BetaZero: Belief-State Planning for Long-Horizon POMDPs using Learned Approximations

Robert J. Moss, Anthony Corso, Jef Caers, Mykel J. Kochenderfer
Stanford University, {mossr, acorso, caers, mykel}@stanford.edu

Abstract

Real-world planning problems, including autonomous driving and sustainable energy applications like carbon storage and resource exploration, have recently been modeled as partially observable Markov decision processes (POMDPs) and solved using approximate methods. To solve high-dimensional POMDPs in practice, state-of-the-art methods use online planning with problem-specific heuristics to reduce planning horizons and make the problems tractable. Algorithms that learn approximations to replace heuristics have recently found success in large-scale fully observable domains. The key insight is the combination of online Monte Carlo tree search with offline neural network approximations of the optimal policy and value function. In this work, we bring this insight to partially observable domains and propose *BetaZero*, a belief-state planning algorithm for high-dimensional POMDPs. BetaZero learns offline approximations that replace heuristics to enable online decision making in long-horizon problems. We address several challenges inherent in large-scale partially observable domains; namely challenges of transitioning in stochastic environments, prioritizing action branching with a limited search budget, and representing beliefs as input to the network. To formalize the use of all limited search information, we train against a novel Q-weighted visit counts policy. We test BetaZero on various well-established POMDP benchmarks found in the literature and a real-world problem of critical mineral exploration. Experiments show that BetaZero outperforms state-of-the-art POMDP solvers on a variety of tasks.[1]

1 Introduction

Optimizing sequential decisions in real-world settings is challenging due to uncertainties about the true state of the environment. Modeling such problems as partially observable Markov decision processes (POMDPs) has shown recent success in autonomous driving (Wray et al., 2021), robotics (Lauri et al., 2022), and aircraft collision avoidance (Kochenderfer et al., 2012). Solving large or continuous POMDPs require approximations in the form of state-space discretizations or modeling assumptions, e.g., assuming full observability. Although these approximations are useful when making decisions in a short time horizon, scaling these solutions to long-horizon problems is challenging (Shani et al., 2013). Recently, POMDPs have been used to model large-scale information gathering problems such as carbon capture and storage (CCS) (Corso et al., 2022; Wang et al., 2023), remediation for groundwater contamination (Wang et al., 2022), and critical mineral exploration for battery metals (Mern & Caers, 2023), and are solved using online tree search methods such as DESPOT (Ye et al., 2017) and POMCPOW (Sunberg & Kochenderfer, 2018). The performance of these online methods rely on heuristics for action selection (to reduce search tree expansion) and heuristics to estimate the value function (to avoid expensive rollouts and reduce tree search depth). Without heuristics, online methods have difficulty planning for long-term information acquisition to reason about uncertain future events. Thus, algorithms to solve high-dimensional POMDPs need to be designed to learn heuristic approximations to enable decision making in long-horizon problems.

[1] Code: `https://github.com/sisl/BetaZero.jl`

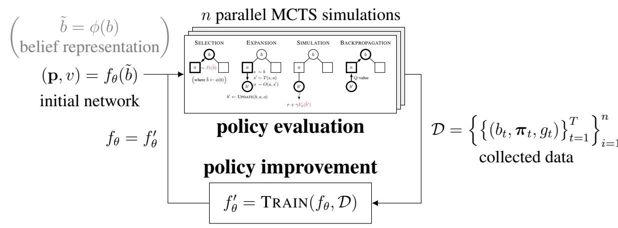

Figure 1: The *BetaZero* POMDP policy iteration algorithm.

Contributions. This work aims to address the problem of high-dimensional, long-horizon POMDPs by using the insight of combining online MCTS planning with learned offline neural network approximations that replace heuristics. Our main contribution is the *BetaZero* belief-state planning algorithm for POMDPs (fig. 1), addressing the challenges of partial observability in large discrete action spaces and continuous state and observation spaces. To handle stochastic belief-state transitions, BetaZero uses progressive widening (Couëtoux et al., 2011) to limit belief-state expansion. When planning in belief space, expensive belief updates *limit the search budget in practice* (e.g., $\mathcal{O}(n)$ for particle filters (Thrun et al., 2005) or $\mathcal{O}(n^3)$ for Kalman filters (Welch & Bishop, 1995)). Therefore, we sample from the policy network to prioritize branching on promising actions, and we introduce a novel *Q-weighted visit count* policy target that formalizes the use of all information seen during the limited search for policy imitation. While planning occurs over the full belief, we use a parametric belief representation $\tilde{b} = \phi(b)$ to capture state uncertainty as input to the network. BetaZero uses the learned policy network $P_\theta(\tilde{b})$ to reduce search breadth and the learned value estimate $V_\theta(\tilde{b})$ to reduce search depth to enable long-horizon online planning (shown in red in fig. 2).

2 Problem Formulation

A partially observable Markov decision process (POMDP) is a model for sequential decision making problems where the true state is unobservable. Defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$, POMDPs are an extension to the Markov decision process (MDP) used in reinforcement learning and planning with the addition of an observation space $\mathcal{O}$ (where $o \in \mathcal{O}$) and observation model $O(o \mid a, s')$. Given a current state $s \in \mathcal{S}$ and taking an action $a \in \mathcal{A}$, the agent transitions to a new state s' using the transition model $s' \sim T(\cdot \mid s, a)$. Without access to the true state, an observation is received $o \sim O(\cdot \mid a, s')$ and used to update the belief b over the possible next states s' to get the posterior

$$b'(s') \propto O(o \mid a, s') \int_{s \in \mathcal{S}} T(s' \mid s, a) b(s) \, \mathrm{d}s. \tag{1}$$

An example of a type of belief is the non-parametric *particle set* that can represent a broad range of distributions (Thrun et al., 2005), and Lim et al. (2023) show that optimality guarantees exist in finite-sample particle-based POMDP approximations. Despite choosing to study particle-based beliefs, our work generalizes well to problems with parametric beliefs.

A stochastic POMDP policy $\pi(a \mid b)$ is defined as the distribution over actions given the current belief b. After taking an action $a \sim \pi(\cdot \mid b)$, the agent receives a reward r from the environment according to the reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ or $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ using the next state.

Belief-state MDPs. In *belief-state* MDPs, the POMDP is converted to an MDP by treating the belief as a state (Kaelbling et al., 1998; Kochenderfer et al., 2022). The reward function then becomes a weighted sum of the state-based reward:

$$R_b(b, a) = \int_{s \in \mathcal{S}} b(s) R(s, a) \, \mathrm{d}s \approx \sum_{s \in b} b(s) R(s, a) \tag{2}$$

The belief-state MDP shares the same action space as the POMDP and operates over a belief space $\mathcal{B}$ that is the simplex over the state space $\mathcal{S}$. The belief-MDP defines a new belief-state transition function $b' \sim T_b(\cdot \mid b, a)$ as:

$$s \sim b(\cdot) \qquad s' \sim T(\cdot \mid s, a) \qquad o \sim O(\cdot \mid a, s') \qquad b' \leftarrow \textsc{Update}(b, a, o) \tag{3}$$

where the belief update can be done using a particle filter (Gordon et al., 1993). Therefore, the belief-state MDP is defined by the tuple $\langle \mathcal{B}, \mathcal{A}, T_b, R_b, \gamma \rangle$ with the finite-horizon discount factor $\gamma \in [0, 1)$ that controls the effect that future rewards have on the current action.

