detecting beliefs that have a non-negligible probability of arriving at an undesired final state or belief of interest at any instance, given the underlying policy of the agent. Our tree search method constructs a tree backward in time, branching from a single leaf node to multiple root nodes, for the desired number of timesteps.[1] This concept is illustrated in Figure 2. The tree is constructed beginning with the green leaf node and progressing to the red root nodes, which is the reverse of the usual forward search. In the backward tree, we constrain our focus to a single final state or belief, and branch backward in time for a desired depth.

This approach ensures that the preceding branches of the tree are systematically conditioned to lead to the regarded final leaf node. The constructed tree yields a set of likely beliefs that reach the regarded undesired final belief within the specified number of timesteps. To our knowledge, our paper is the first to construct a Monte Carlo tree backward in time for partially observable domains.

The contributions of this study are as follows:

[1]As we expand the tree backward in time, we instantiate the construction of the tree from the bottom-most node, which we call the *leaf* node.

(1) The construction of a backward tree that is guided by a linear programming formulation to output a set of beliefs that eventually transition to an undesired state, given the policy of the agent. We assume that the policy can be represented by α-vectors (Roijers et al. 2015).
(2) Our formulation to find previous beliefs yields a face of a convex polygon from which we can sample. The computed polygon is a subset of, and has a significantly smaller dimension than, the entire belief-space.

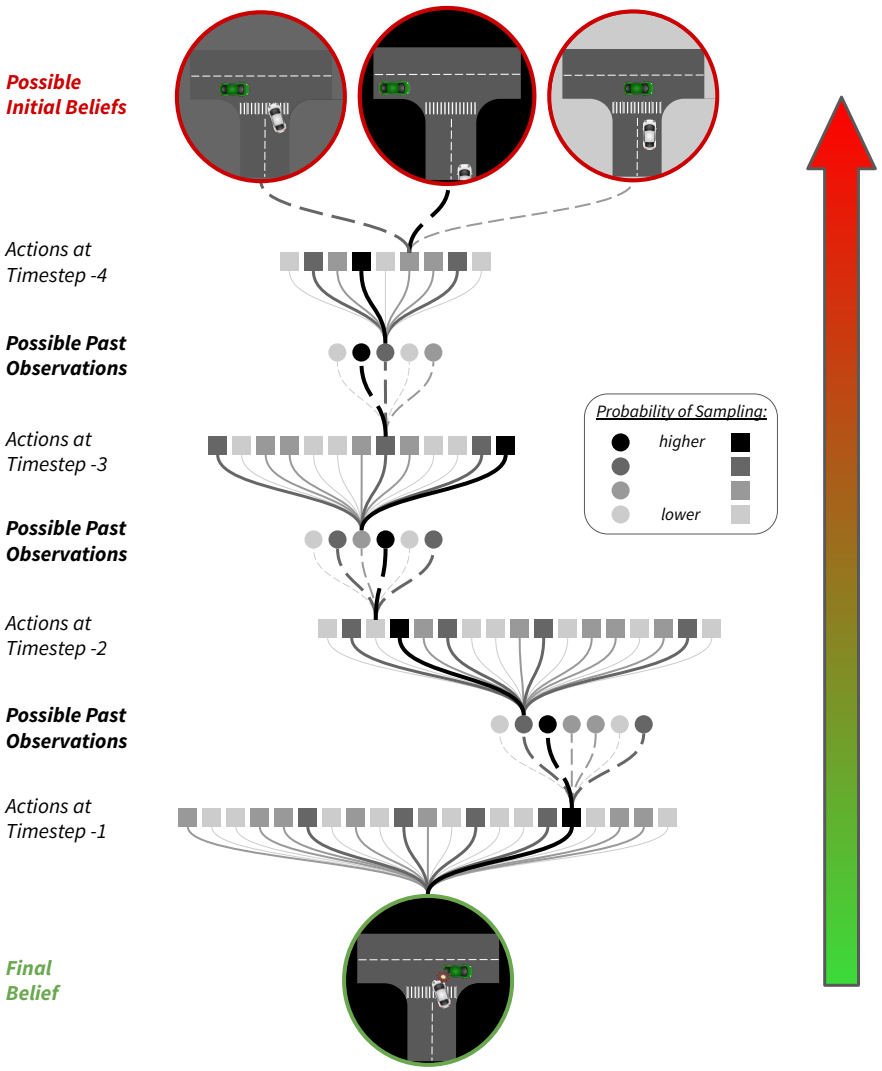


Fig. 2. In this paper, we propose a Backward MCTS that starts from a final belief node, and branches out to possible initial beliefs backward in time, demonstrated for five timesteps here. This backward tree returns a set of initial beliefs that have non-negligible probabilities of reaching the final root node of interest. (Some new additional intermediate nodes that are introduced in Section 5 are hidden for brevity.)

We use weighted sampling techniques to capture belief samples that efficiently span the belief region of interest.

(3) For these unsafe beliefs discovered, we propose an empirically accurate approximation of their probability of reaching an undesired state within the given number of timesteps.

(4) From the set of unsafe beliefs, we propose a generalized approach that can be used to estimate the probability of danger of any belief in real-time.

2 Related Work

To ensure safety in partially observable settings, there have been efforts that focus on synthesizing safe policies. Chance-constrained POMDPs (Huang et al. 2018; Rodrigues Quemel e Assis Santana et al. 2016) and risk-aware POMDPs (Hou et al. 2016; Kim et al. 2019) incorporate the notion of risk and cost into traditional POMDP formulation. Wang, Chaudhuri, et al. (2018) introduces goal-constrained belief-space to improve efficiency in computing a policy that satisfies a safe reachability objective and Wang, Newaz, et al. (2021) improves the scalability of this approach by introducing the concept of partial conditional plan that cover sample events to approximate a full conditional plan.

Another approach focuses formal methods for safe policy synthesis. Bouton et al. (2020) focus on synthesizing a policy with a maximum probability of satisfying an objective defined as a Linear Temporal Logic (LTL) formula (Pnueli 1977). To solve this problem in a tractable way, they frame it as a POMDP reward maximization problem and use SARSOP (Hanna Kurniawati 2008) to approximate a solution. Junges, Jansen, et al. (2021) focuses on computing winning regions from where a policy that satisfies reachability specification (i.e., the system reaches its goal without visiting unsafe states) exists. This winning region is computed in a backward manner, by adding states to an initial belief support states if there is a policy reaching these states without visiting unsafe ones. For safe exploration, a shield for POMDPs to block the actions that cause the system to leave this winning region is defined. Different from this work, we are interested in computing a set of beliefs that lead the system to certain belief regions under a given policy to determine the safety of this policy. The approach in our paper is tailored for real-time estimation of the probability of reaching unsafe states in partially observable domains. Our method operates during the execution phase of the autonomous policy, providing dynamic risk assessments that inform immediate decision-making.

In addition to policy synthesis, a body of work in the literature focuses on developing monitoring and validation techniques to assess whether a policy is safe for deployment. Rapidly-Exploring Random Trees (RRTs) (LaValle 1998) have traditionally been used for searching high-dimensional spaces while the tree expands toward unexplored regions. Backward-Forward Search (Garrett et al. 2015) combines forward and backward search where the backward search constructs a reachability graph from a goal region based on the sampled actions that drive the system to the goal region. Holtzen et al. (2021) introduce a probabilistic model-checking tool that uses decision diagrams to efficiently compute the probability of a target state being reached within a finite number of steps for a Markov Chain.

Partial observability has also been considered in policy validation and verification. Norman et al. (2017) focus on the problem of verifying that a temporal logic property holds for all policies in a discrete-time POMDP setting. The verification is performed similarly to the policy synthesis problem in which the POMDP is transformed to a fully observable continuous-space MDP and a solution is approximated based on a finite set of grid points following grid-based approximation techniques (Lovejoy 1991; Yu and Bertsekas 2004). Bork et al. (2020) also use results from Lovejoy (1991) and constructs an MDP whose optimal policy over-approximates the optimal policy in the POMDP setting to verify that the probability of reaching a bad state is below a given threshold for any policy. Ahmadi et al. (2018) treat POMDPs as switched systems and verify the safety and optimality of a given policy by computing barrier certificates using sum-of-squares programming. Bouton et al. (2020) use

temporal logic and a point-based algorithm to approximate the maximum probability of satisfying an objective for POMDP problems in a tractable way in addition to synthesizing the associated policy. Junges, Torfah, et al. (2021) introduce a runtime monitoring technique that uses conditional reachability probability computations to determine the state risk for partially observable systems with probabilistic and nondeterministic dynamics.

The primary shortcoming of the aforementioned method is that their assumption of a finite-state or automata-theoretic policy. As elaborated in the next sections, we use policies represented as alpha-vectors (Kochenderfer, Wheeler, and Wray 2022) that act directly on belief points. Alpha-vectors cannot be approximated by a finite state machine. E.g. there always exists arbitrarily many action/observation sequences that result in different actions being taken by the approximated and alpha-vector policy.

Our work is novel in the sense that it combines the following advantages: 1) applicability to partially observable domains, 2) use of a policy over beliefs directly, 3) explicit focus on the regions of the belief-space that lead to an undesired state, and 4) ability to validate the performance of a precomputed policy.

3 Background

This section covers some background topics that are relevant to the rest of the paper.

3.1 POMDPs

A sequential decision making problem can be modeled as a partially observable Markov decision process. A POMDP model is represented by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma \rangle$ where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, and $\mathcal{O}$ is a finite state of observations. The system takes an action $a \in \mathcal{A}$ from state $s \in \mathcal{S}$ and transitions to the next state $s' \in \mathcal{S}$ according to the probabilistic transition function $T(s' \mid s, a) = P(s' \mid s, a)$, which models the environment dynamics. From state s' the system obtains observation $o \in \mathcal{O}$ according to the observation function $Z(o \mid s') = P(o \mid s')$ and receives a reward according to the reward function $R(s, a)$.

Because the system does not have access to the true world states, it maintains a *belief* $\mathbf{b}(s)$, which is a vector representation of a discrete distribution over states. A discount factor $\gamma \in [0, 1)$ may also be used to prioritize earning rewards sooner than later.

After an action a is taken and an observation o is received, the belief is updated to $\mathbf{b}'_{(\mathbf{b},a,o)}$ using the following Bayesian relation:

$$\begin{aligned}
\mathbf{b}'_{(\mathbf{b},a,o)}(s) = \mathbf{b}'(s) &\triangleq P(s' \mid \mathbf{b}, a, o) \\
&= \sum_s P(s' \mid s, \mathbf{b}, a, o)\ P(s \mid \mathbf{b}, a, o) \\
&= \sum_s \frac{P(o \mid s', s, \mathbf{b}, a) P(s' \mid s, \mathbf{b}, a)}{P(o \mid s, \mathbf{b}, a)}\ P(s \mid \mathbf{b}, a, o) \\
&= \sum_s \frac{P(o \mid s') P(s' \mid s, a)}{P(o \mid s, \mathbf{b}, a)}\ P(s \mid \mathbf{b}) \\
&\propto Z(o \mid s') \sum_s T(s' \mid s, a)\ \mathbf{b}(s). \qquad (1)
\end{aligned}$$

3.2 Alpha Vectors

An alpha vector α_π is a vector representation of the expected utility when plan π is followed from different states. The optimal policy of the agent for a given belief is computed using:

$$\pi^*(\mathbf{b}) = \underset{\pi\,:\,\alpha_\pi \in \Gamma}{\arg\max}\ \alpha_\pi^\intercal \mathbf{b} \qquad (2)$$

where Γ is a set of of alpha vectors, each annotated with different actions.

The α-vectors that belong to the set Γ of a POMDP agent can be computed through various methods, such as QMDP (Hauskrecht 2000), FIB (Smith and Simmons 2005), PBVI (Shani et al. 2013). Other solution methods are reviewed by Kochenderfer, Wheeler, and Wray (2022).

In this work, we use α-vectors to describe the agent's policy because they (1) operate strictly under beliefs, and (2) have a direct mapping from a belief to an action. Surrogate models, constructed to mimic real-world dynamics in a cost-efficient way, usually accompany discrepancies from the actual system/environment behavior. Therefore, working with beliefs can increase the overall robustness to these modeling errors.

4 Problem Definition

The problem that we are interested in this paper is as follows:

Problem: Extract a subset from the entire belief-space such that, given a specific final belief $\mathbf{b}_\tau$, any initial belief $\mathbf{b}_0$ from this subset will reach $\mathbf{b}_\tau$ within τ timesteps through a specific sequence of actions and observations $(a_0, o_0, a_1, o_1, ...)$.

We assume that an undesired final state is fully observable.[2] That is, $\mathbf{b}_\tau$ is a concentrated belief into a single state. This is a reasonable assumption as for most autonomous robotics applications, an undesired state, such as the ego vehicle crashing, is robustly detectable.

Forward sampling-based methods frequently encounter difficulties in assessing whether a particular root node is close to a specific leaf belief. This limitation arises due to the fact that as the number of timesteps increases, the multiplication of probabilities across each timestep diminishes exponentially, approaching infinitesimal values. Consequently, the likelihood of the agent's sampled trajectory leading to the final belief $\mathbf{b}_\tau$ of interest becomes exceedingly low, rendering it practically infeasible. Techniques such as importance sampling (Bucklew 2004) may also be investigated. However, if samples from the proposal distribution used in importance sampling are not weighed appropriately, these resulting samples would be biased (Kochenderfer, Wheeler, and Wray 2022).

To counter the issues above, we propose a backward sampling method. In our study, we construct a tree from the sampled beliefs backward in time and assign a weight to each branch using their forward probabilities. The tree is constructed by carrying out these steps for τ layers to sample beliefs from a belief set with non-negligible probability of reaching a specific belief after τ steps.[3]

Constructing a backward Monte Carlo tree involves several nontrivial challenges. The next section formalizes the backward search as a series of convex optimization problems and elaborates the branching mechanism.

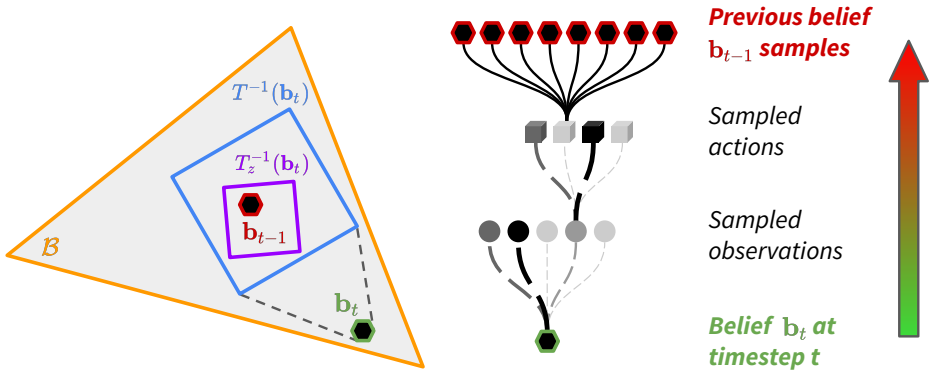
5 Backward Monte Carlo Tree Search

The construction of a backward Monte Carlo tree, as the name suggests, is executed backward in time. At any level, the tree aims to branch from a belief $\mathbf{b}_t$ at time t, to possible previous possible beliefs $\{\mathbf{b}_{t-1}\}$ at time $t-1$. Our approach is illustrated in Figure 3.

For a belief $\mathbf{b}_t$ at time t, we identify the region from the entire belief-space $\mathcal{B}$ where the possible previous possible beliefs $\{\mathbf{b}_{t-1}\}$ at time $t-1$ can reside. We denote this operation as $T^{-1}(\mathbf{b}_t; a_{t-1}, o_t,)$ given the action taken at $t-1$ and the observations received at t. As discussed in the remainder of our paper, we employ an intelligently guided strategy to sample both the actions and the observations as the backward tree is being built. $T^{-1}(\mathbf{b}_t)$ results in a convex polytope, which we can sample beliefs from, as depicted in Figure 3(a). However, some of the beliefs in $T^{-1}(\mathbf{b}_t)$ have very low likelihood of actually reaching $\mathbf{b}_t$. Therefore, we further narrow our focus of interest to a smaller polytope $T_z^{-1}(\mathbf{b}_t; a_{t-1}, o_t,)$ whose beliefs have a reachability probability to $\mathbf{b}_t$

[2] Though, this assumption can be relaxed, as discussed in Section B.