The objective to solve belief-MDPs is to find a policy π that maximizes the *value function*

$$V^{\pi}(b_0) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t R_b(b_t, a_t) \mid b_t \sim T_b, a_t \sim \pi\right] \tag{4}$$

from an initial belief b_0. Instead of explicitly constructing a policy over all beliefs, online planning algorithms estimate the next best action through a planning procedure, often a best-first tree search.

2.1 Monte Carlo tree search (MCTS)

Monte Carlo tree search (Coulom, 2007; Browne et al., 2012) is an online, recursive, best-first tree search algorithm to determine the approximately optimal action to take from a given root state of an MDP. Extensions to MCTS have been applied to POMDPs through several algorithms: *partially observable Monte Carlo planning* (POMCP) treats the state nodes as histories h of action-observation trajectories (Silver & Veness, 2010), *POMCP with observation widening* (POMCPOW) constructs weighted particle sets at the observation nodes and extends POMCP to fully continuous domains (Sunberg & Kochenderfer, 2018), and *particle filter trees* (PFT) and *information PFT* (IPFT) treat the POMDP as a belief-state MDP and plan directly over the belief-state nodes using a particle filter (Fischer & Tas, 2020). All variants of MCTS execute the following four steps. In this section we use s to represent the state, the history h, and the belief state b and refer to them as "the state".

1. **Selection.** During *selection*, an action is selected from the children of a state node based on criteria that balances exploration and exploitation. The *upper-confidence tree* algorithm (UCT) (Kocsis & Szepesvári, 2006) is a common criterion that selects an action that maximizes the upper-confidence bound $Q(s, a) + c\sqrt{\log N(s)/N(s, a)}$ where $Q(s, a)$ is the Q-value estimate for state-action pair (s, a) with a visit count of $N(s, a)$, the total visit count of $N(s) = \sum_a N(s, a)$ for the children $a \in A(s)$, and c is an exploration constant. Rosin (2011) introduced the *UCT with predictor* algorithm (PUCT), modified by Silver et al. (2017), where a predictor $P(s, a)$ guides the exploration towards promising branches and selects an action according to the following:

$$\underset{a \in A(s)}{\operatorname{argmax}}\; Q(s, a) + c\left(P(s, a)\frac{\sqrt{N(s)}}{1 + N(s, a)}\right) \tag{5}$$

2. **Expansion.** In the *expansion* step, the selected action is taken in simulation and the transition model $T(s' \mid s, a)$ is sampled to determine the next state s'. When the transitions are deterministic, the child node is always a single state. If the transition dynamics are stochastic, techniques to balance the branching factor such as progressive widening (Couëtoux et al., 2011) and state abstraction refinement (Sokota et al., 2021) have been proposed.

3. **Rollout/Simulation.** In the *rollout* step, also called the *simulation* step due to recursively simulating the MCTS tree expansion, the value is estimated through the execution of a rollout policy until termination or using heuristics to approximate the value function from the given state s'. Expensive rollouts done by AlphaGo were replaced with a value network lookup in AlphaGo Zero and AlphaZero (Silver et al., 2016; 2017; 2018).

4. **Backpropagation.** Finally, during the *backpropagation* step, the Q-value estimate from the rollout is propagated up the path in the tree as a running average.

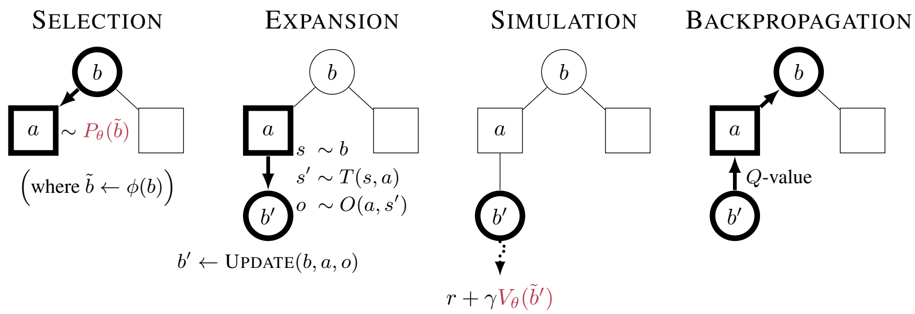

Figure 2: The four stages of MCTS belief-state planning in *BetaZero* using the value V_θ and policy P_θ network heads (the *policy evaluation* step in fig. 1).

Root node action selection. After repeating the four steps of MCTS, the final action is selected from the children $a \in A(s)$ of the root state s and executed in the environment. One way to select the best root node action, referred to as the *robust child* (Schadd, 2009; Browne et al., 2012), selects the action with the highest visit count as $\text{argmax}_a N(s, a)$. Sampling from the normalized counts, exponentiated by an exploratory temperature τ, is also common (Silver et al., 2017). Another method uses the highest estimated Q-value as $\text{argmax}_a Q(s, a)$. Both criteria have been shown to have problem-based trade-offs (Browne et al., 2012).

Double progressive widening. To handle stochastic state transitions and large or continuous state and action spaces, double progressive widening (DPW) balances between sampling new nodes to expand on or selecting from existing nodes already in the tree (Couëtoux et al., 2011). Two hyperparameters $\alpha \in [0, 1]$ and $k \geq 0$ control the branching factor. If the number of actions tried from state s is less than $kN(s)^\alpha$, then a new action is sampled from the action space and added as a child of node s. Likewise, if the number of expanded states from node (s, a) is less than $kN(s, a)^\alpha$, then a new state is sampled from the transition function $s' \sim T(\cdot \mid s, a)$ and added as a child. If the state widening condition is not met, then a next state is sampled from the existing children.

Note, in the following sections we will refer to the belief state as b and the true (hidden) state as s.

3 Proposed Algorithm: BetaZero

We introduce the *BetaZero* POMDP planning algorithm that replaces heuristics with learned approximations of the optimal policy and value function. BetaZero is a belief-space policy iteration algorithm with two *offline* steps that learn a network used *online*:

1. **Policy evaluation**: Evaluate the current value and policy network through n parallel episodes of MCTS (fig. 2) and collect training data: $\mathcal{D} = \left\{\{(b_t, \boldsymbol{\pi}_t, g_t)\}_{t=1}^T\right\}_{i=1}^n$
2. **Policy improvement**: Improve the estimated value function and policy by retraining the neural network parameters θ with data from the n_{buffer} most recent MCTS simulations.

The policy vector over actions $\mathbf{p} = P_\theta(\tilde{b}, \cdot)$ and the value $v = V_\theta(\tilde{b})$ are combined into a single network with two output heads $(\mathbf{p}, v) = f_\theta(\tilde{b})$; we refer to P_θ and V_θ separately for convenience. During *policy evaluation*, training data is collected from the outer POMDP loop. The belief b_t and the tree policy $\boldsymbol{\pi}_t$ are collected for each time step t. At the end of each episode, the returns $g_t = \sum_{i=t}^T \gamma^{(i-t)} r_i$ are computed from the set of observed rewards for all time steps up to a terminal horizon T. Traditionally, MCTS algorithms use a tree policy $\boldsymbol{\pi}_t$ that is proportional to the root node visit counts of its children actions $\boldsymbol{\pi}_t(b_t, a) \propto N(b_t, a)^{1/\tau}$. The counts are sampled after exponentiating with a temperature τ to encourage exploration but evaluated online with $\tau \to 0$ to return the maximizing action (Silver et al., 2017). In certain settings, relying solely on visit counts may overlook crucial information (see fig. 3).

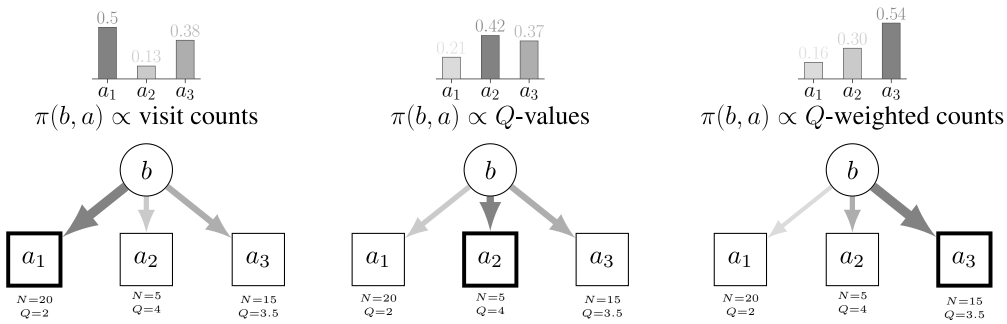

Figure 3: An illustrative example of when collecting policy data based purely on visit counts (left) or Q-values (middle) would perform worse than weighting the visit counts based on Q-values (right). This is useful when using a small MCTS budget with high exploration. Using both the Q-values *and* visit counts, we incorporate both what the tree search *focused on* and the *values it found*.

Policy vector as Q-weighted counts. When planning in belief space, expensive belief updates occur in the tree search and thus may limit the MCTS budget. Therefore, the visit counts may not converge towards an optimal strategy as the budget may be spent on exploration. Danihelka et al. (2022) and Czech et al. (2021) suggest using knowledge of the Q-values from search in MCTS action selection. Using only tree information, we incorporate Q-values and train against the policy

$$\boldsymbol{\pi}_t(b_t, a) \propto \left(\left(\frac{\exp Q(b_t, a)}{\sum_{a'} \exp Q(b_t, a')} \right)^{z_q} \left(\frac{N(b_t, a)}{\sum_{a'} N(b_t, a')} \right)^{z_n} \right)^{1/\tau} \tag{6}$$

which is then normalized to get a valid probability distribution. Equation (6) simply weights the visit counts by the softmax Q-value distribution with parameters $z_q \in [0,1]$ and $z_n \in [0,1]$ defining the influence of the values and the visit counts, respectively. If $z_q = z_n = 1$, then the influence is equal and if $z_q = z_n = 0$, then the policy becomes uniform. Once the tree search finishes, the root node action is selected from $a \sim \boldsymbol{\pi}_t(b_t, \cdot)$ and returns the argmax when the temperature $\tau \to 0$.

Loss function. Using the latest collected data, the *policy improvement* step retrains the policy network head using the cross-entropy loss $\mathcal{L}_{P_\theta}(\boldsymbol{\pi}_t, \mathbf{p}_t) = -\boldsymbol{\pi}_t^\top \log \mathbf{p}_t$. The value network head is simultaneously trained to fit the returns g_t using mean-squared error (MSE) or mean-absolute error (MAE) to predict the value of the belief b_t. Note that we use either MSE or MAE value losses $\mathcal{L}_{V_\theta}$ for different problems depending on the characteristics of the return distribution. In sparse reward problems, MAE is a better choice as the distribution is closer to Laplacian (Hodson, 2022). When the reward is distributed closer to Gaussian, then MSE is more suitable (Chai & Draxler, 2014). The final loss function combines the value and policy losses with L_2-regularization scaled by λ:

$$\ell_{\beta_0} = \mathcal{L}_{V_\theta}(g_t, v_t) + \mathcal{L}_{P_\theta}(\boldsymbol{\pi}_t, \mathbf{p}_t) + \lambda \|\theta\|^2 \tag{7}$$

Prioritized action widening. Planning in belief space explicitly handles state uncertainty but may incur computational overhead when performing belief updates, therefore we avoid trying all actions at every belief node. We apply *action progressive widening* (Couëtoux et al., 2011) to limit action expansion, which has been used in the context of continuous action spaces (Moerland et al., 2018) and large discrete action spaces (Yee et al., 2016). Browne et al. (2012) found action progressive widening to be effective in cases where favorable actions were tried first and Mern et al. (2021) show that prioritizing actions can improve MCTS performance in large discrete action spaces. Therefore, BetaZero selects actions through progressive widening and uses information from the learned policy network to sample new actions $a \sim P_\theta(\tilde{b}, \cdot)$, line 4, alg. 1. This way, we first focus the expansion on *promising actions*, then make the final selection based on PUCT.[2] In section 5, we perform an ablation to measure the effect of using the policy P_θ to prioritize actions when widening the tree.

[2]PUCT uses normalized Q-values from 0 to 1 ($\bar{Q}$) so c can be problem independent (Schrittwieser et al., 2020).

Algorithm 1: BetaZero action progressive widening.