[3] Insights on choosing the right value of τ is discussed in Section C.



(a) (b)

Fig. 3. Overview of how a backward Monte Carlo tree is constructed.

above a threshold.[4] We can now sample previous beliefs $\mathbf{b}_{t-1} \sim T_z^{-1}(\mathbf{b}_t)$. We outline these steps on a backward tree in Figure 3(b). The backward tree construction starts from a leaf node $\mathbf{b}_t$. Possible observations that might have been received at time $t-1$ are sampled. These observation nodes are depicted as circular nodes. Next, we sample actions that the agent might have taken, that would lead to the succeeding observation and belief node. These action nodes are depicted as cube nodes. For any (a_{t-1}, o_t) path, we can formulate a linear program whose solution is the set of previous beliefs $T_z^{-1}(\mathbf{b}_t; a_{t-1}, o_t)$. Since this solution set is a convex polytope, we can then sample a previous belief $\mathbf{b}_{t-1}$. Sampled previous beliefs are depicted as hexagon nodes in Figure 3(b). The arrow in Figure 3(b) denotes the direction in which the backward tree is constructed.

The recurrence in Figure 3(b) is repeated, starting from every sampled $\mathbf{b}_{t-1}$, and finding possible beliefs $\{\mathbf{b}_{t-2}\}$. There is a single bottom-most leaf node in the entire backward tree, as depicted in Figure 2, which is the undesired final state or belief we wish to avoid. The tree is constructed for a desired depth. Algorithm 1 summarizes the steps involved, where the details for individual steps are elaborated in Section A.

The sampling procedure in BMCTS is guaranteed to converge in probability to the true reachability estimates as the number of simulations increases. Section D provides a theoretical discussion of these probabilistic convergence properties.

Algorithm 1 Construction of the backward Monte Carlo tree. Additional detail is provided in Section A.8.

while max depth is not reached **do**
  Sample an observation (Section A.1)
  Sample an action (Section A.2)
  Find region of possible previous beliefs (Sections A.3 to A.5)
  Sample a single belief from this region (Section A.6)
end while

While constructing the tree, we assume that the Luce Choice Axiom (Luce 2012) holds, which posits that the probability of selecting an item from a set is proportional to its utility and independent of irrelevant alternatives.

[4]This threshold is characterized by hyperparameter z (see Section A.4). This hyperparameter addresses the issue of diminishing probabilities across belief nodes in the tree by maintaining a balance between likelihood and coverage.

This axiom ensures that distinct choices (or transitions, in our context) retain their proportional likelihood without interference from unrelated factors. Similarly, in BMCTS, we assume that every (action, observation) pair leads to a distinct belief. This assumption simplifies the belief update process in POMDPs by treating belief transitions as distinct entities, each contributing proportionally to the belief dynamics. Ensuring logical distinctiveness aligns with the independence principle central to Luce's Axiom, where transitions maintain their individual probabilities in the belief-space.

This assumption is made primarily to ensure computational tractability. Without this assumption, equivalent (a, o) pairs would generate $|\mathcal{A}|^r$ indistinguishable paths, each contributing negligible information but significantly inflating computational requirements. In practice, the transition and observation dynamics, $\mathcal{T}$ and O respectively, particularly when learned from real-world datasets, naturally result in distinct belief updates for most (action, observation) pairs. This distinctiveness emerges from the probabilistic models used in real-world systems, making the assumption a realistic and practical approximation.

6 Generalizing the Backward Tree for Any Belief

Section 5 described how belief nodes are sampled for every timestep backward, thereby constructing a backward tree. In Section A.7, we detail how we approximate a belief node's (e.g. red nodes in Figure 2) probability of reaching the final leaf node (green node in Figure 2). We refer to this probability value as the *terminal Bayesian probability* of a belief. This process can be summarized as tracing the belief node's path in the tree and recursively computing and multiplying $p(\mathbf{b}', o \mid \mathbf{b}, a)$ after applying Bayesian belief updates. However, this method requires that the histories were explicitly realized within the tree for a given belief node. In this section, we discuss how this approximation can be extended to determine the terminal Bayesian probability for *any* belief in the belief-space $\mathcal{B}$. By doing so, we are able to generalize the identification of unsafe beliefs to the entire belief-space of the problem.

Our BMCTS approach assumes that by increasing the number of distinct branches in the backward tree, we can effectively cover the part of the belief-space that is of importance to us. Once there are sufficient belief nodes in the backward tree $\mathcal{T}$, we can make the following assumption:

$$\begin{aligned} \pi^*(\tilde{\mathbf{b}} \mid o_1, o_2, \dots) &= \pi^*(\mathbf{b} \mid o_1, o_2, \dots) \\ &= \{a_1, a_2, \dots\} \quad \forall \tilde{\mathbf{b}} \notin \mathcal{T} \\ \text{s.t.} \quad \mathbf{b} = \underset{\mathbf{b} \mid \{\mathbf{b},h\} \in \mathcal{T}.B}{\arg\min} & \; dist(\mathbf{b}, \tilde{\mathbf{b}}) \text{ where } h = \{a_1, o_1, a_2, o_2, \dots\}. \end{aligned} \tag{3}$$

In other words, any belief $\tilde{\mathbf{b}}$ that is not a node in $\mathcal{T}$, the sequence of actions that is closest to the optimal is approximated to be the same of the belief $\mathbf{b} \in \mathcal{B}$ having smallest distance to $\tilde{\mathbf{b}}$, given that the same observations are received. Here, we use the phrase *distance* loosely, which can incorporate domain knowledge, where a smaller distance signifies greater similarity between two beliefs. In this study, we use the L2 norm to compute the distance between two beliefs:

$$dist(\mathbf{b}, \tilde{\mathbf{b}}) \triangleq \|\mathbf{b} - \tilde{\mathbf{b}}\|_2. \tag{4}$$

By making the assumption in Equation (3), we are partitioning the entire belief-space, each cell denoting a specific history of actions and observations. These belief nodes become the *seeds* of a cell, creating a Voronoi diagram from the entire belief-space. Thus, every other belief falls into the Voronoi cell with the shortest distance from its seed. Per the assumption in Equation (3), we assume that every belief in each Voronoi cell follows the same history that will lead them to the leaf $\mathbf{b}_{final}$ of $\mathcal{T}$. As we empirically demonstrate in Section 7, the discrepancy from the actual action sequence followed is negligible (i.e. Equation (3) is a valid assumption) when there are a sufficient number of beliefs to effectively cover the entire belief-space. Additionally, Theorem 6.1

shows that the terminal Bayesian probability computed for a belief node in $\mathcal{T}$ is a strict lower bound on the true probability of reaching the final belief when observations differ from the recorded histories.

THEOREM 6.1. *The terminal Bayesian probability (see Section A.7) of the belief nodes in the tree is a lower bound to the probability of reaching the final belief while any observation can be observed.*

Proof. Let $\mathbf{b} \in \mathcal{B}$ be a belief, and $h \in (\mathcal{A} \times O)^n$ be an n-step history. Recall that for a fixed POMDP, Bayes' rule deterministically maps $(\mathbf{b}, h)$ to a new belief $\mathbf{b}'$, i.e.,

$$f : \mathcal{B} \times (\mathcal{A} \times O)^n \to \mathcal{B}.$$

As the backward tree is built, by construction, we enumerate a subset $\mathcal{Y}$ of $f^{-1}(\mathbf{b}_{\text{final}})$ where every path in the tree eventually leads to $\mathbf{b}_{\text{final}}$, i.e.,

$$\mathcal{Y} \leq f^{-1}(\mathbf{b}_{\text{final}}).$$

Then, the probability of reaching $\mathbf{b}_{\text{final}}$ for these two sets have the relation,

$$\sum_{\mathbf{b} \in \mathcal{Y}} p(\mathbf{b}) \leq \sum_{\mathbf{b} \in f^{-1}(\mathbf{b}_{\text{final}})} p(\mathbf{b}). \tag{5}$$

Following from Theorem 6.1, we conclude that the terminal Bayesian probabilities found for belief nodes in tree $\mathcal{T}$ are strict lower bound if observations different from the recorded histories had been received, which is empirically shown in Figure 4 in the next section. Furthermore, we can combine our Voronoi approximation method with Theorem 6.1 to account for any belief in the belief-space. By doing so, we have a method to approximate the lower bound for any belief's probability of reaching the undesired $\mathbf{b}_{final}$ during real-time monitoring.

Although lower bounds provide conservative risk estimates, they are highly effective in guiding policy retraining, as demonstrated in Section 7, by identifying scenarios where the current policy may fall short in ensuring safety. Notably, retraining the policy to improve its lower bound on risk guarantees that the agent achieves a level of safety that meets or exceeds the updated lower bound. This iterative process ensures consistent safety improvements, which is essential for systems requiring rigorous safety guarantees or measurable certifications.

We empirically show in Section 7 that this approximation is accurate when there are sufficient number of beliefs in $\mathcal{T}$ to representatively span $\mathcal{B}$, yielding a strict lower bound in real-time.

Discussion on Upper vs Lower Probability Bounds. While BMCTS yields a lower bound on the probability of reaching unsafe beliefs, this conservative estimate is advantageous in practice. By optimizing the policy to reduce this lower bound, we guarantee that the true risk does not increase, providing a monotonically non-decreasing safety guarantee. Consequently, BMCTS serves as a conservative screening mechanism that prioritizes safety improvements without requiring upper-bound computation.

Extension to Partially Observable Final States. In scenarios where the final state is only partially observable (e.g., a nearby vehicle collision unobserved by the ego vehicle), BMCTS can be extended by sampling a set of candidate final beliefs representing these events. Running BMCTS for each sampled belief allows estimation of reachability probabilities without requiring full observability. This conceptual extension demonstrates how BMCTS naturally generalizes to uncertain final states (see Section B for further discussion).

7 Results

In this section, we present our results from our Backward Monte Carlo Tree Search (BMCTS) approach. We aim to answer the following questions:

(1) What are the errors in reachability probabilities of the belief nodes in the backward tree?
(2) How many forward epochs are required to match the performance of the backward search?

(3) How do the mean relative and mean absolute errors scale with the distance from Voronoi centroids for random beliefs in the belief-space?
(4) How much improvement can be made on the agent's performance by retraining its policy using the undesired belief nodes found by the backward tree?

We first present our results in the partially observable toy gridworld domain. This first domain allows us to display easily interpretable results on how BMCTS works. Next, we present our results on an autonomous driving scenario, where the ego vehicle has to cross an intersection. This second domain allows us to exhibit a deployable use case for BMCTS.

7.1 Gridworld Domain

The gridworld domain is a partially observable environment where the agent is located at a single grid cell at any instance. The grid cells compose the state space of the domain. The agent takes one of the four main actions at each discrete timestep: {`left, right, up, down`}. Each cell in the gridworld has an associated reward, where a majority of the cells have a reward of zero. Our assumption is that for any cell having a non-zero reward, that specific state is terminal and is fully observable. This is aligned with our assumption from Section 4 that terminating states are fully observable.

The gridworld is 8×8 with 64 cells. Nonzero rewards are only located at the following cells: reward of +30 at $(4, 2)$, +20 at $(6, 7)$, −25 at $(2, 5)$, and −15 at $(7, 4)$. The reward structure is summarized in Section E.

We set the probabilities of successful transitions and observations to 70% and 90%, respectively. I.e. if the agent takes a `left` action, it will indeed translate to the grid cell to its left with a probability of 70%, otherwise, it will take one of the other three actions with an equal probability of 10%. If an inadmissible action is taken, e.g. a `right` when the agent is already positioned at the right-most column, then the agent's position will not change.

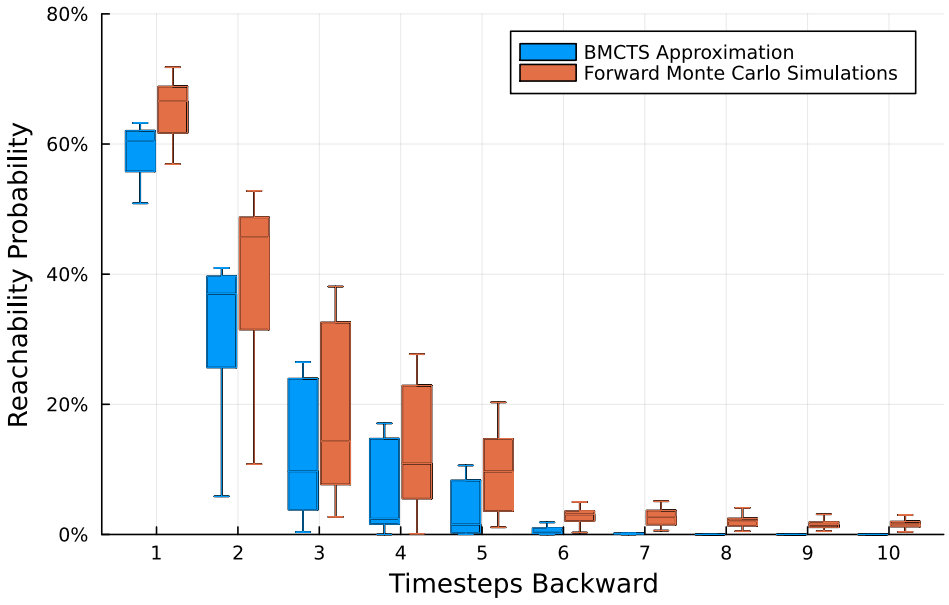


Fig. 4. Reachability probability of belief nodes in the tree for each backward timestep.

Similarly, the agent's real position is observed correctly 90% of the time, otherwise, one of the imminent neighbors is observed; if there are 8 neighbor grids, they are observed as the agent's state with an equal probability of 1.25%.

We construct two separate backward trees for this layout: the first one where the agent's goal is to reach the positive reward of +30 at cell $(4, 2)$, and a second one where the agent's goal is to avoid the negative reward of -25 at cell $(2, 5)$.

7.1.1 BMCTS with Desired Root Node. In this setting, we consider the case where the agent's goal is to reach the reward of +30 at cell $(4, 2)$; thus, the backward tree's leaf (bottom-most) node corresponds to this particular state. This final state is fully observable, as stated in Section 4. We construct a backward tree of 896 belief nodes in total, for a maximum of 10 timesteps backward in time.

Figure 4 presents the reachability probability values to the final desired state of $(4, 2)$ as boxplots, where the minimum, maximum, median, and upper/lower quartile values are depicted for the belief nodes at each backward timestep. The boxes for BMCTS approximation describe the reachability probability values where the history of actions and observations are prescribed by the edges in the tree. As proven in Equation (5), the reachability probability values found by BMCTS should be a lower bound to the case where *any* observations can be received. To empirically show this is the case, we benchmark the belief nodes in the tree against Monte Carlo simulations run forward in time. Each belief is simulated for 100,000 epochs using the agent's policy, and the empirical likelihood of reaching the final state $(4, 2)$ is recorded. As can be seen, the probability values that we compute through BMCTS with respect to the visited observations in the tree are a strict lower bound to any other observations that could have been received when the agent's policy is followed.