1 **function** ActionSelection(f_θ, b)
2 $\quad \tilde{b} \leftarrow \phi(b)$ ▷ belief representation
3 $\quad$ **if** $|A(b)| \leq k_a N(b)^{\alpha_a}$ ▷ action progressive widening
4 $\quad\quad a \sim P_\theta(\tilde{b}, \cdot)$ ▷ prioritized from network
5 $\quad\quad N(b,a) \leftarrow N_0(b,a)$
6 $\quad\quad Q(b,a) \leftarrow Q_0(b,a)$ ▷ bootstrap initial Q-value
7 $\quad\quad A(b) \leftarrow A(b) \cup \{a\}$ ▷ add to visited actions $A(b)$
8 $\quad$ **return** $\underset{a \in A(b)}{\text{argmax}}\ \bar{Q}(b,a) + c\left(P_\theta(\tilde{b},a)\frac{\sqrt{N(b)}}{1+N(b,a)}\right)$

Algorithm 2: BetaZero belief-state progressive widening.

function BeliefStateExpansion(b, a)
$\quad$ **if** $|B(b,a)| \leq k_b N(b,a)^{\alpha_b}$ ▷ belief progressive widening
$\quad\quad b' \sim T_b(\cdot \mid b, a)$ ▷ eq. (3)
$\quad\quad B(b,a) \leftarrow B(b,a) \cup \{b'\}$ ▷ add to visited beliefs
$\quad$ **else**
$\quad\quad b' \sim B(b,a)$ ▷ sample from belief-states in the tree
$\quad r \leftarrow R(b,a)$ or $r \leftarrow R(b,a,b')$
$\quad$ **return** b', r

Stochastic belief-state transitions. A challenge with partially observable domains is handling non-deterministic belief-state transitions in the tree search. The belief-state transition function T_b consists of several stochastic components and the belief is continuous (being a probability distribution over states). To address this, we use progressive widening from Couëtoux et al. (2011) (algorithm 2). Other methods for state expansion, like *state abstraction refinement* from Sokota et al. (2021), rely on problem-specific distance metrics between states to perform a nearest neighbor search. Progressive widening avoids problem-specific heuristics by using information only available in the search tree to provide artificially limited belief-state branching. Limited branching is important as the belief updates can be computationally expensive, thus limiting the MCTS search budget in practice.

Parametric belief representation. Inputting state histories into the network has been done in the literature, in both the context of MDPs (Silver et al., 2018) and POMDPs (Cai & Hsu, 2022) Using only state information does not generalize to complex POMDPs (seen in fig. 8), therefore, a representation of the belief is required. Although a particle belief is not parametrically defined, approximating the belief as summary statistics (e.g., mean and std) may capture enough information for value and policy estimation to be used during planning (Coquelin et al., 2008). Approximating the particle set parametrically is easy to implement and computationally inexpensive. We show that the approximation works well across various problems and, unsurprisingly, using only the mean state is inadequate (see section 5). We represent the particle set b parametrically as $\phi(b) = [\mu(b), \sigma(b)]$. BetaZero plans over the full belief b in the tree and only converts to the belief representation $\tilde{b} = \phi(b)$ for network evaluations. We do not depend on the *exact* way in which the belief is represented, so long as it captures state uncertainty. Coquelin et al. (2008) consider how to represent a particle filter belief as a finite set of features for policy gradient and suggest the approximation that consists of the mean and covariance, but only consider the class of policies depending on a single feature of the mean. Their work suggests that other features, such as entropy, could also be used. Other algorithms (e.g., FORBES from Chen et al. (2022)) could instead be used to learn this belief representation. Another example approach could use principle component analysis (PCA) to learn lower-dimensional features for belief representation (Roy et al., 2005).

Bootstrapping initial Q-values. The value network V_θ is used during the *simulation* step to replace rollouts with a network lookup (line 7, alg. 3). When a new state-action node is added to the tree, initial Q-values can also use the value network to bootstrap the estimate:

$$Q_0(b,a) \stackrel{\text{def}}{=} R_b(b,a) + \gamma V_\theta(\phi(b')) \quad \text{where} \quad b' \sim T_b(\cdot \mid b, a) \tag{8}$$

Bootstrapping occurs in algorithm 1 (line 6) and incurs an additional belief update through the belief-state transition T_b and may be opted only during online execution. The bootstrapped estimate is more robust (Kumar et al., 2019) and can be useful to initialize online search. Note that bootstrapping is also used in the model-free *MuZero* algorithm (Schrittwieser et al., 2020).

Complexity analysis. The runtime complexity of MCTS is $M = \mathcal{O}(ndm)$ for the n number of MCTS iterations (denoted n_{online} in algorithm 3), for the search depth d, and with a belief update over m particles at each belief-state node. The full complexity of BetaZero is $\mathcal{O}(pmTM/c)$ for p parallel runs (denoted n_{data} in algorithm 5), an episode horizon of T (each step updating the belief over m particles), the MCTS complexity of M, and the number of CPU cores c.

The memory complexity for MCTS is $E = \mathcal{O}(k^d)$ for $k = |A(b)||B(b,a)|$ where $|B(b,a)|$ is the number of belief-action nodes and $|A(b)|$ is the number of children, which depend on progressive widening parameters. The memory complexity for BetaZero is $\mathcal{O}(TPE|\theta|)$ for the collected data sizes of the belief and returns T (same as the horizon), the policy vector size of $P = |\mathcal{A}|$ (i.e., action space size), the MCTS memory complexity of E, and the network size of $|\theta|$. Compared to standard MCTS applications to belief-state MDPs, BetaZero requires additional memory for data collection and neural network storage.

Algorithm 3 details MCTS for BetaZero with extensions for belief-state planning with learned approximations. The full BetaZero algorithm is shown in algorithms 4 to 6.

Algorithm 3: BetaZero MCTS simulation.

function $\textsc{Simulate}(f_\theta, b, d)$
  if $d = 0$ **return** 0
  if $b \notin \mathcal{T}$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{b\}$
    $N(b) \leftarrow N_0(b)$
    $\tilde{b} \leftarrow \phi(b)$ ▷ belief representation
    return $V_\theta(\tilde{b})$ ▷ value lookup
  $N(b) \leftarrow N(b) + 1$
  $a \leftarrow \textsc{ActionSelection}(f_\theta, b)$
  $(b', r) \leftarrow \textsc{BeliefStateExpansion}(b, a)$
  $q \leftarrow r + \gamma\textsc{Simulate}(f_\theta, b', d-1)$
  $N(b,a) \leftarrow N(b,a) + 1$
  $Q(b,a) \leftarrow Q(b,a) + \frac{q - Q(b,a)}{N(b,a)}$
  return q

4 Related Work

Algorithms to solve high-dimensional, *fully observable* Markov decision processes (MDPs) have been proposed to learn approximations that replace problem-specific heuristics. Silver et al. (2018) introduced the *AlphaZero* algorithm for large, deterministic MDPs and showed considerable success in games such as Go, chess, shogi, and Atari (Silver et al., 2018; Schrittwieser et al., 2020). The success is attributed to the combination of online Monte Carlo tree search (MCTS) and a neural network that approximates the optimal value function and the offline policy. Extensions of AlphaZero and the model-free variant *MuZero* (Schrittwieser et al., 2020) have already addressed several challenges when applying to broad classes of MDPs. For large or continuous action spaces, Hubert et al. (2021) introduced a policy improvement algorithm called *Sampled MuZero* that samples an action set of an *a priori* fixed size every time a node is expanded. Antonoglou et al. (2021) introduced *Stochastic MuZero* that extends MuZero to games with stochastic transitions but assumes a finite set of possible next states so that each transition can be associated with a chance outcome. Applying these algorithms to large or continuous spaces with partially observability remains challenging.

To handle partial observability in stochastic games, Ozair et al. (2021) combine VQ-VAEs with MuZero to encode future discrete observations into latent variables. Other approaches handle partial observability by inputting action-observation histories directly into the network (Kimura et al., 2020; Vinyals et al., 2019). Similarly, Igl et al. (2018) introduce a method to learn a belief representation within the network when the agent is only given access to histories. Their work focuses on the *reinforcement learning* (RL) domain and they show that a belief distribution can be represented as a latent state in the learned model. The FORBES algorithm (Chen et al., 2022) builds a normalizing flow-based belief and learns a policy through an actor-critic RL algorithm. Methods to learn the belief are necessary when a prior belief model is not available. When such models *do* exist, as is the case with many POMDPs that we study, using the models can be valuable for long-term planning. Hoel et al. (2019) apply AlphaGo Zero (Silver et al., 2017) to an autonomous driving POMDP using the most-likely state as the network input but overlook significant belief uncertainty information.

Planning vs. reinforcement learning. In POMDP *planning*, models of the transitions, rewards, and observations are known. In contrast, in the model-based partially observable reinforcement learning (PORL) domain, these models are learned along with a policy or value function (Sutton & Barto, 2018; Subramanian et al., 2022). A difference between these settings is that PORL algorithms reset the agent and learn through experience, while planning algorithms, like MCTS, must consider future trajectories from any state. When RL problems have deterministic state transitions, they can be cast as a planning problem by replaying the full state trajectory along a tree path, which may be prohibitively expensive for long-horizon problems. Both settings are closely related and pose interesting research challenges. Specifically, sequential planning over given models in high-dimensional, long-horizon POMDPs remains challenging (Lauri et al., 2022).

Online POMDP planning. Sunberg & Kochenderfer (2018) introduced the *POMCPOW* planning algorithm that iteratively builds a particle set belief within the tree, designed for fully continuous spaces. In practice, POMCPOW relies on heuristics for value function estimation and action selection (e.g., work from Mern & Caers (2023)). Wu et al. (2021b) introduced *AdaOPS* that adaptively approximates the belief through particle filtering and maintains value function bounds that are initialized with heuristics (e.g., solving the MDP or using expert policies). The major limitation of existing solvers is the reliance on heuristics to make long-horizon POMDPs tractable, which may not scale to high-dimensional problems. Cai & Hsu (2022) proposed *LeTS-Drive* applied to autonomous driving that combines planning and learning similar to BetaZero, and uses HyP-DESPOT with PUCT exploration (Cai et al., 2021) as the planning algorithm, instead of MCTS. It uses a state-history window as input to the network, which may not adequately capture the state uncertainty. LeTS-Drive expands on all actions during planning, which we show may lead to suboptimal planning under limited search budgets (shown in figs. 14 and 16). To handle long-horizon POMDPs, Mazzi et al. (2023) propose learning logic-based rules as policy guidance in POMCP, yet domain-specific knowledge is required to define the set of features for the rules, which may not be easily generalized to complex POMDPs we study in this work. Therefore, we identified the need for a general POMDP planning algorithm that does not rely on problem-specific heuristics for good performance.

5 Experiments

Three benchmark problems were chosen to evaluate the performance of BetaZero. Figure 4 details the POMDP sizes and appendices further describe the POMDPs, network architectures, and experimental design.

	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{O}\rvert$
LightDark(5 and 10)	$\lvert\mathbb{R}\rvert$	3	$\lvert\mathbb{R}\rvert$
RockSample(15, 15)	7,372,800	20	3
RockSample(20, 20)	419,430,400	25	3
Mineral Exploration	$\lvert\mathbb{R}^{32\times 32}\rvert$	38	$\lvert\mathbb{R}_{\geq 0}\rvert$

Figure 4: POMDP space dimensions.

In LightDark(y) from Platt Jr. et al. (2010), the goal of the agent is to execute a `stop` action at the origin while receiving noisy observations of its true location. The noise is minimized in the "light" region $y = 5$. We also benchmark against a more challenging version with the light region at $y = 10$ from Sunberg & Kochenderfer (2018), and restrict the agent to only three actions: move `up` or `down` by one, or `stop`. The modified problem requires information gathering over longer horizons. Next is the RockSample(n, k) POMDP (Smith & Simmons, 2004), which is a scalable information gathering problem where an agent moves in an $n \times n$ grid to observe k rocks with an objective to sample only the "good" rocks. Well-established POMDP benchmarks go up to $n = 15$ and $k = 15$; we also test a harder version with $n = 20$ and $k = 20$ to show the scalability of BetaZero, noting that this case has been evaluated in the multi-agent setting (Cai et al., 2021). Finally, in the real-world Mineral Exploration problem (Mern & Caers, 2023), the agent drills over a 32×32 region to determine if a subsurface ore body should be mined or abandoned and the continuous ore quality is observed at the drill locations to build a belief. Drilling incurs a penalty, and if chosen to mine, then the agent is rewarded or penalized based on an economic threshold of the extracted ore mass. The problem is challenging due to reasoning over limited observations with sparse rewards.

We baseline BetaZero against several online POMDP algorithms, namely AdaOPS, POMCPOW, DESPOT, and LeTS-Drive (HyP-DESPOT with a learned network). In LightDark, we solve for an approximately optimal policy using *local approximation value iteration* (LAVI) (Kochenderfer, 2015) over a discretized parametric belief space, and for mineral exploration, the value estimates come from privileged information described in the appendix. For a fair comparison, parameters were set to roughly match the total number of simulations of about one million per algorithm.

5.1 Empirical results and discussion

Table 1 shows that BetaZero outperforms state-of-the-art algorithms in most cases, with larger improvements when baseline algorithms do not rely on heuristics. While BetaZero has a large offline timing component, similar to LeTS-Drive, it is significantly less than solving for the approximately