We also inspect discrepancy between the reachability probability approximated by BMCTS vs. the actual trajectories found when forward Monte Carlo simulations run using the histories determined by the tree (where

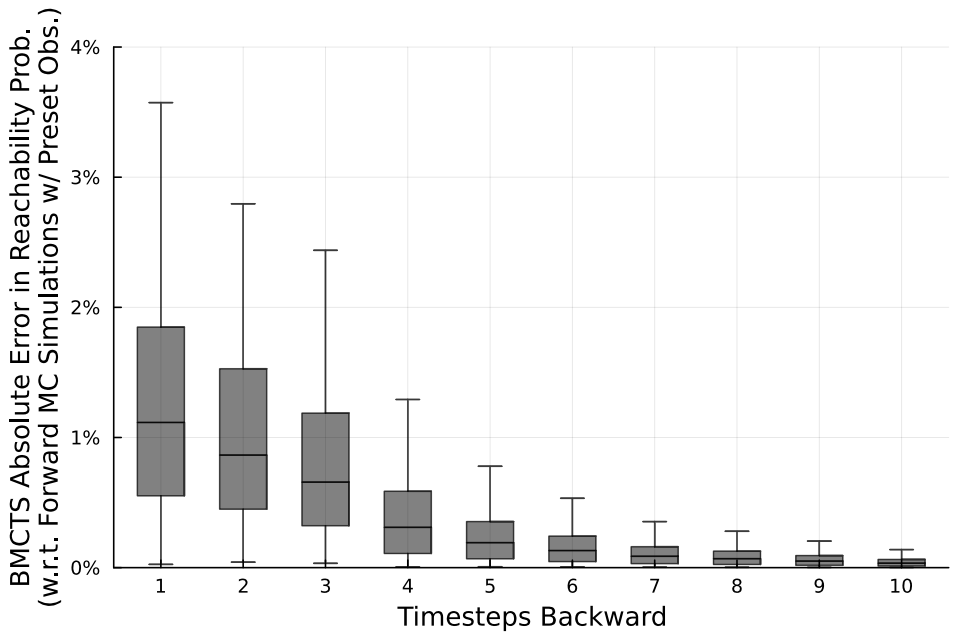


Fig. 5. Absolute error in reachability probability for belief nodes in the tree for each backward timestep. Error is computed against forward Monte Carlo simulations where the (a, o) sequence is determined by the tree.

a further analysis can be found in Section A.7). We present our results in Figure 5 where we plot the absolute errors in reachability probability for belief nodes in the tree at each timestep. The comparison of the BMCTS approximation was conducted against 100,000 epochs of forward Monte Carlo simulations, where the (a, o) histories were pre-determined by the tree, and the empirical likelihood of reaching the final state $(4, 2)$ is recorded. Results show that the median discrepancy is less than 1.5% for every timestep. We also observe a decrease in absolute error as the number of timesteps increases. This is an expected result since the likelihood of the agent reaching a specific goal decreases with higher number of timesteps due to the stochasticity and partial observability of the domain.

In Figure 6, we inspect how the number of epochs (how many forward simulations have been run) affects the empirical likelihood benchmarks computed in Figures 4 and 5. We observe that values more than 100,000 show no significant difference in terms of converging to the reachability probabilities found by BMCTS, justifying our choice of using 100,000 epochs for empirical approximations in Figures 4 and 5. In Figure 6, we also show, using the dashed line, the number of epochs where backward and forward search congruence: it is where the wallclock time taken to perform a single epoch of backward search is equal to the given number of forward search epochs.

Next, we inspect how the BMCTS approach works for *any* belief in the belief-space, using the Voronoi approach described in Section 6. We construct a Voronoi diagram where each belief node in $\mathcal{T}$ is a seed. Then, for each seed, we uniformly sample 1000 valid beliefs from multi-dimensional spheres (Muller 1959) of varying radii, where the centroids of the spheres are the seed beliefs. We plot the results as a heatmap in Figure 7. The mean relative errors in reachability probability are computed for various radii distanced from the seeds (x-axis), and are categorized by how many timesteps backward they correspond to (y-axis) before reaching cell $(4, 2)$. We can see from Figure 7 that the mean relative error rises as the distance from the Voronoi cell centroid increases. Furthermore, we observe that the relative error grows as the number of timesteps backward increases. These are both expected results, as there may be larger discrepancies between the approximated backward and the actual

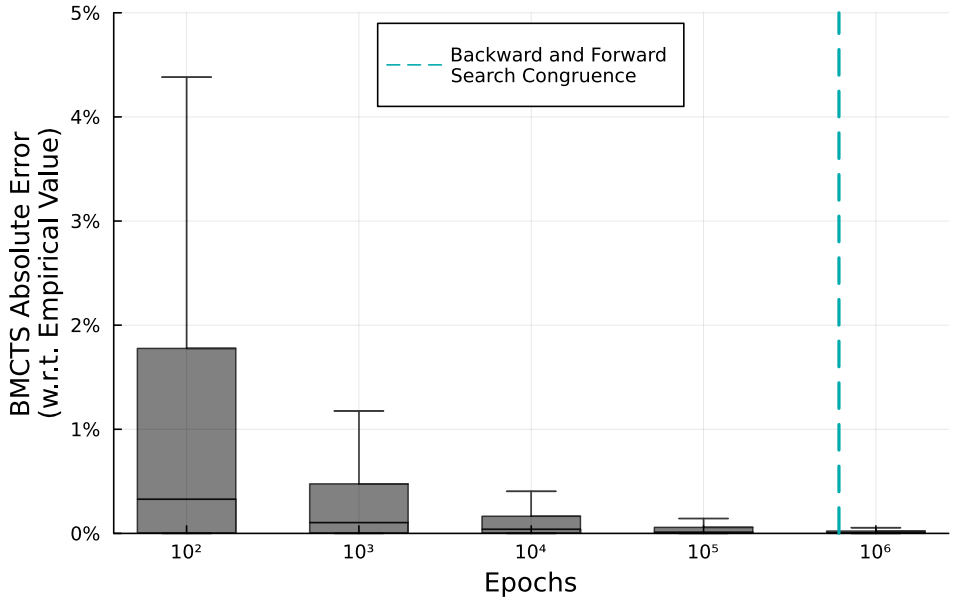


Fig. 6. Reachability probability values of each belief node in the tree. Nodes are sorted by their BMCTS values.

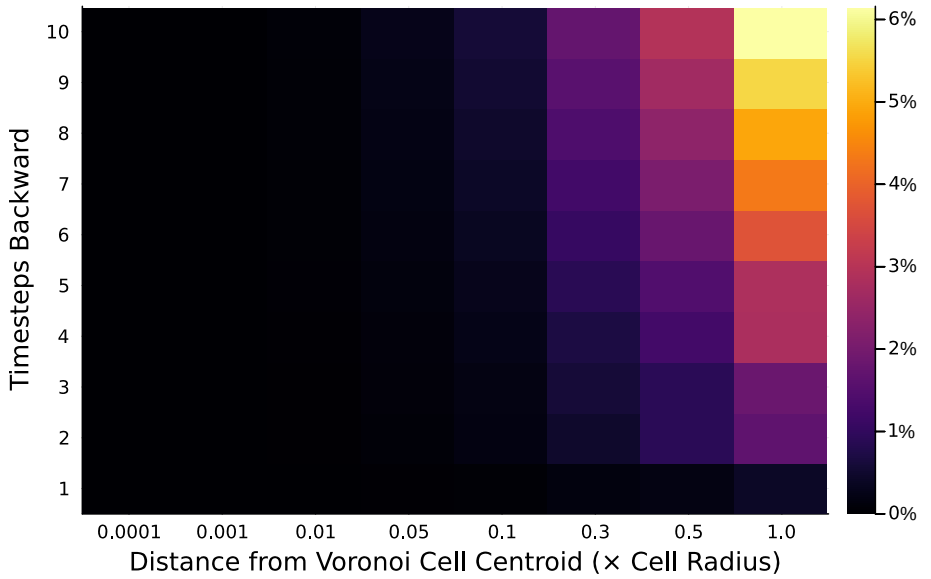


Fig. 7. Heatmap showing the mean relative errors of reachability probability for beliefs sampled at various distances from their corresponding Voronoi seed.

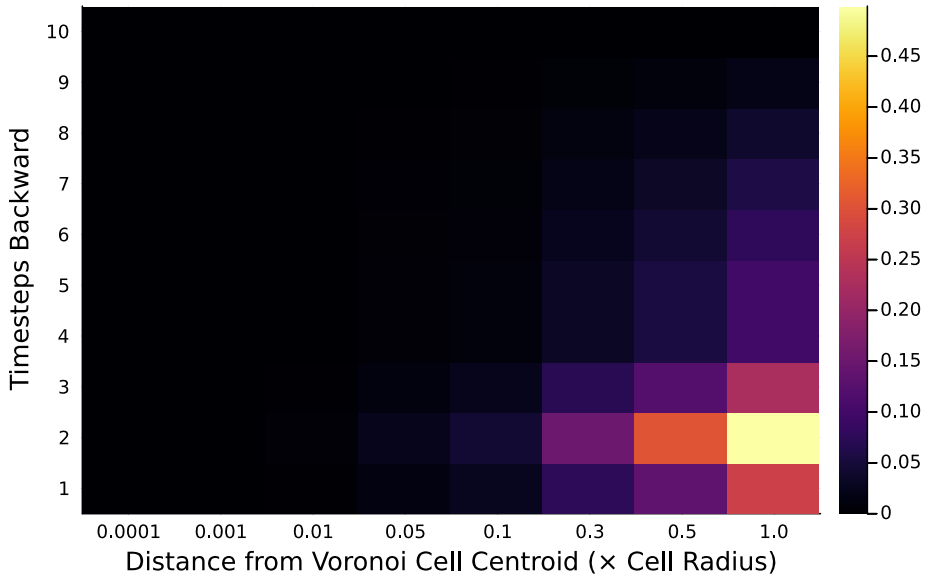


Fig. 8. Heatmap showing the mean absolute errors of reachability probability for beliefs sampled at various distances from their corresponding Voronoi seed.

forward trajectories when: (1) the sampled beliefs are farther from the nodes in the tree, and (2) there are more timesteps where Bayesian belief updates take place, thereby accumulating errors from previous timesteps.

We also examine the mean absolute errors of reachability probabilities, as seen in Figure 8. We observe that, similar to Figure 7, the error rises for each timestep backward in time as the distances from the Voronoi cell centroids increase. Nevertheless, the greatest error occurs for two timesteps backward in time. This suggests that the problem is most challenging when the agent is two timesteps away from cell $(4, 2)$, thus leading to the highest error at this timestep.

We suspect this is due to the fact that, at higher depths, the reachability probability is effectively zero, and thereby causing the lower bound to also be zero. Hence, the absolute errors are highest at lower depths.

Table 1. Reachability statistics for both the initial policy of the agent, and a retrained version of it.

			Reachability Statistics (lower is better)			
	Timesteps	Num. of Nodes	Minimum	Median	Mean	Maximum
Initial Policy	1	88	9.62%	20.81%	20.33%	29.09%
	2	147	1.03%	5.62%	5.67%	13.20%
	3	150	0.00%	1.93%	2.73%	33.25%
	4	250	0.00%	1.46%	2.05%	12.63%
	5	376	0.00%	0.87%	1.87%	13.31%
	6	356	0.00%	1.61%	2.93%	15.06%
	7	399	0.00%	3.51%	4.10%	11.96%
	8	450	0.11%	4.43%	4.50%	12.05%
	9	468	0.00%	4.77%	4.69%	12.83%
	10	503	0.23%	4.71%	4.84%	11.59%
Retrained Policy	1	88	**0.00%**	**0.12%**	**5.09%**	**27.61%**
	2	147	**0.00%**	**0.09%**	**0.24%**	**4.94%**
	3	150	**0.00%**	**0.00%**	$\mathbf{6 \times 10^{-3}\%}$	**0.22%**
	4	250	**0.00%**	**0.00%**	$\mathbf{4 \times 10^{-4}\%}$	**0.15%**
	5	376	**0.00%**	**0.00%**	**0.00%**	**0.00%**
	6	356	**0.00%**	**0.00%**	**0.00%**	**0.00%**
	7	399	**0.00%**	**0.00%**	**0.00%**	**0.00%**
	8	450	**0.00%**	**0.00%**	**0.00%**	**0.00%**
	9	468	**0.00%**	**0.00%**	**0.00%**	**0.00%**
	10	503	**0.00%**	**0.00%**	**0.00%**	**0.00%**

7.1.2 BMCTS with Undesired Root Node. In this setting, we consider the case where the leaf (bottom-most) node of the backward tree corresponds to cell $(2, 3)$ having reward -10. Thus, our focus is identifying cases where the agent performs off-policy and reaches this undesired state. We aim to retrain the agent's policy on under-performing belief regions to improve their performance on avoiding the undesired states. We show that the improvement in the policy can be generalized to any other belief in the belief-space.

We construct a new backward tree of 3187 belief nodes in total, for a maximum of 10 timesteps backward in time. We present our results in Table 1. For each timestep, we exhibit the minimum, median, mean, and maximum reachability probabilities of the belief nodes in the tree. The top half includes the statistics for the case where the tree is constructed when the agent has an initial generically trained policy. Our results indicate that, by

retraining the agent's policy over these discovered beliefs, we are able to significantly improve their performance by drastically decreasing reachability probabilities.[5]

Furthermore, the decrease in reachability probability through policy retraining can be generalized to *any* other belief in the belief-space. Similar to before, we uniformly sample 1000 valid beliefs from multi-dimensional spheres centered at tree nodes. Our samples constitute of distances $r = \{0.0001, 0.01, 0.05, 0.1, 0.3, 0.5, 1.0\}$ times each Voronoi cell's radii. We find that the median and mean improvement are 4.69% and 8.04%, respectively. The variance in median reachability probabilities with respect to sample distance r and number of timesteps are 6×10^{-4}% and 0.29%, respectively. The variance in mean reachability probabilities with respect to sample distance r and number of timesteps are 4×10^{-3}% and 0.25%, respectively. These results suggest that performance improvements do not show a strong correlation to distance r or the number of timesteps backward.

Qualitatively, the identified belief nodes in the backward tree correspond to positions where the agent, if following its policy, risks entering undesired zones:

(1) **Proximity to obstacles:** Beliefs representing positions near walls or other blocking elements where a misstep at any point in time could lead to high penalties in the future.
(2) **Ambiguous transitions:** Beliefs where multiple actions may result in transitioning into unsafe areas due to stochastic dynamics or partial observability.

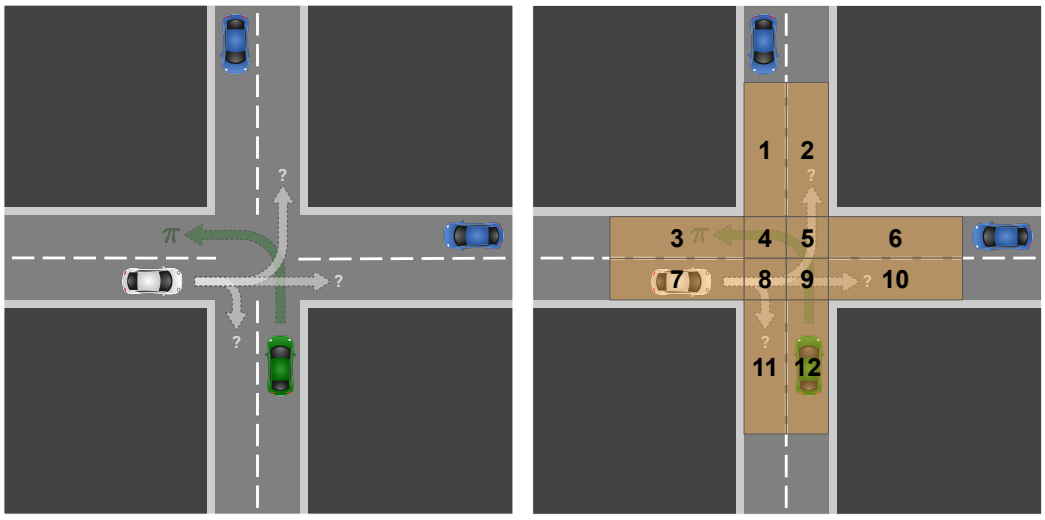
7.2 Autonomous Car Domain

We now present our results in a more realistic scenario of an autonomous driving car passing through a 4-way intersection. Results from our backward tree search may be used as a deployable real-time safety warning system, where the driver may be alerted when the ego vehicle has detected a non-negligible crash probability. The problem is illustrated in Figure 9(a). As the ego vehicle (green) attempts to make a left turn in a 4-way intersection, there are other vehicles in the vicinity. At the timeframe of illustration in Figure 9(a), only the white vehicle is in proximity to the intersection, and the agent needs to infer its intention and the associated risk of crashing.