	LightDark(5)		LightDark(10)		RockSample(15, 15)		RockSample(20, 20)		Mineral Exploration	
	returns	time [*off*,*on*] s	returns	time [*off*,*on*] s	returns	time [*off*,*on*] s	returns	time [*off*,*on*] s	returns	time [*off*,*on*] s
BetaZero	$\mathbf{4.47 \pm 0.28}$	[2274, 0.014]	$\mathbf{16.77 \pm 1.28}$	[2740, 0.331]	20.15 ± 0.71	[5701, 0.477]	$\mathbf{13.09 \pm 0.55}$	[7081, 1.109]	$\mathbf{10.67 \pm 2.25}$	[22505, 5.126]
Raw Policy P_θ	4.44 ± 0.28	[2274, 0.004]	13.74 ± 1.33	[2740, 0.004]	10.96 ± 0.98	[5701, 0.018]	2.03 ± 0.34	[7081, 0.084]	8.67 ± 2.52	[22505, 0.533]
Raw Value $V_\theta{}^*$	3.16 ± 0.40	[2274, 0.008]	12.70 ± 1.46	[2740, 0.009]	9.96 ± 0.65	[5701, 0.158]	3.57 ± 0.40	[7081, 0.204]	9.75 ± 2.42	[22505, 1.420]
AdaOPS	3.78 ± 0.27 (3.79 ± 0.07)	[68, 0.089]	5.22 ± 1.77	[81, 0.510]	$\mathbf{20.67 \pm 0.72}$ (17.16 ± 0.21)	[7, 2.768]	—	—	3.33 ± 1.95	[5, 0.112]
AdaOPS (fixed bounds)	3.70 ± 0.25	[0, 0.039]	4.98 ± 2.01	[0, 0.573]	13.37 ± 0.71	[0, 1.349]	11.66 ± 0.49	[1, 1.458]	"	"
POMCPOW	3.21 ± 0.38 (3.23 ± 0.11)	[59, 0.189]	0.68 ± 0.41	[70, 1.261]	11.14 ± 0.59 (10.40 ± 0.18)	[0, 0.929]	10.22 ± 0.47	[0, 1.480]	9.43 ± 2.19	[0, 6.728]
POMCPOW (no heuristics)	1.96 ± 0.58	[0, 0.099]	-5.90 ± 5.78	[0, 0.742]	10.17 ± 0.61	[0, 1.485]	4.03 ± 0.44	[0, 5.173]	5.38 ± 2.15	[0, 5.915]
DESPOT	2.37 ± 0.37 (2.50 ± 0.10)	[0, 0.008]	0.43 ± 0.36	[0, 0.046]	18.44 ± 0.69 (15.67 ± 0.20)	[7, 3.822]	—	—	5.29 ± 2.17	[5, 0.283]
DESPOT (fixed bounds)	2.70 ± 0.50	[0, 0.008]	0.49 ± 0.30	[0, 0.025]	4.29 ± 0.45	[0, 5.091]	0.00	[0, 5.179]	"	"
LeTS-Drive (HyP-DESPOT + f_θ)	3.05 ± 0.25	[1260, 0.019]	4.08 ± 5.48	[1529, 0.058]	11.22 ± 0.27	[48064, 1.576]	9.68 ± 0.25	[63850, 2.018]	3.17 ± 2.04	[11738, 4.613]
Approx. Optimal	4.06 ± 0.31	[18359, 0.094]	15.04 ± 1.27	[19548, 0.024]	—	—	—	—	11.90 ± 0.18	N/A

[*] One-step look-ahead over all actions using only the value network with 5 observations per action.
Entries with "—" failed to run, " are the same as the ones above, and entries in (parentheses) are from the literature.

Table 1: Results comparing *BetaZero* to various state-of-the-art POMDP solvers. Reporting return mean and standard error over 100 seeds, and [*offline*, *online*] timing in seconds.

optimal policy. Figure 5 compares the raw BetaZero value and policy network with *value iteration* for LightDark(10). Qualitatively, BetaZero learns an accurate optimal policy and value function in areas where training data was collected. Areas where BetaZero and the approximately optimal policy diverge may be a result of a lack of training data in those regions (top right corners). Despite this, BetaZero remains nearly optimal as those beliefs do not occur during execution. Out-of-distribution methods could quantify this uncertainty, e.g., an ensemble of networks (Salehi et al., 2022).

In RockSample$(15, 15)$, BetaZero is comparable to AdaOPS yet scales better to higher dimensional problems such as the RockSample$(20, 20)$ POMDP. AdaOPS computes an upper bound using QMDP (Littman et al., 1995) to find the optimal utility of the fully observable MDP over all $k - 1$ rock combinations, which scales exponentially in n. In problems with higher state space dimensions, like RockSample$(20, 20)$, the QMDP solution is intractable. Thus, fixed bounds are used in AdaOPS assuming an optimistic $V_{\max}$ (Wu et al., 2021a). The appendix further details the heuristics used by the baseline algorithms. Indicated in table 1, the raw networks alone perform well but outperform when combined with online planning, enabling reasoning with current information.

If online algorithms ran for a large number of iterations, one might expect to see convergence to the optimal policy. In practice, this may be an intractable number as fig. 6 shows POMCPOW has not reached the required number of iterations for RockSample. The advantage of BetaZero is that it can generalize from a more diverse set of experiences. The inability of existing online algorithms to plan over long horizons is also evident in the mineral exploration POMDP (fig. 7). POMCPOW

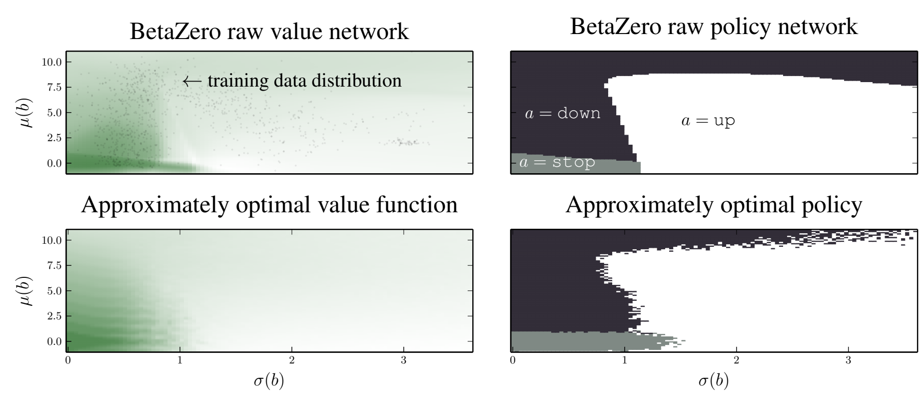

Figure 5: LightDark(10) value and policy plots over belief mean and std. High uncertainty (horizontal axis) makes the agent localize `up` near $y = 10$, then moves `down` and `stops` at the origin.

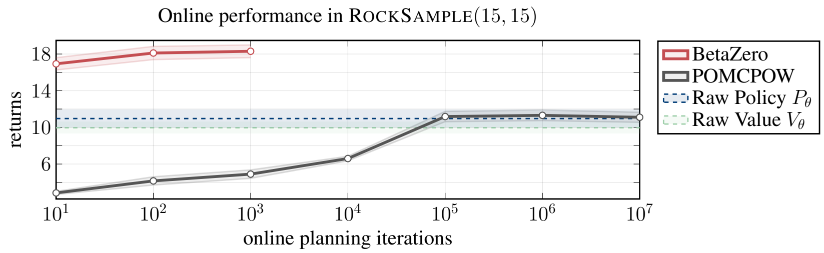

Figure 6: Performance of POMCPOW with heuristics up to 10 million *online* iterations plateaus, indicating that extending online searches alone misses valuable *offline* experience.

ran for one million online iterations without a value estimator heuristic and BetaZero ran online for 100 iterations (using about 850,000 offline simulations). In the figure, the probability of selecting a drilling location is shown as vertical bars for each action, overlaid on the initial belief uncertainty (i.e., the std of the belief in subsurface ore quality). BetaZero learned to take actions in areas of the belief space with high uncertainty and high value (which matches the domain-specific heuristics developed for the mineral exploration problem from Mern & Caers (2023)), while POMCPOW fails to distinguish between the actions and resembles a uniform policy.

The most closely related algorithm, LeTS-Drive (Cai & Hsu, 2022), which also includes an offline learning component with online tree search planning, performs better than its DESPOT counterpart without the use of offline heuristics. This is observed in all studied POMDPs except for the mineral exploration problem where the DESPOT bounds use privileged information from the approximately optimal bounds on the value function. Table 1 highlights that LeTS-Drive is able to scale DESPOT to the RockSample(20, 20) problem, with overall similar timing results as BetaZero but worse performance. This could be attributed to the HyP-DESPOT online tree search used in LeTS-Drive that plans over observation space (similar to POMCPOW) and implicitly constructs beliefs from a set of K scenarios in the tree. Therefore, the beliefs are dependent on the number of in-tree scenarios executed, hence the comparable timing results, and not on the actual root node belief that is updated along the tree paths (where belief-state planning incurs different computational expense but with the benefit of planning over reachable beliefs into the future). Instead of the state history as network input, we use the in-tree belief for a better comparison. The HyP-DESPOT planner expands the tree over *all* actions instead of using progressive widening with prioritization, and, as we observe in the ablation studies in the next section, expanding on all actions may limit the effective use of the tree search budget, thus potentially missing promising areas of the reachable futures.

Ablation studies. To test the effect of each contribution, we run several ablation studies. The influence of value and visit count information when selecting an action is shown in fig. 9. Each cell is the mean return for the RockSample(20, 20) problem over 100 online trials, selecting root-node actions via the argmax of eq. (6) given z_q and z_n. The cell at $(0, 0)$ corresponds to a uniform policy

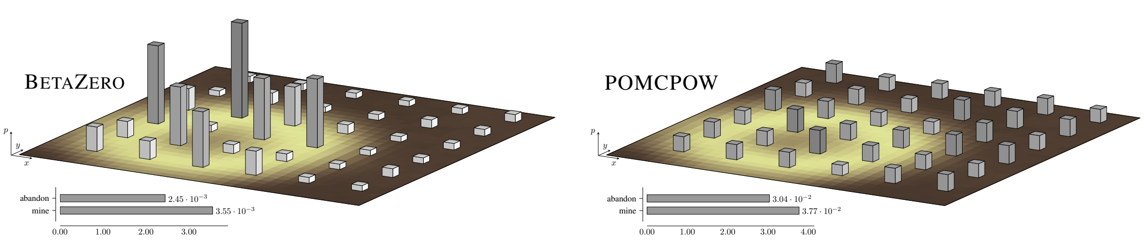

Figure 7: Mineral exploration policies: BetaZero prioritizes uncertainty, matching heuristics from Mern & Caers (2023) (i.e., select action with high uncertainty, shown in yellow).

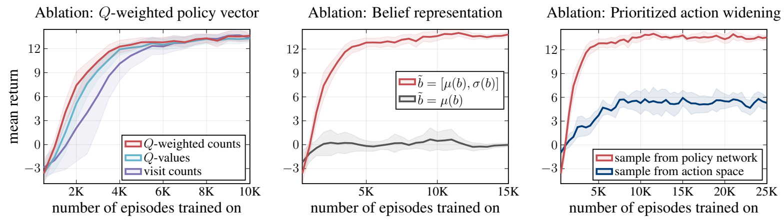

Figure 8: LightDark(10) ablation study. (Left) Learning is faster when the network is trained using Q-weighted visit counts. (Middle) Incorporating belief uncertainty is crucial for learning. (Right) Action widening from the policy network shows significant improvement. The same red curves are shown with varying horizontal axes, and one std is shaded from three seeds using 0.6 exponential smoothing.

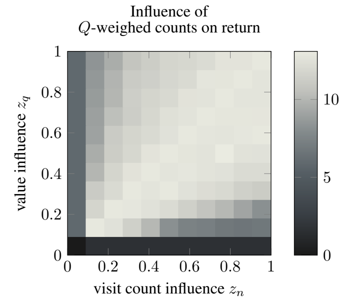

Figure 9: Ablation study in RockSample(20, 20). Combining value and count information leads to the highest return. The diagonal is identical due to the argmax of eq. (6).

and thus samples actions instead. Using only the visit counts (bottom cells) or only the values (left cells) to make decisions is worse than using a combination of the two. The effect of the Q-weighting is also shown in the leftmost fig. 8, which suggests that it helps learn faster in LightDark(10).

Unsurprisingly, using the *state uncertainty* encoded in the belief is crucial for learning as indicated in the middle of fig. 8. Future work could directly input the particle set into the network, first passing through an order invariant layer (Zaheer et al., 2017), to offload the belief approximation to the network itself. Finally, the rightmost plot in fig. 8 suggests that when branching on actions using progressive widening, it is important to first prioritize the actions suggested by the policy network. Offline learning fails if instead we sample uniformly from the action space (even in the LightDark case with only three actions).

6 Conclusions

We propose the *BetaZero* belief-state planning algorithm for POMDPs; designed to learn from *offline* experience to inform *online* decisions. Planning in belief space explicitly handles state uncertainty and learning offline approximations to replace heuristics enables effective online planning in long-horizon POMDPs. Although belief-space planning incurs expensive belief updates in the tree search, we address the limited search budget used in practice by incorporating all information available in the search tree to (a) train the policy vector target (using the Q-weighted visit counts), and (b) sample from the policy network during action progressive widening to prioritize promising actions. Stochastic belief-state transitions in MCTS are addressed using *progressive widening* and we test a belief representation of summary statistics to allow beliefs as input to the value and policy network. Results indicate that BetaZero scales to larger problems where certain heuristics break down and, as a result, can solve large-scale POMDPs by learning to plan in belief space using zero heuristics.

Limitations. It is standard for POMDP planning algorithms to assume known models but this may limit the applicability to certain problems where reinforcement learning may be better suited. We chose a simplified belief representation to allow for further research innovations in using other parametric and non-parametric representations. Other limitations include compute resource requirements for training neural networks and parallelizing MCTS simulations. We designed BetaZero to use a single GPU for training and to scale based on available CPUs. Certain POMDPs may not require this training burden, especially when known heuristics perform well. BetaZero is useful for long-horizon, high-dimensional POMDPs but may be unnecessary when offline training is computationally limited. BetaZero is designed for problems where the simulation cost is the dominating factor compared to offline training time.

Acknowledgments

We would like to thank Johannes Fischer, Arec Jamgochian, Markus Zechner, Dylan Asmar and Ashwin Kanhere for their insights and comments on this work. This work is supported by funding from the Stanford Institute for Human-Centered AI (HAI), Stanford Mineral-X, and OMV.

References

Ioannis Antonoglou, Julian Schrittwieser, Sherjil Ozair, Thomas K. Hubert, and David Silver. Planning in Stochastic Environments with a Learned Model. In *International Conference on Learning Representations (ICLR)*, 2021.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

Panpan Cai and David Hsu. Closing the Planning–Learning Loop With Application to Autonomous Driving. *IEEE Transactions on Robotics*, 39(2):998–1011, 2022.