We frame this problem as a POMDP. We model each ego-other vehicle interaction as an individual POMDP instantiation, which can then be centralized that takes the safest action across all instantiations (Wray et al. 2017). We discretize the possible locations that the ego and other vehicles can be in to boxes from a top down view perspective. We denote a total of 12 boxes from the viewpoint of the ego vehicle, as illustrated in Figure 9(b). We also consider the speed of which the other vehicle is moving within a box, discretized to be one the following values: {`slow, moderate, fast`}. These speed definitions are based on expert knowledge to categorize actual continuous values into these three groups. The intention of the other vehicle can also be discretized to be one of the following values (with respect to the ego vehicle): {`left, right, straight`}. Thus, the state space of the POMDP consists of the follow features: the box location of the ego and other vehicle of interest, velocity range of the other vehicle, and intention of the other vehicle. We simulate the problem with a frequency of 10 Hz. At each timestep instance, the POMDP is only concerned with the longitudinal control and has three actions that it can take: {`accelerate, coast, brake`}. The ego vehicle receives noisy observations over the features of the other vehicle's true state. We simulate 100,000 scenarios in a lightweight autonomous driving simulator, Carlo (Cao et al. 2020), to form an empirical likelihood over the transition and observation functions of the POMDP. The deployment of a POMDP framed in this manner has been shown to work well in real-world intersection crossing scenarios (Yildiz et al. 2023).

We create a backward tree with 5261 nodes. We are considering the case were the ego and another vehicle both being in box 9 shown in Figure 9(b). The backward tree is branched out from multiple leaf nodes where we

[5]For both the gridworld domain and the autonomous car domain described in Section 7.2, policy retraining was conducted using point-based value iteration (Kochenderfer, Wheeler, and Wray 2022). The retraining process specifically targeted unsafe beliefs identified through BMCTS. Section F provides the algorithm used for policy retraining.



(a) (b)

Fig. 9. a) The ego vehicle (green) would like to make a left turn through the 4-way intersection. To safely do so, it has reason about the intention of the vehicles in its vicinity (only the white car here), which can be modeled as a POMDP. b) For our experiments, we discretize the locations other vehicles might be with respect to the ego vehicle into 12 boxes.

sample states that correspond to different speeding values and ground-truth intentions of the other vehicle. We branch out 20 timesteps backward in time. Since the problem is simulated to a frequency of 10 Hz, this allows us to look back at beliefs 2.0 seconds before a collision. Our results are presented in Table 2. We display the minimum, median, mean and maximum reachability probabilities of the belief nodes in the tree, for different times (in seconds) backward prior to a collision. The upper half of Table 2 shows the statistics for the tree built when the agent has an initial policy that has been trained before the tree is generated. Our findings suggest that by retraining the agent's policy on the beliefs that have been identified through the backward tree, we are able to significantly enhance the performance of the car by greatly reducing the collision (reachability) probabilities into the future.[6] Our study demonstrates the effectiveness of the proposed approach in enhancing autonomous vehicle safety, providing a foundation for further advancements in real-world complex traffic interactions.

Typically, the identified beliefs are associated with scenarios where the ego agent is at risk of causing or being involved in a collision. These include:

(1) **Near collisions:** Beliefs where the car is in close proximity to other vehicles, especially in scenarios requiring precise maneuvers.
(2) **Uncertainty in sensor data:** Beliefs where observation noise significantly impacts the agent's ability to accurately infer the state, increasing the likelihood of risky decisions.

[6]Reward structures (the same used for the pre-retrained policy) are implemented to discourage behaviors that hinder task completion, such as staying put to de-risk collision. For example, we implement reward structures that penalize inaction or unnecessary delays, encouraging the agent to continue progressing towards its goals while maintaining safety. Refer to Section E for detailed definitions of rewards, and Section F for information on policy retraining.

Table 2. Reachability statistics for both the initial and retrained policies. The nodes in each time period indicate the amount of seconds to be elapsed prior to a collision.

			Reachability Statistics (lower is better)			
	Time (s)	Num. of Nodes	Minimum	Median	Mean	Maximum
Initial Policy	0.1	49	34.53%	60.26%	61.62%	77.01%
	0.2	71	27.92%	40.27%	41.81%	50.66%
	0.3	94	22.18%	29.79%	31.06%	35.62%
	0.4	116	19.67%	23.31%	23.34%	26.58%
	0.5	139	16.84%	20.39%	19.83%	21.28%
	0.6	161	15.30%	17.15%	18.15%	19.57%
	0.7	184	12.86%	16.73%	16.19%	15.91%
	0.8	207	12.50%	14.20%	15.25%	14.34%
	0.9	229	11.06%	13.09%	13.81%	13.06%
	1.0	252	9.84%	12.62%	12.57%	11.95%
	1.1	274	8.86%	11.25%	10.79%	11.43%
	1.2	297	7.61%	10.61%	9.93%	10.43%
	1.3	319	6.96%	9.41%	9.13%	9.10%
	1.4	343	6.20%	8.54%	8.98%	9.09%
	1.5	365	5.52%	7.67%	8.36%	8.20%
	1.6	387	4.82%	7.49%	7.50%	7.54%
	1.7	410	4.30%	6.72%	6.53%	6.49%
	1.8	432	3.92%	6.19%	6.35%	6.43%
	1.9	455	3.43%	5.77%	5.50%	5.95%
	2.0	477	3.24%	5.03%	5.06%	5.49%
Retrained Policy	0.1	49	**9.82%**	**35.03%**	**34.53%**	**52.33%**
	0.2	71	**3.69%**	**14.79%**	**16.69%**	**25.46%**
	0.3	94	**1.46%**	**6.75%**	**7.81%**	**12.90%**
	0.4	116	**0.60%**	**3.02%**	**3.48%**	**6.42%**
	0.5	139	**0.23%**	**1.41%**	**1.55%**	**3.07%**
	0.6	161	**0.09%**	**0.64%**	**0.73%**	**1.59%**
	0.7	184	**0.04%**	**0.28%**	**0.33%**	**0.76%**
	0.8	207	**0.01%**	**0.13%**	**0.15%**	**0.38%**
	0.9	229	$\mathbf{5 \times 10^{-3}\%}$	**0.06%**	**0.07%**	**0.19%**
	1.0	252	$\mathbf{3 \times 10^{-3}\%}$	**0.02%**	**0.03%**	**0.09%**
	1.1	274	$\mathbf{8 \times 10^{-4}\%}$	**0.01%**	**0.02%**	**0.05%**
	1.2	297	$\mathbf{3 \times 10^{-4}\%}$	$\mathbf{5 \times 10^{-3}\%}$	**0.01%**	**0.02%**
	1.3	319	$\mathbf{1 \times 10^{-4}\%}$	$\mathbf{3 \times 10^{-3}\%}$	$\mathbf{3 \times 10^{-3}\%}$	**0.01%**
	1.4	343	$\mathbf{5 \times 10^{-5}\%}$	$\mathbf{9 \times 10^{-4}\%}$	$\mathbf{2 \times 10^{-3}\%}$	**0.01%**
	1.5	365	$\mathbf{1 \times 10^{-5}\%}$	$\mathbf{4 \times 10^{-4}\%}$	$\mathbf{7 \times 10^{-4}\%}$	$\mathbf{3 \times 10^{-3}\%}$
	1.6	387	$\mathbf{7 \times 10^{-6}\%}$	$\mathbf{2 \times 10^{-4}\%}$	$\mathbf{3 \times 10^{-4}\%}$	$\mathbf{1 \times 10^{-3}\%}$
	1.7	410	$\mathbf{2 \times 10^{-6}\%}$	$\mathbf{9 \times 10^{-5}\%}$	$\mathbf{1 \times 10^{-4}\%}$	$\mathbf{6 \times 10^{-4}\%}$
	1.8	432	$\mathbf{1 \times 10^{-6}\%}$	$\mathbf{4 \times 10^{-5}\%}$	$\mathbf{6 \times 10^{-5}\%}$	$\mathbf{3 \times 10^{-4}\%}$
	1.9	455	$\mathbf{4 \times 10^{-7}\%}$	$\mathbf{1 \times 10^{-5}\%}$	$\mathbf{3 \times 10^{-5}\%}$	$\mathbf{1 \times 10^{-4}\%}$
	2.0	477	$\mathbf{1 \times 10^{-7}\%}$	$\mathbf{8 \times 10^{-6}\%}$	$\mathbf{1 \times 10^{-5}\%}$	$\mathbf{8 \times 10^{-5}\%}$

8 Conclusions

In this study, we introduced and evaluated Backward Monte Carlo Tree Search (BMCTS) to address the challenge of charting unsafe belief regions in the belief-space. We construct a backward tree by consecutively sampling from the pre-image of belief transitions. The resulting nodes in the backward tree can then be leveraged to reason about any belief in the entire belief-space of the problem, demonstrating empirical accuracy. We illustrated our approach in two different domains.

For the gridworld domain, we demonstrated the effectiveness of BMCTS in finding a tight lower bound that tracks forward simulations. Our results indicated low absolute errors in reachability probabilities for belief nodes in the backward tree, highlighting the accuracy of the BMCTS approach. We also explored the impact of the number of epochs chosen for forward simulations, showing that values above 100,000 (our chosen value for verification) did not significantly affect convergence.

Additionally, we applied BMCTS to an autonomous driving scenario, specifically a 4-way intersection. The results showcased the adaptability of BMCTS in a real-world, deployable use case. By framing the problem as a Partially Observable Markov Decision Process (POMDP), discretizing the state space, and simulating scenarios in a lightweight autonomous driving simulator, Carlo, we successfully built a backward tree that provided valuable insights into collision risk reduction.

For both of these domains, our study delved into the retraining of the agent's policy based on beliefs identified through the backward tree. For scenarios where the agent aimed to reach an undesired state, retraining the policy effectively decreased the reachability probabilities of unsafe regions. In the gridworld domain, retraining resulted in a significant reduction in the likelihood of entering cells with high negative rewards. In the autonomous car domain, retraining reduced collision probabilities substantially, enhancing safety.

Moreover, the methods used for policy modification based on undesirable belief states have broader applications beyond safety improvements. For instance, they can refine notification systems, such as for aircraft collision avoidance, by optimizing alerts based on the understanding of undesirable belief states. These techniques can also enhance forward-search algorithms by identifying rare but high-value outcomes, significantly improving planning performance. Additionally, they can be applied to validate the safety of agents that cannot be modified, such as those sourced externally or from adversarial systems, by analyzing belief states that might lead to harmful consequences. These diverse applications highlight the versatility and impact of BMCTS in improving decision-making, ensuring safety, and optimizing system performance across various domains.

Overall, our findings highlight the efficacy of the BMCTS approach in addressing complex decision-making problems in partially observable domains. The ability to provide a deployable real-time safety warning system and the potential for policy retraining based on backward tree insights make BMCTS a promising tool for enhancing the performance of autonomous systems. Future work should explore further applications, scalability to larger and continuous domains, optimization of the retraining process, and balancing competing objectives in real-world implementations.

Acknowledgments

The research reported in this work was supported by Nissan Advanced Technology Center, Silicon Valley, Nissan North America, Inc. (1258649-2-UDBDS).

References

M. Ahmadi, M. Cubuktepe, N. Jansen, and U. Topcu. 2018. "Verification of uncertain POMDPs using barrier certificates." In: *Allerton Conference on Communication, Control, and Compution*, 115–122.

P. Auer, N. Cesa-Bianchi, and P. Fischer. 2002. "Finite-time analysis of the multiarmed bandit problem." *Machine Learning*, 47, 235–256.

D. Barber. 2012. *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

A. Bork, S. Junges, J.-P. Katoen, and T. Quatmann. 2020. "Verification of Indefinite-Horizon POMDPs." In: *Automated Technology for Verification and Analysis*, 288–304.
M. Bouton, J. Tumova, and M. J. Kochenderfer. 2020. "Point-based methods for model checking in partially observable Markov decision processes." In: *AAAI Conference on Artificial Intelligence (AAAI)*.
J. A. Bucklew. 2004. *Introduction to Rare Event Simulation*. Springer.
Z. Cao, E. Biyik, W. Z. Wang, A. Raventos, A. Gaidon, G. Rosman, and D. Sadigh. 2020. "Reinforcement Learning based Control of Imitative Policies for Near-Accident Driving." In: *Robotics: Science and Systems*.
A. R. Cassandra. 1998. "A survey of POMDP applications." In: *AAAI Fall Symposium on Planning with Partially Observable Markov Decision Processes*.
A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. 2011. "Continuous upper confidence trees." In: *International Conference on Learning and Intelligent Optimization (LION)*, 433–445.
I. Dunning, J. Huchette, and M. Lubin. 2017. "JuMP: A modeling language for mathematical optimization." *SIAM Review*, 59, 2, 295–320.
C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. 2015. "Backward-forward search for manipulation planning." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6366–6373.
Gurobi Optimization, LLC. 2022. *Gurobi Optimizer Reference Manual*. (2022). https://www.gurobi.com.
W. S. L. Hanna Kurniawati David Hsu. 2008. "SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces." In: *Robotics: Science and Systems*.
M. Hauskrecht. 2000. "Value-function approximations for partially observable Markov decision processes." *Journal of Artificial Intelligence Research*, 13, 33–94.
S. Holtzen, S. Junges, M. Vazquez-Chanlatte, T. Millstein, S. A. Seshia, and G. Van den Broeck. 2021. "Model Checking Finite-Horizon Markov Chains with Probabilistic Inference." In: *International Conference on Computer-Aided Verification (CAV)*, 577–601.
P. Hou, W. Yeoh, and P. Varakantham. 2016. "Solving Risk-Sensitive POMDPs With and Without Cost Observations." *AAAI Conference on Artificial Intelligence (AAAI)*.
X. Huang, A. Jasour, M. Deyo, A. Hofmann, and B. C. Williams. 2018. "Hybrid Risk-Aware Conditional Planning with Applications in Autonomous Vehicles." In: *IEEE Conference on Decision and Control (CDC)*, 3608–3614.
S. Junges, N. Jansen, and S. A. Seshia. 2021. "Enforcing Almost-Sure Reachability in POMDPs." In: *International Conference on Computer-Aided Verification (CAV)*, 602–625.
S. Junges, H. Torfah, and S. A. Seshia. 2021. "Runtime Monitors for Markov Decision Processes." In: *International Conference on Computer-Aided Verification (CAV)*, 553–576.
S.-K. Kim, R. Thakker, and A.-A. Agha-Mohammadi. 2019. "Bi-directional value learning for risk-aware planning under uncertainty." *IEEE Robotics and Automation Letters*, 4, 3, 2493–2500.
M. J. Kochenderfer and T. A. Wheeler. 2019. *Algorithms for Optimization*. MIT Press.
M. J. Kochenderfer, T. A. Wheeler, and K. H. Wray. 2022. *Algorithms for Decision Making*. MIT Press.
L. Kocsis and C. Szepesvári. 2006. "Bandit based monte-carlo planning." In: *European Conference on Machine Learning (ECML)*, 282–293.
S. LaValle. 1998. "Rapidly-exploring random trees: A new tool for path planning." *Computer Science Dept., Iowa State University*, 98, 11.
N. G. Leveson. 2016. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press.
F. Lindsten and T. B. Schön. 2013. "Backward Simulation Methods for Monte Carlo Statistical Inference." *Foundations and Trends in Machine Learning*, 6, 1, 1–143.
W. S. Lovejoy. 1991. "Computationally Feasible Bounds for Partially Observed Markov Decision Processes." *Operations Research*, 39, 1, 162–175.
R. D. Luce. 2012. *Individual Choice Behavior: A Theoretical Analysis*. Courier Corporation.
M. E. Muller. 1959. "A note on a method for generating points uniformly on n-dimensional spheres." *Communications of the ACM*, 2, 4, 19–20.
G. Norman, D. Parker, and X. Zou. 2017. "Verification and control of partially observable probabilistic systems." *Real-Time Systems*, 53, 354–402.
A. Pnueli. 1977. "The temporal logic of programs." In: *Symposium on Foundations of Computer Science*, 46–57.
P. Rodrigues Quemel e Assis Santana, S. Thiébaux, and B. Williams. 2016. "RAO*: An Algorithm for Chance-Constrained POMDPs." *AAAI Conference on Artificial Intelligence (AAAI)*.
D. M. Roijers, S. Whiteson, and F. A. Oliehoek. 2015. "Point-based planning for multi-objective POMDPs." In: *International Joint Conference on Artificial Intelligence (IJCAI)*, 1666–1672.
G. Shani, J. Pineau, and R. Kaplow. 2013. "A survey of point-based POMDP solvers." *Autonomous Agents and Multi-Agent Systems*, 27, 1–51.
D. Silver, T. Hubert, et al.. 2017. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm." *arXiv preprint arXiv:1712.01815*.
D. Silver and J. Veness. 2010. "Monte-Carlo planning in large POMDPs." *Advances in Neural Information Processing Systems (NIPS)*.
T. Smith and R. Simmons. 2005. "Point-based POMDP algorithms: improved analysis and implementation." In: *Conference on Uncertainty in Artificial Intelligence (UAI)*, 542–549.
Y. Wang, S. Chaudhuri, and L. E. Kavraki. 2018. "Bounded Policy Synthesis for POMDPs with Safe-Reachability Objectives." In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 238–246.

Y. Wang, A. A. R. Newaz, J. D. Hernández, S. Chaudhuri, and L. E. Kavraki. 2021. "Online Partial Conditional Plan Synthesis for POMDPs With Safe-Reachability Objectives: Methods and Experiments." *IEEE Transactions on Automation Science and Engineering*, 18, 3, 932–945.

K. H. Wray, S. J. Witwicki, and S. Zilberstein. 2017. "Online Decision-Making for Scalable Autonomous Systems." In: *International Joint Conference on Artificial Intelligence (IJCAI)*, 4768–4774.

A. Yildiz, E. Yel, A. L. Corso, K. H. Wray, S. J. Witwicki, and M. J. Kochenderfer. 2023. "Experience Filter: Using Past Experiences on Unseen Tasks or Environments." In: *IEEE Intelligent Vehicles Symposium (IV)*, 1–7.

H. Yu and D. P. Bertsekas. 2004. "Discretized Approximations for POMDP with Average Cost." In: *Conference on Uncertainty in Artificial Intelligence (UAI)*, 619–627.

A Steps for Constructing the Backward Tree

This appendix covers each step involved in the construction of the backward tree, as outlined in Algorithm 1.

A.1 Sampling Observations

Observation nodes are sampled independently for each belief node as depicted in Figure 3(b). The general idea is that, for the given next belief $\mathbf{b}_t$, the observations in the previous time step can only be observed from the states with non-zero entries in $\mathbf{b}_t$. We define the entries of the next belief $\mathbf{b}_t$ as $\{P(s'_1), P(s'_2), \dots\}$. Note that these are probability values, hence $0 \leq P(s'_k) \leq 1,\ \forall k$ and $\sum_k P(s'_k) = 1$. We compute the probability of each observation as:

$$P(o) = \sum_k Z(o \mid s'_k) P(s'_k) \quad \forall o \in O. \tag{6}$$

Notice that the operation above can be easily vectorized. During the construction of the tree, we sample observations nodes preceding $\mathbf{b}_t$ with respect to their probabilities $P(o)$.

A.2 Sampling Actions: Backward UCT Exploration

There are variations of the Monte Carlo tree search algorithm to effectively manage large state, action, and observation spaces. Instead of expanding all available actions, action branches can be progressively widened. In our backward tree search, we employ an approach similar to upper confidence trees (UCT) (Couëtoux et al. 2011) during action branching.[7] We incorporate the following UCB1 (Auer et al. 2002) strategy:

$$P(a \mid oh) = Q(aoh) + k_{\text{ucb}} \sqrt{\frac{\log N(oh)}{N(aoh)}} \tag{7}$$

where o is a sampled observation, $Q(h)$ is the expected terminal Bayesian probability (see Section A.7) of an action-observation history h, and $N(h)$ counts the number of times h has been visited in the tree. Here, the expression aoh refers to history h expanded backward in time, with action a and observation o preceding it; i.e. after action a is taken and observation o is received, history h occurs. Similarly, oh refers to history h expanded backward in time, with observation o preceding it, and is not attached to any specific action. The scalar hyperparameter k_{ucb} balances exploration and exploitation.

Once the tree is branched using a sampled observation and with the action that solves Equation (7), the previous belief nodes are computed using a linear program formulation, as explained in detail in the next section.

A.3 Formulating a Backward Step as a Linear Program

Given the sampled observation and selected action, we compute previous beliefs ($\mathbf{b}_{t-1}$ nodes depicted in Figure 3(b)) using a linear program formulation. In this problem, we assume that we are given the belief $\mathbf{b}_t$ at timestep t and

[7] Breadth-first search provides better coverage at the expense of directed search, whereas, depth-first search does directed search and has better memory properties but has the downside of not propagating information gain in the opposite direction. UCT is designed to balance both approaches.

agent policy represented by Γ containing alpha-vectors $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_{|\Gamma|}$. For a sampled observation o_i, and selected action a^j corresponding to the alpha-vector $\boldsymbol{\alpha}_j \in \Gamma$, we formulate our linear program as follows:

$$\min_{\mathbf{x},\mathbf{u}} \mathbf{1}^\top \mathbf{u} \tag{8}$$

$$\text{s.t.} \quad \boldsymbol{\alpha}_j \mathbf{x} \geq \boldsymbol{\alpha}_k \mathbf{x} \quad \forall \boldsymbol{\alpha}_k \in \Gamma \tag{9}$$

$$\mathbf{x}^k = 0 \quad \forall \operatorname{col}^k \left(\bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \right) = \mathbf{0} \tag{10}$$

$$\mathbf{x} \geq \mathbf{0} \tag{11}$$

$$\mathbf{1}^\top \bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x} = 1 \tag{12}$$

$$(\hat{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} - z\mathbf{1}^\top)\mathbf{x} \geq 0 \tag{13}$$

$$\mathbf{b}_t - \bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x} \leq \mathbf{u} \tag{14}$$

$$\mathbf{b}_t - \bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x} \geq -\mathbf{u} \tag{15}$$

where

$$\bar{\mathbf{O}}_{o_i} = \begin{bmatrix} Z(o_i \mid s'_1) & & \\ & \ddots & \\ & & Z(o_i \mid s'_N) \end{bmatrix} \tag{16}$$

$$\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} = \begin{bmatrix} T(s'_1 \mid s_1, a^j) & \ldots & T(s'_1 \mid s_N, a^j) \\ \vdots & \vdots & \vdots \\ T(s'_N \mid s_1, a^j) & \ldots & T(s'_N \mid s_N, a^j) \end{bmatrix} \tag{17}$$

$$\begin{aligned} \hat{\mathbf{O}}_{o_i} &= \begin{bmatrix} Z(o_i \mid s'_1) & \cdots & Z(o_i \mid s'_N) \end{bmatrix} \\ &\equiv \operatorname{diag}\left(\bar{\mathbf{O}}_{o_i}\right) \end{aligned} \tag{18}$$

and z is a hyperparameter whose value can be dynamically adjusted using the technique described in Section A.4. Here, $\mathbf{x}^k$ denotes the kth element of $\mathbf{x}$, and $\operatorname{col}^k(\cdot)$ denotes the kth column of $(\cdot)$.

The details of how we arrive at Equations (8) to (18) are elaborated in Section G. Here are the purposes of each of these equations, explained briefly: Equations (8), (14) and (15) formulate the objective function as minimizing the L1 error of the belief update between two timesteps. Equation (9) assures the selected action a^j is optimal. Equations (10) to (12) ensures the computed previous belief is valid. Equation (13) adjusts how probable the belief update between the two timesteps is, through the parameter z.

We compute the previous $\mathbf{b}^{ij}_{t-1}$ at time $t-1$ (before taking action a^j and receiving observation o_i), given the belief $\mathbf{b}_t$ at time t, as $\frac{\mathbf{x}}{\mathbf{1}^\top \mathbf{x}}$. The value of the variable $\mathbf{u}$ is of no direct interest.

A.4 Choosing the Value of z

The constraint in Equation (13) requires choosing the value of hyperparameter z, which may not be intuitive depending on the problem. Higher z values will enforce solutions $\mathbf{b}_{t-1}$ (i.e. $\frac{\mathbf{x}}{\mathbf{1}^\top \mathbf{x}}$) to have a higher probability of reaching $\mathbf{b}_t$. Nonetheless, selecting z too high may render the linear program of Equation (8) infeasible, since transition and observation dynamics of the POMDP may not allow it. To remedy this uncertainty, we can use a sampling method to pick a feasible z value. We first determine the maximum possible value of z for a given linear program. This can be achieved by solving a prior lightweight linear program that outputs $z_{\max}$, the maximum

possible value of z for a given observation o_i, and the action a^j corresponding to the alpha-vector $\boldsymbol{\alpha}_j \in \Gamma$, before attempting to solve Equation (8). Finding $z_{\max}$ can be achieved by solving the following linear program:

$$\max_{\mathbf{x},z} z \tag{19}$$

$$\text{s.t.} \quad \boldsymbol{\alpha}_j \mathbf{x} \geq \boldsymbol{\alpha}_k \mathbf{x} \quad \forall \boldsymbol{\alpha}_k \in \Gamma,\ k \neq j \tag{9}$$

$$\mathbf{x}^k = 0 \quad \forall \operatorname{col}^k \left(\bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \right) = \mathbf{0} \tag{10}$$

$$\hat{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x} = z \tag{20}$$

$$\mathbf{1}^\top \mathbf{x} = 1 \tag{21}$$

$$\mathbf{1} \geq x \geq \mathbf{0} \quad \forall x \in \mathbf{x} \tag{22}$$

where we refer to Equation (19) as the *inverse linear program* of Equation (8). Here, the first two constraints above are Equations (9) and (10) from Section A.3. Equation (20) enforces the probability $p(o_i \mid \mathbf{x}, a^j) = z$. Equations (21) and (22) assures $\mathbf{x}$ is a valid belief (i.e. a valid probability distribution).

Once the the value of $z_{\max}$ is found by solving Equation (19), z can be sampled arbitrarily from the interval $(0,\ z_{\max}]$ to solve Equation (8). As described earlier, smaller values of z cause BMCTS to explore unlikely regions of the belief-space, whereas values on the higher end result in a concentration on beliefs resulting from the most probable transitions and observations. In this paper, we propose a balance between the two extrema by sampling z from the following exponential distribution:

$$P(z) \propto \begin{cases} \exp\left(z k_{\text{z-exp}}\right), & \text{if } z \in (0, z_{\max}] \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

where $k_{\text{z-exp}}$ is a hyperparameter to tune the steepness of the exponential curve. Using an exponential curve encourages the selection of beliefs that have a higher likelihood, and therefore, the overall tree contains paths that with non-diminishing probabilities.

A.5 Extracting Vertices from the Optimal Solution Polytope

The linear program (LP) in Equation (8) can be solved by using off-the-shelf solvers such as Gurobi (Gurobi Optimization, LLC 2022) or JuMP (Dunning et al. 2017). However, such solvers only return a single vertex (the first one found) that minimizes the given objective function. In our tree construction, we require knowing the entire optimal polytope, not just a single vertex that belongs to it, to allow us to better sample from the belief-space. To do so, we use *pivoting* (Kochenderfer and Wheeler 2019) to jump between vertices of the optimal polytope until they are all discovered. Pivoting can only be done when the linear program is written in its *equality form*:

$$\begin{aligned} \min_{\mathbf{v}} \ & \mathbf{c}^\top \mathbf{v} \\ \text{s.t.} \ & \mathbf{A}\mathbf{v} = \mathbf{d} \\ & \mathbf{v} \geq \mathbf{0}. \end{aligned}$$

Hence, we transform the standard LP formulation in Equation (8) into an equality form.

As before, let N be the number of states of the POMDP formulation. Explicitly writing $\mathbf{A}, \mathbf{c}, \mathbf{d}$ requires the introduction of new slack variables. We first define our vector of all variables $\mathbf{v}$ as follows:

$$\mathbf{v} = \begin{bmatrix} \mathbf{x} & \mathbf{u} & \mathbf{g} & \mathbf{m} & \mathbf{n} & r \end{bmatrix}^\top. \tag{24}$$

Here, $\mathbf{x} \in \mathbb{R}^N$ contains the values that we are actually interested in, and $\mathbf{u}, \mathbf{m}, \mathbf{n} \in \mathbb{R}^N$, $\mathbf{g} \in \mathbb{R}^{|\Gamma|-1}$, and $r \in \mathbb{R}$ are slack variables that will allow us to write our previous LP in equality form. With $\mathbf{v}$ defined as above, we can write

the objective vector $\mathbf{c}$ consisting of zeros and ones as follows:

$$\mathbf{c} = \begin{bmatrix} \mathbf{0} \in \mathbb{R}^N \\ \mathbf{1} \in \mathbb{R}^N \\ \mathbf{0} \in \mathbb{R}^{|\Gamma|-1} \\ \mathbf{0} \in \mathbb{R}^N \\ \mathbf{0} \in \mathbb{R}^N \\ 0 \in \mathbb{R} \end{bmatrix}. \tag{25}$$

Note that the objective $\min_{\mathbf{v}} \mathbf{c}^\mathsf{T}\mathbf{v}$ is identical to the one in Equation (8). To formulate the constraints as $\mathbf{A}\mathbf{v} = \mathbf{d}$, we first define $\mathbf{A}$ as follows:

$$\mathbf{A} = \left[\begin{array}{c|ccccc} (\boldsymbol{\alpha}_j - \boldsymbol{\alpha}_1)^\mathsf{T} & & & & & \\ \vdots & 0 & -\mathbf{I} & & 0 & \\ (\boldsymbol{\alpha}_j - \boldsymbol{\alpha}_{k\neq j})^\mathsf{T} & & & & & \\ (\boldsymbol{\alpha}_j - \boldsymbol{\alpha}_{|\Gamma|-1})^\mathsf{T} & & & & & \\ \hline [0 \dots 1_{Null^1} \dots 0] & & & & & \\ [0 \dots 1_{Null^{k\in Null}} \dots 0] & & & 0 & & \\ \vdots & & & & & \\ [0 \dots 1_{Null^{|Null|}} \dots 0] & & & & & \\ \hline \mathbf{1}^\mathsf{T}\bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} & & & 0 & & \\ \hline \hat{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} - z\mathbf{1}^\mathsf{T} & & & 0 & & -1 \\ \hline \bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} & \mathbf{I} & 0 & -\mathbf{I} & 0 & 0 \\ \hline \bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} & -\mathbf{I} & & 0 & \mathbf{I} & \end{array}\right] \tag{26}$$

where *Null* denotes the set of indices k that satisfy Equation (10). We refer to matrix $\mathbf{A}$ as having six "*rows*" corresponding to individual constraints, and six "*columns*" corresponding to individual variables of $\mathbf{v}$. The subparts in matrix $\mathbf{A}$ denoted with 0 describe being filled with zeros entirely, and $\mathbf{I}$ is an identity matrix of appropriate size. Rows 1 and 2 contain multiple expressions stacked vertically. Table 3 tabulates how constraints of Equation (8) have been presented as a "row" in matrix $\mathbf{A}$.

Table 3. Constraints of Equation (8) and their corresponding "rows" of $\mathbf{A}$.

Constraint	***"Row" of* A**
Equation (9)	1
Equation (10)	2
Equation (12)	3
Equation (13)	4
Equation (14)	5
Equation (15)	6
Equation (11)	$\mathbf{v} \geq \mathbf{0}$

Similarly, we can define $\mathbf{d}$ as:

$$\mathbf{d} = \begin{bmatrix} \mathbf{0} \in \mathbb{R}^{|\Gamma|-1} \\ \mathbf{0} \in \mathbb{R}^{|Null|} \\ 1 \in \mathbb{R} \\ 0 \in \mathbb{R} \\ \mathbf{b}_t \in \mathbb{R}^N \\ \mathbf{b}_t \in \mathbb{R}^N \end{bmatrix}. \tag{27}$$

To extract vertices from the optimal polytope, we first run an off-the-shelf solver to find an initial feasible solution, using the above definitions of $\mathbf{A}, \mathbf{c}, \mathbf{d}$. Since the problem is convex, a solution is returned in polynomial time. Then, we recursively and exhaustively *pivot* all vertex indices that can be swapped from this initial solution. Ultimately, we are able to recover all vertices of the optimal polytope. Note that any convex combination of these vertices is still an optimal solution, and this will be the basis of our belief sampling technique described in the next subsection.

A.6 Sampling Beliefs

In the previous subsections, we first outlined how to formulate a linear program (LP) for finding previous beliefs, given the next belief and an action-observation pair (a, o). Here, action a was chosen from the UCB1 strategy after an observation o was sampled. Then, we have shown that this same LP can be rewritten as an equality form. Using this equality form, when an LP has multiple solutions (i.e., solutions lie on a polytope), all of the vertices can be obtained exhaustively using pivoting. Since both the objective function and the constraints are represented as linear relations, any convex combination of the extracted vertices is still an optimal solution to the LP. We can use this knowledge to sample different belief nodes in the tree and thereby create multiple branches bottom-up. Assume the optimal vertices are $\{\mathbf{v}_1, \mathbf{v}_2, \ldots \mid \mathbf{A}\mathbf{v}_k = \mathbf{d},\ \forall \mathbf{v}_k\}$ to the LP that approximates the previous belief $\mathbf{b}_{t-1}^{ij}$. We are only interested in the $\mathbf{x}$ components of these vertices (values of the slack variables are of no use, see Equation (24)). We create a belief sample by computing the convex combination:

$$\mathbf{x}_{sample} = w_1 \mathbf{v}_1^{\mathbf{x}} + w_2 \mathbf{v}_2^{\mathbf{x}} + \ldots \tag{28}$$

where $\mathbf{v}^{\mathbf{x}}$ represents the normalized $\mathbf{x}$ portion of solution $\mathbf{v}$, and the weights w_k are sampled from the following Dirichlet distribution (Barber 2012):

$$(w_1, w_2, \ldots) \sim \mathrm{Dir}\left(\frac{P(o_i \mid a^j, \mathbf{v}_1^{\mathbf{x}})}{\sigma}, \frac{P(o_i \mid a^j, \mathbf{v}_2^{\mathbf{x}})}{\sigma}, \ldots\right) \tag{29}$$

where

$$\sigma \triangleq \min_k P(o_i \mid a^j, \mathbf{v}_k^{\mathbf{x}})$$

$$P(o_i \mid a^j, \mathbf{x}) = \sum_s \sum_{s'} Z(o_i \mid s') T(s' \mid a^j, s) \frac{\mathbf{x}^s}{\mathbf{1}^\top \mathbf{x}}.$$

Sampling from the distribution above satisfies $\sum_k w_k = 1$, and thereby guarantees Equation (28) is a convex combination. Since each term satisfies $\frac{P(o_i|a^j,\mathbf{v}_k^{\mathbf{x}})}{\sigma} \geq 1$, Equation (29) behaves as a sampler that weighs each vertex proportionally to their reachability probability $P(o_i \mid a^j, \mathbf{v}_k^{\mathbf{x}})$.

Referring back to Figure 3(b), each red hexagon node is a belief sample that optimizes the linear program constructed for the preceding action-observation history. These beliefs are sampled from the attached previous observation node, using Equation (29).

A.7 Terminal Bayesian Probability of a Belief with History

In this subsection, we discuss how to compute the *terminal Bayesian probability* of a belief, given its history. Backward Monte Carlo Tree Search (BMCTS) returns a set of beliefs and their histories that contain individual sequences of actions and observations. We would like to identify, given their history, the probability of each belief reaching a specific final state or belief. We call this value the belief's terminal probability, and we compute it through a sequence of Bayesian operations. Given belief $\mathbf{b}_\tau$, and its history h, the terminal Bayesian probability of this belief can be computed using Algorithm 2.

Algorithm 2 Computing the terminal Bayesian probability of a belief, given its history.

function BayesianProb($\mathbf{b}_\tau$, h)
  $p \leftarrow 1$
  $\mathbf{b} \leftarrow \mathbf{b}_\tau$
  for $(a, o) \in h$ **do** ▷ Loops through action-observation pairs, forward in time.
    $p \leftarrow p \times$ BranchWeight($\mathbf{b}, a, o$)
    $\mathbf{b} \leftarrow$ UpdateBelief($\mathbf{b}, a, o$)
  end for
  return p
end function

function BranchWeight($\mathbf{b}, a, o$)
  $p \leftarrow \hat{\mathbf{O}}_o^\top \bar{\mathbf{T}}_a \mathbf{b}$ ▷ Approximately equal to $P(\mathbf{b}', o \mid \mathbf{b}, a)$. See Section A.7.
  return p
end function

function UpdateBelief($\mathbf{b}, a, o$)
  $\mathbf{b} \leftarrow \bar{\mathbf{O}}_o \bar{\mathbf{T}}_a \mathbf{b}$ ▷ Equivalent to $\mathbf{b}^{s'} \propto Z(o \mid s') \sum_s T(s' \mid s, a)\, \mathbf{b}^s \;\; \forall s' \in \mathcal{S}$.
  $\mathbf{b} \leftarrow \frac{\mathbf{b}}{\mathbf{1}^\top \mathbf{b}}$ ▷ Normalization.
  return $\mathbf{b}$
end function

The main function of Algorithm 2 is the BayesianProb function. We initialize the terminal probability value as $p = 1$. The BayesianProb function loops through all the action-observation pairs recorded in history h. For each (a, o) pair, a BranchWeight value is computed. This value approximates the probability $p = P(\mathbf{b}', \; o \mid \mathbf{b}, \; a)$ through the following relation:

$$
\begin{aligned}
P(\mathbf{b}', o \mid \mathbf{b}, a) &= P(\mathbf{b}' \mid \mathbf{b}, a, o)\; P(o \mid \mathbf{b}, a) \\
&= P(\mathbf{b}' \mid \mathbf{b}, a, o) \sum_{s'} P(o \mid \mathbf{b}, a, s')\; P(s' \mid \mathbf{b}, a) \\
&= P(\mathbf{b}' \mid \mathbf{b}, a, o) \sum_{s'} P(o \mid \mathbf{b}, a, s') \sum_{s} P(s' \mid \mathbf{b}, a, s)\; P(s \mid \mathbf{b}, a) \\
&= \cancelto{\approx 1}{P(\mathbf{b}' \mid \mathbf{b}, a, o)} \sum_{s'} Z(o \mid s') \sum_{s} T(s' \mid a, s)\, \mathbf{b}^s \\
&\approx \sum_{s'} Z(o \mid s') \sum_{s} T(s' \mid a, s)\, \mathbf{b}^s
\end{aligned} \tag{30}
$$

where $\mathbf{b}'$ denotes the updated belief of $\mathbf{b}$ if action a was taken, and observation o was received. We note that the final expression of Equation (30) becomes independent from $\mathbf{b}'$. Here, we make one crucial assumption that $P(\mathbf{b}' \mid \mathbf{b}, a, o) \approx 1$ (we explain why this is a logical assumption later in this section). The final expression can also be vectorized as $\hat{\mathbf{O}}_o^\top \bar{\mathbf{T}}_a \mathbf{b}$ where the definitions of $\bar{\mathbf{T}}_{a_j}$ and $\hat{\mathbf{O}}_{o_i}$ were given in Equations (17) and (18), respectively. After computing the value of $P(\mathbf{b}', o \mid \mathbf{b}, a)$ using BranchWeight function, this value is multiplied by p to update the value of p (Line 5 in BayesianProb). Then, belief $\mathbf{b}$ is deterministically updated (forward in time) using the UpdateBelief function, which uses Bayes' rule with action a and observation o (Line 6 in BayesianProb). This operation is identical to Equation (33). These two operations, BranchWeight and UpdateBelief, are called iteratively for all (a, o) pairs in h.

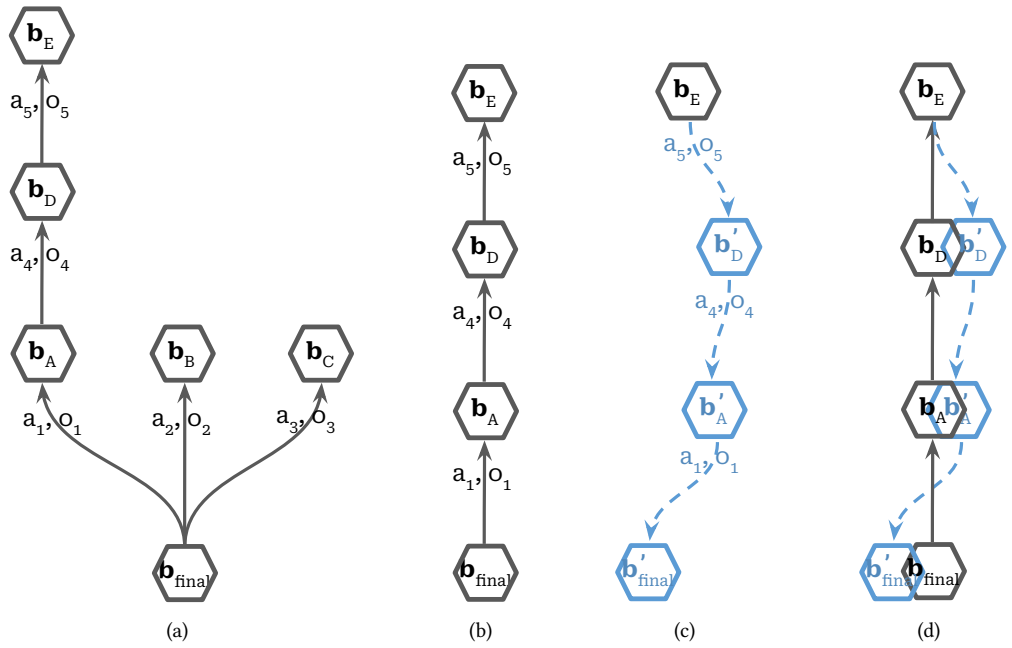


Fig. 10. The tree branches backwards in time, starting from the bottom-most leaf node of $\mathbf{b}_{\text{final}}$ shown in (a). For now, let us focus on the sample trajectory given in (b) of reaching a possible root node $\mathbf{b}_{\text{E}}$. The probability of reaching $\mathbf{b}_{\text{final}}$ from $\mathbf{b}_{\text{E}}$ when the given history of actions-observations can be computed through Bayesian relations forward in time. However, these forward Bayesian updates, shown in (c), will slightly deviate from the approximated backward trajectory. Nonetheless, the deviation between the approximated backward and re-computed forward trajectories, depicted in (d), will be negligibly small, as explained in the rest of this section, and demonstrated in Section 7.

To understand why each branch (i.e. edge in the tree) is weighted by $P(\mathbf{b}', o \mid \mathbf{b}, a)$, we need to first understand the *cause and effect relation* between beliefs, actions, and observations: When we update beliefs forward in time, the cause is the previous belief and the action that was taken, and as an effect, an observation is received. Using this observation, we update our belief. Therefore, the *cause* is the previous belief $\mathbf{b}$ and action a, where their effect is receiving observation o, and thereby updating to belief $\mathbf{b}'$; whose probability is represented by $P(\mathbf{b}', o \mid \mathbf{b}, a)$. However, this probability value is valid for a single timestep update. Since the branches in the backward tree include multiple layers, we would like to expand this probability computation across a sequence of belief updates. We graphically show this in Figure 10.

From a leaf belief node, multiple previous belief nodes can be computed or sampled. The early stages of backward tree construction are demonstrated in Figure 10(a). The bottom-most leaf node is $\mathbf{b}_{\text{final}}$, and three belief nodes are sampled in the second to last layer, with respect to the given (action, observation) pairs in the figure. We note that the representation in Figure 10 uses the same depiction for a backward tree construction as Figure 3(b), but the action and observation nodes are omitted for brevity. Each of these $\mathbf{b}_{\text{A}}$, $\mathbf{b}_{\text{B}}$, $\mathbf{b}_{\text{C}}$ nodes can be further constructed backwards in time. In Figure 10(b), we constrain our focus on a specific 4-layer trajectory between $\mathbf{b}_{\text{final}}$ and $\mathbf{b}_{\text{E}}$. In this example path, our backward tree tells us that there is a non-zero probability that, when started from $\mathbf{b}_{\text{E}}$, if action a_5 is taken and observation o_5 is observed, the belief would be updated to $\mathbf{b}_{\text{D}}$ (forward in time). Similarly, the pair (a_4, o_4) from $\mathbf{b}_{\text{D}}$ would lead to $\mathbf{b}_{\text{A}}$. And finally, the pair (a_1, o_1) from $\mathbf{b}_{\text{A}}$ would lead to $\mathbf{b}_{\text{final}}$. In other words, by following history $h = (a_5, o_5, a_4, o_4, a_1, o_1)$ when the initial belief is $\mathbf{b}_{\text{E}}$, the final belief would be $\mathbf{b}_{\text{final}}$, and the probability of this update happening is non-zero. We use BayesianProb to compute this probability $P(\mathbf{b}_{\text{final}}, o_5, o_4, o_1 \mid \mathbf{b}_{\text{E}}, a_5, a_4, a_1)$.

As depicted in Figure 10(c), we simulate the root node $\mathbf{b}_{\text{E}}$ forward in time. First, we calculate the probability $P(\mathbf{b}_{\text{D}}, o_5 \mid \mathbf{b}_{\text{E}}, a_5)$ using the BranchWeight function. Then, we deterministically update $\mathbf{b}_{\text{E}}$ forward in time, using (a_5, o_5), where the resulting new belief is $\mathbf{b}'_{\text{D}}$. Here, beliefs $\mathbf{b}_{\text{D}}$ and $\mathbf{b}'_{\text{D}}$ may not be equivalent, depending on the objective value J after the linear program for that layer is solved. That being said, J will have a negligible value, and be very close to zero.[8] For this exact reason, we have assumed $P(\mathbf{b}' \mid \mathbf{b}, a, o) \approx 1$, in this case $P(\mathbf{b}'_{\text{D}} \mid \mathbf{b}_{\text{E}}, a_5, o_5) \approx 1$, in Equation (30).[9] This assumption is empirically demonstrated to be accurate in Section 7. The belief update from $\mathbf{b}_{\text{E}}$ to $\mathbf{b}'_{\text{D}}$ is handled by UpdateBelief.

As demonstrated in Figure 10(c) and described in Algorithm 2, Lines 4–7, we iteratively call the BranchWeight function over the new beliefs found by the UpdateBelief function. These Bayesian relations allow us to simulate a root node forward in time, given its history, up until the leaf node $\mathbf{b}_{\text{final}}$:

$$\begin{aligned} P(\mathbf{b}_{\text{final}}, o_5, o_4, o_1 \mid \mathbf{b}_{\text{E}}, a_5, a_4, a_1) & \\ &= P(\mathbf{b}_{\text{D}}, o_5 \mid \mathbf{b}_{\text{E}}, a_5)\ P(\mathbf{b}_{\text{A}}, o_4 \mid \mathbf{b}_{\text{D}}, a_4)\ P(\mathbf{b}_{\text{final}}, o_1 \mid \mathbf{b}'_{\text{A}}, a_1) \\ &\approx P(\mathbf{b}'_{\text{D}}, o_5 \mid \mathbf{b}_{\text{E}}, a_5)\ P(\mathbf{b}'_{\text{A}}, o_4 \mid \mathbf{b}'_{\text{D}}, a_4)\ P(\mathbf{b}'_{\text{final}}, o_1 \mid \mathbf{b}'_{\text{A}}, a_1). \end{aligned} \tag{31}$$

The first relation above holds due to the Markovian property of the problem, i.e., a belief update depends only on the most recent action-observation pair, as demonstrated in the proof below.

Lemma A.1. *The terminal Bayesian probability for a multi-layer history path is the product of the reachability probabilities of all edges in the path.*

Proof. Let's consider computing the terminal Bayesian probability of a 3-layer history $h = (a_1, o_1, a_2, o_2)$ for the tree path $\mathbf{b} \xrightarrow{a_1,o_1} \mathbf{b}' \xrightarrow{a_2,o_2} \mathbf{b}''$ (root to leaf). Then, the probability of reaching belief $\mathbf{b}''$ from $\mathbf{b}$ while following

[8] This is because previous beliefs are found *locally*, i.e., considering a single backward timestep. In a 3-layer branch in the tree: $\mathbf{b}'' \to \mathbf{b}' \to \mathbf{b}$, the generated or sampled $\mathbf{b}'$ from $\mathbf{b}$ might have a large entropy (a spread belief), and therefore, no action-observation pair from $\mathbf{b}''$ might make it attainable to exactly update to $\mathbf{b}'$. This can be easily remedied by a more domain-specific belief sampling scheme (instead of the one suggested in Section A.6).

[9] In the case where $J = 0$ occurs, then we would indeed observe $\mathbf{b}'_{\text{D}} \equiv \mathbf{b}_{\text{D}}$, which would then imply $P(\mathbf{b}'_{\text{D}} \mid \mathbf{b}_{\text{E}}, a_5, o_5) = 1$.

history h can be computed as follows:

$$\begin{aligned} P(\mathbf{b}'', o_1, o_2 \mid \mathbf{b}, a_1, a_2) &= P(o_1 \mid \mathbf{b}, a_1, a_2)\ P(\mathbf{b}'', o_2 \mid \mathbf{b}, a_1, a_2, o_1) \\ &= P(o_1 \mid \mathbf{b}, a_1, a_2) \sum_{\tilde{\mathbf{b}'} \in \mathcal{B}} P(\mathbf{b}'', o_2 \mid \tilde{\mathbf{b}'}, \mathbf{b}, a_1, a_2, o_1)\ P(\tilde{\mathbf{b}'} \mid \mathbf{b}, a_1, a_2, o_1) \\ &= P(o_1 \mid \mathbf{b}, a_1) \sum_{\tilde{\mathbf{b}'} \in \mathcal{B}} P(\mathbf{b}'', o_2 \mid \tilde{\mathbf{b}'}, a_2)\ P(\tilde{\mathbf{b}'} \mid \mathbf{b}, a_1, o_1) \\ &= P(o_1 \mid \mathbf{b}, a_1)\ P(\mathbf{b}'', o_2 \mid \mathbf{b}', a_2)\ P(\mathbf{b}' \mid \mathbf{b}, a_1, o_1) \\ &= P(\mathbf{b}', o_1 \mid \mathbf{b}, a_1)\ P(\mathbf{b}'', o_2 \mid \mathbf{b}', a_2) \end{aligned} \tag{32}$$

where the summation over all possible next beliefs ($\tilde{\mathbf{b}'} \in \mathcal{B}$) simplified to the case of just $\mathbf{b}'$. This is because only $\mathbf{b}'$ can be achieved when belief $\mathbf{b}$ is updated with respect to (a_1, o_1), i.e., belief updates are deterministic: $P(\tilde{\mathbf{b}'} \mid \mathbf{b}, a_1, o_1) = 0 \quad$ for all $\tilde{\mathbf{b}'} \in \mathcal{B},\ \tilde{\mathbf{b}'} \neq \mathbf{b}'$. The expression in Equation (32) can be generalized to a tree path of arbitrary layers.

As depicted in Figure 10(d), the simulated forward trajectory slightly deviated from the approximated backward trajectory.[10] However, due to the assumption $P(\mathbf{b}' \mid \mathbf{b}, a, o) \approx 1$ for every layer during belief updates, the deviation is negligibly small. As a result, the terminal Bayesian probability, found by BayesianProb, of $\mathbf{b}_{\mathrm{E}}$ for following its given history, is highly accurate.

Algorithm 3 Starting backward search from a specific final belief.

Params: $t_{\max}$ (tree search max depth), m_{sims} (number of simulations)

```
function Search(b_final)
    T = InitializeTree()
    for t = 1 to t_max do
        for m = 1 to m_sims do
            b_t, h ← SampleLeaf(T, t − 1)
            Simulate(T, b_t, h)
        end for
    end for
    return T.B
end function
```

A.8 Putting It All Together: Backward Monte Carlo Tree Search

In this section, we present Backward Monte Carlo Tree Search (BMCTS) with its entirety. The construction of the backward tree starts with Algorithm 3. In addition to the POMDP formulation of the problem, the only input argument that needs to be passed is the final belief node (the single bottom-most leaf node) $\mathbf{b}_{final}$ that the tree will start searching upwards from. The Search function uses two hyperparameters: the maximum number of timesteps we desire the tree to search backward (depth) $t_{\max}$ and the number of simulations desired for each timestep layer in the tree m_{sims}. The Search function starts by initializing an empty tree $\mathcal{T}$. Here, $\mathcal{T}$ is a structure that has three fields: $N(h)$ that counts the number of times an action-observation history h has been visited in the tree, $Q(h)$ that records h, the estimated mean reachability probability for all histories that have the suffix h,

[10] This difference Δp arises from several factors, including the belief sampling technique used, non-unique solutions to the LP objective function, and numerical precision errors in the LP solver.

and B which is a set of (belief, history) pairs. At each timestep layer t, a leaf node is sampled to branch out the remaining timesteps $t_{\max} - t$. The function SampleLeaf samples a (belief, history) pair from $\mathcal{T}.B$.[11] For $t = 1$, the only pair that SampleLeaf returns is $(\mathbf{b}_{final}, \emptyset)$. For each of the $(\mathbf{b}_t, h)$ pairs sampled, the tree is simulated through the Simulate function, defined in Algorithm 4, that modifies $\mathcal{T}$ in-place. We note that Algorithm 3 is an anytime algorithm and is parallelizable.

In Algorithm 4, we simulate a branch for each starting leaf node, backward in time. The required hyperparameters are: $t_{\max}$, same as Algorithm 3, k_{ucb}, the UCB1 exploration constant from Section A.2, and $k_{\max}$, the z-value's exponential distribution constant from Section A.4. The Simulate function is very similar to the original forward-search Monte Carlo planning algorithm (Silver and Veness 2010). At every call, it first appends the (belief, history) pair to the tree. Then, the history is checked whether it satisfies the maximum depth requirement, and if not, the tree continues branching backward in time. If a certain history has not been visited before, the branch rolls out through the Rollout function in Algorithm 5. Otherwise, an observation is sampled from the probability distribution $P(o)$ as described in Section A.1. Afterward, as discussed in Section A.2, an action is sampled with respect to the UCB1 strategy. The function continues with the intent to construct a linear program (LP), as discussed in Section A.3, to detect possible previous beliefs. To construct the LP, a z-value is required.

Algorithm 4 Simulating a single belief node backwards in time.

Params: $t_{\max}$ (tree search max depth), k_{ucb} (exploration constant), $k_{\text{z-exp}}$ (z-value sampling constant)

function Simulate($\mathcal{T}$, $\mathbf{b}_t$, h)
  $\mathcal{T}.B \leftarrow \mathcal{T}.B \cup \{(\mathbf{b}_t, h)\}$ ▷ Append belief and its history to tree.
  if Depth(h) = $t_{\max}$ **then**
    return BayesianProb($\mathbf{b}_t$, h) ▷ Termination due to depth.
  end if
  if $(\cdot, h) \notin \mathcal{T}.B$ **then**
    return Rollout($\mathbf{b}_t$, h) ▷ Rollout due to lack of history in existing tree.
  end if
  $o \leftarrow$ SampleObs($\mathbf{b}_t$)
  $a \leftarrow$ UCB1(k_{ucb}, h, o)
  $z \leftarrow$ SampleInverseLP($\mathbf{b}_t$, a, o, $k_{\text{z-exp}}$) ▷ Sample exponentially from $(0, z_{\max}]$.
  $\mathbf{v} \leftarrow$ ConstructAndSolveLP($\mathbf{b}_t$, z, a, o) ▷ Optimal vertices of the LP.
  if $\mathbf{v} = \emptyset$ **then**
    $q \leftarrow 0$ ▷ Discourage histories that are unable to reach depth of $t_{\max}$.
  else
    $\mathbf{b}_{t-1} \leftarrow$ SampleBelief($\mathbf{v}$, a, o)
    $q \leftarrow$ Simulate($\mathcal{T}$, $\mathbf{b}_{t-1}$, aoh) ▷ Recursively simulate backwards.
  end if
  $\mathcal{T}.N(oh) \leftarrow \mathcal{T}.N(oh) + 1$
  $\mathcal{T}.N(aoh) \leftarrow \mathcal{T}.N(aoh) + 1$
  $\mathcal{T}.Q(aoh) \leftarrow \mathcal{T}.Q(aoh) + \frac{q - \mathcal{T}.Q(aoh)}{\mathcal{T}.N(aoh)}$
  return q
end function

As explained in Section A.4, the maximum possible value of z, denoted $z_{\max}$, can be solved for using the inverse of the original linear program. We note that the $z_{\max}$ value is unique for each $(\mathbf{b}_t, a, o)$ triplet. Once $z_{\max}$ is known,

[11] We use dot notation to denote the fields N, Q and B of a tree $\mathcal{T}$. For example, $\mathcal{T}.B$ denotes the B field of the tree $\mathcal{T}$.

a z-value can be sampled from the exponential distribution, parametrized by $k_{\text{z-exp}}$, from an interval between 0 and $z_{\max}$. Both of these operations, computing $z_{\max}$ and sampling the z-value, given by Equations (19) and (23) respectively, are handled by SampleInverseLP. Once this z-value is obtained, the LP can now be formulated.

Since we are interested not only in *one* optimal solution but rather the *vertices* spanning the optimal solution polygon, we need to transform the original linear program into its equality form. The process of the equality transformation, and how to extract the optimal vertices from it, is elaborated in Section A.5. In Algorithm 4, the function ConstructAndSolveLP handles these operations: constructs the linear program with respect to action a and observation o, whose belief reachability is parametrized by z, and then transforms it into its equality form. Then using the equality form, the optimal vertices can be extracted by pivoting, as discussed in Section A.5. The function ConstructAndSolveLP returns the set of optimal vertices $\mathbf{v}$. If there are no vertices, (i.e., $\mathbf{v}$ is an empty set), then we set the local Q-value, denoted as q, to be zero for history h. Otherwise, we are able to sample a belief from the optimal polygon of the LP, using vertices $\mathbf{v}$. The function SampleBelief uses the approach described in Section A.6 to return a feasible previous belief $\mathbf{b}_{t-1}$. The tree can further branch backward by recursively calling the Simulate function, now with the previous belief $\mathbf{b}_{t-1}$ with its history aoh. Once the tree reaches the desired depth, the terminal Bayesian probability of the root belief, by following its history thus far, is returned. The terminal Bayesian probability value is computed by BayesianProb using the relations described in Section A.7. Finally, the N and Q fields of $\mathcal{T}$ are updated accordingly.

Algorithm 5 Rolling-out a single belief node backwards in time.

Params: $t_{\max}$ (tree search max depth), $k_{\text{z-exp}}$ (z-value sampling constant)

1: **function** Rollout($\mathcal{T}$, $\mathbf{b}_t$, h)
2: $\quad\mathcal{T}.B \leftarrow \mathcal{T}.B \cup \{(\mathbf{b}_t, h)\}$ ▷ Append belief and its history to tree.
3: $\quad$**if** Depth(h) = $t_{\max}$ **then**
4: $\quad\quad$**return** BayesianProb($\mathbf{b}_t$, h) ▷ Termination due to depth.
5: $\quad$**end if**
6: $\quad o \leftarrow$ SampleObs($\mathbf{b}_t$)
7: $\quad a \leftarrow \pi_{\text{rollout}}(h)$
8: $\quad z \leftarrow$ SampleInverseLP($\mathbf{b}_t$, a, o, $k_{\text{z-exp}}$) ▷ Sample exponentially from $(0, z_{\max}]$.
9: $\quad\mathbf{v} \leftarrow$ ConstructAndSolveLP($\mathbf{b}_t$, z, a, o) ▷ Optimal vertices of the LP.
10: $\quad$**if** $\mathbf{v} = \emptyset$ **then**
11: $\quad\quad$**return** 0 ▷ Discourage histories that are unable to reach depth of $t_{\max}$.
12: $\quad$**else**
13: $\quad\quad\mathbf{b}_{t-1} \leftarrow$ SampleBelief($\mathbf{v}$, a, o)
14: $\quad\quad$**return** Rollout($\mathcal{T}$, $\mathbf{b}_{t-1}$, aoh) ▷ Recursively rollout backwards.
15: $\quad$**end if**
16: **end function**

Algorithm 5 explains how Rollout works, which is very similar to the way Simulate operates. Rollout starts by appending the (belief, history) pair to the tree, and then the depth requirement is checked. If the desired depth is reached, then the Rollout terminates by returning the terminal Bayesian probability value. Otherwise, similar to the Simulate function, an observation is sampled. However, unlike the Simulate function, the Rollout function uses a *rollout policy* π_{rollout} to select the action. Although the policy can include domain knowledge, we choose to use a random policy in this paper. Then, a z-value is sampled after solving the inverse linear program, and the set of optimal vertices $\mathbf{v}$ are computed. If this set is empty, then the Rollout function returns zero, otherwise, a previous belief is sampled, and the Rollout function is called recursively.

B Relaxing the Assumption of Fully Observable Final States

In this paper, we initially assumed that final states are fully observable, and the backward tree was constructed based on this premise. However, this assumption can be relaxed. In complex environments where certain events may not be directly perceivable by the agent, the assumption of fully observable final states might not hold true.

BMCTS, however, is adaptable and can operate from a belief rather than a collapsed belief. While the problem is often framed using a collapsed final belief for simplicity, the algorithm is capable of handling any set of final beliefs.

For scenarios where the final state is not fully observable, a small set of beliefs representing likely situations (e.g., two vehicles colliding) can be identified based on the transition and observation dynamics. By sampling leaf (most-bottom, $t = 0$) nodes from this set and running BMCTS for each sampled belief, potentially in parallel, it becomes possible to approximate the reachability probabilities for these cases. This approach effectively extends BMCTS to accommodate partially observable final states as well as the simultaneous consideration of multiple distinct final states and beliefs.

C Suggestions on Choosing τ, Number of Layers

The parameter τ in BMCTS determines the number of steps sampled backward, serving as a key factor in balancing computational efficiency and the precision of probability estimates.

A larger τ allows for longer trajectories, which can improve the accuracy of probability estimates by incorporating a more comprehensive history of actions and observations. However, this comes at the cost of higher computational demands and a potential increase in noise. Conversely, a smaller τ simplifies computation and focuses on recent events, but it may overlook the effects of earlier actions, reducing the precision of probability estimates.

Choosing the optimal τ depends on the system's specific dynamics and the required level of safety. Sensitivity analyses are recommended to evaluate how varying τ values influence both probability estimates and safety performance. Additionally, leveraging domain-specific knowledge can help ensure that the chosen τ aligns with the temporal characteristics of the system's behavior.

Building on AlphaZero (Silver, Hubert, et al. 2017), a potential direction for future research could involve designing a *backward value network* for BMCTS. Just as value networks in AlphaZero minimize the need for extensive lookahead, a backwards value network could reduce the dependency on large τ values by offering more efficient estimates of reachability probabilities. This strategy could achieve a better balance between computational efficiency and accuracy, especially in scenarios where deep backward sampling is computationally intensive. We look forward to investigating this intriguing possibility in future studies.

D Probabilistic Convergence Properties of BMCTS

The convergence properties of Backward Monte Carlo Tree Search (BMCTS) show many similarities to the PO-UCT approach first introduced in the POMCP (Silver and Veness 2010) algorithm. The key innovation of the PO–UCT algorithm is applying UCT search to a history-based belief-MDP. Their study leverages the fact that the original UCT (Kocsis and Szepesvári 2006) algorithm converges to the optimal value function for fully observable MDPs, and that a POMDP can always be (although computationally inefficient) represented as a belief-MDP. Therefore, the convergence bounds for the original UCT algorithm also held for PO-UCT. We follow a similar strategy and show that the converge bounds for POMCP also hold for BMCTS.

Theorem D.1. *For a suitable value of k_{ucb}, the Q-values constructed by BMCTS (Algorithm 3) converge to their optimal value, i.e. $Q(h) \xrightarrow{p} Q^*(h)$, when all h branch out from the same (leaf) node. The bias of the Q-value function $\mathbb{E}\left[Q(h) - Q^*(h)\right]$ becomes $O\left(\log N(h)/N(h)\right)$ as $N(h) \to \infty$.*

Proof. The distribution of the rollout policy π_{rollout} used in Algorithm 5 can be equated to its derived MDP rollout distribution using the same formulation in Lemma 2 from the POMCP paper (Silver and Veness 2010). This is because both Rollout (Algorithm 5) and the PO-UCT rollout function in the POMCP paper rely only on history to form a rollout distribution. Therefore, as is the case for PO-UCT, BMCTS simulations can be mapped into UCT simulations (Kocsis and Szepesvári 2006). In addition, similar to the value function $V(h)$ in PO-UCT, the reachability Q-value function $Q(h)$ in BMCTS behaves the same for POMDP histories and its derived MDP states, and therefore, map to the same Q-values. Following these two outcomes, we conclude that the analyses in UCT and PO-UCT can be transferred to BMCTS.

Algorithm 6 Policy retraining using unsafe beliefs identified by BMCTS.

Params: $\mathcal{B}_{\text{init}}$ (initial belief set), π (initial policy), N_{iter} (number of iterations)

function RetrainPolicy($\mathcal{B}_{\text{init}}, \pi$)
  $\mathcal{B}_{\text{unsafe}} \leftarrow$ BMCTS(π)
  $\mathcal{B}_{\text{new}} \leftarrow \mathcal{B}_{\text{init}} \cup \mathcal{B}_{\text{unsafe}}$
  $\Gamma' = \emptyset$ ▷ Initialize set of α-vectors for retrained policy.
  for $n = 1$ **to** N_{iter} **do**
    for each belief $b \in \mathcal{B}_{\text{new}}$ **do**
      for each action $a \in A$ **do**
        $Q_a(b) \leftarrow R(b,a) + \gamma \sum_o P(o \mid b, a)\ \max_{\alpha \in \Gamma'} \alpha^\top \tau(b,a,o)$
      end for
      $\alpha_b \leftarrow \arg\max_a Q_a(b)$
      $\Gamma' \leftarrow \Gamma' \cup \{\alpha_b\}$
    end for
  end for
  $\pi' \leftarrow$ ExtractPolicy($\mathcal{B}_{\text{new}}, \Gamma'$)
  return π'
end function

function ExtractPolicy($\mathcal{B}_{\text{new}}, \Gamma'$)
  Initialize $\pi' = \emptyset$
  for each belief $b \in \mathcal{B}_{\text{new}}$ **do**
    $a^*(b) \leftarrow \arg\max_{a \in A} \alpha_a^\top b \quad$ where $\alpha_a \in \Gamma'$
    $\pi'(b) \leftarrow a^*(b)$
  end for
  return π'
end function

E Reward Structure Details

The reward functions for the Gridworld and Autonomous Car domains are designed to balance goal achievement, efficiency, and safety.

Gridworld Domain

- Reward of +30 for reaching the goal cell at $(4, 2)$,
- Reward of +20 for reaching cell $(6, 7)$,

- Penalty of -25 for entering unsafe cell $(2, 5)$,
- Penalty of -15 for entering cell $(7, 4)$,
- No additional penalties for redundant movements, as the discount factor γ inherently discourages them.

Autonomous Car Domain

- Reward of $+100$ for successfully completing the episode,
- Penalty of -1000 for a collision,
- Penalty of -20 per every 0.1s delay to complete the episode,
- Penalty of -10 for unnecessary braking or maintaining zero velocity when it is safe to proceed.

This structure encourages the agent to continue progressing toward its goal while maintaining safety. The impact of these reward definitions can be seen in Tables 1 and 2, which show reduced collision probability and improved task efficiency after retraining.

F Policy Retraining

To improve performance, we retrain the agent's policy using the unsafe beliefs identified through BMCTS. Retraining is performed via *Point-Based Value Iteration* (PBVI) (Kochenderfer, Wheeler, and Wray 2022), where the belief set is augmented with unsafe beliefs discovered by BMCTS.

Algorithm 6 outlines the retraining procedure, showing how unsafe beliefs are integrated into the PBVI process and how the updated policy π' is obtained from the resulting set of α-vectors. Both the initial and retrained policies incorporate the reward structure (detailed in Section E) that penalizes delay and inaction, balancing safety with task completion.

Our results indicate that retraining the agent's policy over these discovered beliefs significantly improves safety performance by drastically decreasing reachability probabilities, as demonstrated in Tables 1 and 2.

G Linear Program for Computing Previous Beliefs

In this section, we describe in greater detail how the linear program was formulated through Equations (8) to (18).

We denote $\mathbf{b}_{t-1}^{ij}$ as the previous belief to $\mathbf{b}_t$ before taking action a^j and receiving observation o_i. Given that the agent has received observation o_i after taking an action at time $t-1$, we can construct the diagonal observations matrix:

$$\bar{\mathbf{O}}_{o_i} = \begin{bmatrix} Z(o_i \mid s'_1) & & \\ & \ddots & \\ & & Z(o_i \mid s'_N) \end{bmatrix} \tag{16}$$

where $s'_1, \ldots, s'_N$ represent the next states the agent could have transitioned to at time t. The off-diagonal elements in the matrix above are zeros. Given that an agent has taken the action a^j corresponding to the alpha-vector $\boldsymbol{\alpha}_j \in \Gamma$, we can construct the following square transition matrix:

$$\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} = \begin{bmatrix} T(s'_1 \mid s_1, a^j) & \ldots & T(s'_1 \mid s_N, a^j) \\ \vdots & \vdots & \vdots \\ T(s'_N \mid s_1, a^j) & \ldots & T(s'_N \mid s_N, a^j) \end{bmatrix}. \tag{17}$$

It can be seen that, if $\mathbf{b}_{t-1}^{ij}$ were known, the updated belief $\mathbf{b}_t$ could have been computed using:

$$\begin{aligned} \mathbf{b}_t &= \eta\, \bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{b}_{t-1}^{ij} \\ &\propto \bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{b}_{t-1}^{ij} \end{aligned} \tag{33}$$

where η is a normalization constant. Equation (33) is the matrix form of Equation (1).

Now, let us consider the problem of dealing with the converse direction: Assuming that we know the belief $\mathbf{b}_t$ at timestep t, and the observation that the agent has received was o_i after having taken action a^j associated with $\boldsymbol{\alpha}_j$, we would like to find the belief $\mathbf{b}_{t-1}$ of the previous timestep $t-1$. We define the following objective function to formalize this problem as:

$$\min_{\mathbf{x},y} \|y\mathbf{b}_t - \bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\mathbf{x}\|_1 \tag{34}$$

where we define vector $\mathbf{x}$ to be our estimate of $\mathbf{b}_{t-1}^{ij}$ and the scalar y to be the inverse of the normalization constant η as used in Equation (33). Thus, we can rewrite Equation (33) as:

$$y = \mathbf{1}^\mathsf{T}\bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\mathbf{x}. \tag{35}$$

Using an L1 norm forces the optimal solution of the problem to lie on a convex polygon, and guarantees at least one optimal solution as long as the problem is not over-constrained. The expression inside the norm in Equation (34) being zero would imply an exact belief update from $\mathbf{x}$ to $\mathbf{b}_t$ if observation o_i is received when the action of α_j is taken, which is what we desire.

Since the belief should be a valid probability distribution of states, and its elements should sum to 1, we have the following constraints:

$$\mathbf{1}^\mathsf{T}\mathbf{x} = 1 \tag{36}$$

$$\mathbf{1} \geq x \geq \mathbf{0} \quad \forall x \in \mathbf{x} \tag{37}$$

where $\mathbf{0}$ denotes the appropriately sized vector of zeros.

Furthermore, to prevent division by zero during belief update normalization, we force $y > 0$. This can be equivalently represented as:

$$y \geq \epsilon \tag{38}$$

where ϵ is a very small positive number.

Since we assume that the agent's policy Γ (consisting of α-vector plans) is known, the selected action for any potential belief $\mathbf{x}$ would be the one with the highest utility, and can be expressed using the the following set of constraints:

$$\boldsymbol{\alpha}_j\mathbf{x} \geq \boldsymbol{\alpha}_k\mathbf{x} \quad \forall \boldsymbol{\alpha}_k \in \Gamma. \tag{9}$$

Finally, the solution belief $\mathbf{x}$ should not lie in the nullspace of the square matrix $\bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}$, i.e.,

$$\mathbf{x} \notin \text{null}(\bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}).$$

If this constraint is not added, the optimizer can arbitrarily set the elements of $\mathbf{x}$ such that the objective function in Equation (34) can be driven to zero without a meaningful solution. We can equivalently express the nullspace constraint as the following set of equality constraints:

$$\mathbf{x}^k = 0 \quad \forall \text{ col}^k\left(\bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\right) = \mathbf{0} \tag{10}$$

where $\mathbf{x}^k$ denotes the kth element of $\mathbf{x}$, and $\text{col}^k(\cdot)$ denotes the kth column of $(\cdot)$.

Combining all of the derivations above, the entire problem can be formulated as the following linear program (LP):

$$\min_{\mathbf{x},y} ||y\mathbf{b}_t - \bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\mathbf{x}||_1 \tag{39}$$

$$\text{s.t.} \quad \text{Equations (9), (10), (35), (36), (37) and (38).}$$

However, when there are multiple solutions to the LP described in Equation (39), which is often the case, then the scaling of the objective value due to y causes missing out some of the optimal vertices.[12] To remedy the aforementioned issue, we propose making the following modifications:

$$\mathbf{1} \geq \mathbf{x} \geq \mathbf{0} \quad \longrightarrow \quad \mathbf{x} \geq \mathbf{0} \tag{11}$$

$$\mathbf{1}^\top \mathbf{x} = 1 \quad \longrightarrow \quad \text{removed}$$

$$y \geq \epsilon \quad \longrightarrow \quad y \triangleq 1. \tag{40}$$

Combining Equations (35) and (40), we have:

$$\mathbf{1}^\top \bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x} = 1 \tag{12}$$

that simply forces the updated belief to be a valid probability distribution. Accordingly, our new LP formulation becomes the following:

$$\min_{\mathbf{x}} ||\mathbf{b}_t - \bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x}||_1 \tag{41}$$

$$\text{s.t.} \quad \text{Equations (9), (10), (11) and (12)}$$

where our estimate of $\mathbf{b}_{t-1}^{ij}$ is now $\frac{\mathbf{x}}{\mathbf{1}^\top \mathbf{x}}$ instead of just $\mathbf{x}$. This final estimate accounts for the removal of the constraint $\mathbf{1}^\top \mathbf{x} = 1$.

Although the objective function in Equation (41) is indeed correct for all optimal vertices to the LP, the main drawback of this formulation is that the returned vertices often have a small reachability probability. In other words, the probability of transitioning from the estimated $\mathbf{b}_{t-1} \approx \frac{\mathbf{x}}{\mathbf{1}^\top \mathbf{x}}$ to $\mathbf{b}_t$ occasionally ends up being very small, thereby causing the BMCTS to explore unlikely regions of the belief-space. To remedy this, we propose one last constraint to the LP formulation in Equation (41) as follows:

$$\begin{aligned} P(o_i \mid a^j, \mathbf{x}) &= \sum_s \sum_{s'} Z(o_i \mid s') T(s' \mid a^j, s) \frac{\mathbf{x}^s}{\mathbf{1}^\top \mathbf{x}} \\ &\geq z \end{aligned} \tag{42}$$

where $\mathbf{x}^s$ denotes the element of $\mathbf{x}$ corresponding to state s, and $1 > z > 0$ is a dynamically set hyperparameter (see Section A.4).[13] Keeping the value of z relatively high allows the overall paths in the tree to have reasonably high probabilities. Notice that the constraint above can be rewritten as:

$$\frac{\hat{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x}}{\mathbf{1}^\top \mathbf{x}} \geq z \tag{43}$$

where

$$\begin{aligned} \hat{\mathbf{O}}_{o_i} &= \begin{bmatrix} Z(o_i \mid s'_1) & \cdots & Z(o_i \mid s'_N) \end{bmatrix} \\ &\equiv \text{diag}\left(\bar{\mathbf{O}}_{o_i}\right). \end{aligned} \tag{18}$$

[12] To understand why this might happen, we can think of two optimal vertices $\mathbf{x_1}$ and $\mathbf{x_2}$ to Equation (39). Despite the fact that both $\mathbf{x_1}$ and $\mathbf{x_2}$ are valid probability distributions (both satisfy Equations (36) and (37)), the values of $\bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x_1}$ and $\bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x_2}$ could be on different scales (if $y_1 \neq y_2$, for solutions $(\mathbf{x_1}, y_1)$ and $(\mathbf{x_2}, y_2)$). In other words, we would observe $\frac{\bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x_1}}{y_1} = \frac{\bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x_2}}{y_2}$, but $y_1 \mathbf{b}_t - \bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x_1} \neq y_2 \mathbf{b}_t - \bar{\mathbf{O}}_{o_i} \bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} \mathbf{x_2}$, and therefore one of these solutions would be missed.

[13] We have strict inequalities here: If $z = 0$ were to be allowed, this would imply a solution $\mathbf{x}$ that has zero probability of reaching $\mathbf{b}_t$, which is not of interest to us. Having $z = 1$ would require full-observability, which is not the case for a POMDP, and therefore over-constraints the problem.

We can drop the denominator in Equation (43) by re-framing the same expression:

$$\left(\hat{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} - z\mathbf{1}^\top\right)\mathbf{x} \geq 0. \tag{13}$$

Hence, we have our linear program:

$$\begin{aligned} \min_{\mathbf{x}} \; & ||\mathbf{b}_t - \bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\mathbf{x}||_1 \qquad (44) \\ \text{s.t.} \quad & \text{Equations (9), (10), (11), (12) and (13)} \end{aligned}$$

where our estimate of $\mathbf{b}_{t-1}^{ij}$ is again $\frac{\mathbf{x}}{\mathbf{1}^\top\mathbf{x}}$.

The final step is to convert the objective function of Equation (44) containing an L1 norm to an entirely linear relation by defining the following two new sets of constraints:

$$\mathbf{b}_t - \bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\mathbf{x} \leq \mathbf{u} \tag{14}$$

$$\mathbf{b}_t - \bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\mathbf{x} \geq -\mathbf{u} \tag{15}$$

where the vector $\mathbf{u}$ is a slack variable. Correspondingly, the objective function of our LP can now be written $\min_{\mathbf{x},\mathbf{u}} \mathbf{1}^\top\mathbf{u}$. Along with the constraints in Equations (14) and (15), this new objective function is equivalent to the one in Equation (44), but now has a linear relation.

Hence, the complete formulation of our linear program, for a given observation o_i and the action a^j corresponding to the alpha-vector $\boldsymbol{\alpha}_j \in \Gamma$, is as follows:

$$\begin{aligned} & \min_{\mathbf{x},\mathbf{u}} \mathbf{1}^\top\mathbf{u} && (8) \\ \text{s.t.} \quad & \boldsymbol{\alpha}_j\mathbf{x} \geq \boldsymbol{\alpha}_k\mathbf{x} \quad \forall \boldsymbol{\alpha}_k \in \Gamma && (9) \\ & \mathbf{x}^k = 0 \quad \forall \operatorname{col}^k\left(\bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\right) = \mathbf{0} && (10) \\ & \mathbf{x} \geq \mathbf{0} && (11) \\ & \mathbf{1}^\top\bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\mathbf{x} = 1 && (12) \\ & (\hat{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j} - z\mathbf{1}^\top)\mathbf{x} \geq 0 && (13) \\ & \mathbf{b}_t - \bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\mathbf{x} \leq \mathbf{u} && (14) \\ & \mathbf{b}_t - \bar{\mathbf{O}}_{o_i}\bar{\mathbf{T}}_{\boldsymbol{\alpha}_j}\mathbf{x} \geq -\mathbf{u} && (15) \end{aligned}$$

where we compute the previous $\mathbf{b}_{t-1}^{ij}$ at time $t-1$, given the belief $\mathbf{b}_t$ at time t, as $\frac{\mathbf{x}}{\mathbf{1}^\top\mathbf{x}}$.

Received 3 January 2025; accepted 1 December 2025.