Panpan Cai, Yuanfu Luo, David Hsu, and Wee Sun Lee. HyP-DESPOT: A hybrid parallel algorithm for online planning under uncertainty. *The International Journal of Robotics Research*, 40(2-3): 558–573, 2021.

Tianfeng Chai and Roland R. Draxler. Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014.

Xiaoyu Chen, Yao Mark Mu, Ping Luo, Shengbo Li, and Jianyu Chen. Flow-Based Recurrent Belief State Learning for POMDPs. In *International Conference on Machine Learning (ICML)*, 2022.

Pierre-Arnaud Coquelin, Romain Deguest, and Rémi Munos. Particle Filter-Based Policy Gradient in POMDPs. *Advances in Neural Information Processing Systems (NIPS)*, 21, 2008.

Anthony Corso, Yizheng Wang, Markus Zechner, Jef Caers, and Mykel J. Kochenderfer. A POMDP Model for Safe Geological Carbon Sequestration. *NeurIPS Workshop on Tackling Climate Change with Machine Learning*, 2022.

Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous Upper Confidence Trees. In *Learning and Intelligent Optimization*. Springer, 2011.

Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Computers and Games*, 2007.

Katalin Csilléry, Michael G. B. Blum, Oscar E. Gaggiotti, and Olivier François. Approximate Bayesian Computation (ABC) in practice. *Trends in Ecology & Evolution*, 25(7):410–418, 2010.

Johannes Czech, Patrick Korus, and Kristian Kersting. Improving AlphaZero Using Monte-Carlo Graph Search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 31, pp. 103–111, 2021.

Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy Improvement by Planning with Gumbel. In *International Conference on Learning Representations (ICLR)*, 2022.

Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty. *Journal of Machine Learning Research*, 18(26):1–5, 2017.

Johannes Fischer and Ömer Sahin Tas. Information Particle Filter Tree: An Online Algorithm for POMDPs with Belief-Based Rewards on Continuous Domains. In *International Conference on Machine Learning (ICML)*, pp. 3177–3187. PMLR, 2020.

Neil J. Gordon, David J. Salmond, and Adrian F.M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEEE for Radar and Signal Processing*, volume 140, 1993.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural Networks for Machine Learning. Lecture 6a: Overview of Mini-Batch Gradient Descent. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2014.

Timothy O. Hodson. Root-mean-square error (RMSE) or mean absolute error (MAE): When to use them or not. *Geoscientific Model Development*, 15(14):5481–5487, 2022.

Carl-Johan Hoel, Katherine Driggs-Campbell, Krister Wolff, Leo Laine, and Mykel J. Kochenderfer. Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving. *IEEE Transactions on Intelligent Vehicles*, 5(2):294–305, 2019.

Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and Planning in Complex Action Spaces. In *International Conference on Machine Learning (ICML)*, pp. 4476–4486. PMLR, 2021.

Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep Variational Reinforcement Learning for POMDPs. In *International Conference on Machine Learning (ICML)*, pp. 2117–2126. PMLR, 2018.

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

Tomoaki Kimura, Katsuyoshi Sakamoto, and Tomah Sogabe. Development of AlphaZero-Based Reinforcment Learning Algorithm for Solving Partially Observable Markov Decision Process (POMDP) Problem. *Bulletin of Networking, Computing, Systems, and Software*, 9(1):69–73, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Mykel J. Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

Mykel J. Kochenderfer, Jessica E. Holland, and James P. Chryssanthacopoulos. Next-Generation Airborne Collision Avoidance System. *Lincoln Laboratory Journal*, 19(1), 2012.

Mykel J. Kochenderfer, Tim A. Wheeler, and Kyle H. Wray. *Algorithms for Decision Making*. MIT Press, 2022.

Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In *European Conference on Machine Learning*, pp. 282–293. Springer, 2006.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Mikko Lauri, David Hsu, and Joni Pajarinen. Partially Observable Markov Decision Processes in Robotics: A Survey. *IEEE Transactions on Robotics*, 2022.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In *Neural Networks: Tricks of the Trade*, pp. 9–50. Springer, 2002.

Michael H. Lim, Tyler J. Becker, Mykel J. Kochenderfer, Claire J. Tomlin, and Zachary N. Sunberg. Optimality Guarantees for Particle Belief Approximation of POMDPs. *Journal of Artificial Intelligence Research*, 77:1591–1636, 2023.

Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings*, pp. 362–370. Elsevier, 1995.

Giulio Mazzi, Daniele Meli, Alberto Castellini, and Alessandro Farinelli. Learning Logic Specifications for Soft Policy Guidance in POMCP. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 373–381, 2023.

John Mern and Jef Caers. The Intelligent Prospector v1.0: Geoscientific Model Development and Prediction by Sequential Data Acquisition Planning with Application to Mineral Exploration. *Geoscientific Model Development*, 16(1):289–313, 2023.

John Mern, Anil Yildiz, Lawrence Bush, Tapan Mukerji, and Mykel J. Kochenderfer. Improved POMDP Tree Search Planning with Prioritized Action Branching. *AAAI Conference on Artificial Intelligence (AAAI)*, 35(13):11888–11894, 2021.

Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. A0C: Alpha Zero in Continuous Action Space. *arXiv preprint arXiv:1805.09613*, 2018.

Sherjil Ozair, Yazhe Li, Ali Razavi, Ioannis Antonoglou, Aaron Van Den Oord, and Oriol Vinyals. Vector Quantized Models for Planning. In *International Conference on Machine Learning (ICML)*, pp. 8302–8313. PMLR, 2021.

Robert Platt Jr., Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief Space Planning Assuming Maximum Likelihood Observations. *Robotics: Science and Systems VI*, 2010.

Christopher D. Rosin. Multi-Armed Bandits with Episode Context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. Finding Approximate POMDP Solutions Through Belief Compression. *Journal of Artificial Intelligence Research*, 23:1–40, 2005.

Mohammadreza Salehi, Hossein Mirzaei, Dan Hendrycks, Yixuan Li, Mohammad Hossein Rohban, and Mohammad Sabokrou. A Unified Survey on Anomaly, Novelty, Open-Set, and Out of-Distribution Detection: Solutions and Future Challenges. *Transactions of Machine Learning Research*, 2022.

Frederik Christiaan Schadd. Monte-Carlo Search Techniques in the Modern Board Game Thurn and Taxis. *Maastricht University: Maastricht, The Netherlands*, 2009.

Julian Schrittwieser. MuZero Intuition. https://www.furidamu.org/blog/2020/12/22/muzero-intuition/, 12 2020.

Julian Schrittwieser et al. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 588(7839):604–609, 2020.

Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27:1–51, 2013.

David Silver and Joel Veness. Monte-Carlo Planning in Large POMDPs. *Advances in Neural Information Processing Systems (NIPS)*, 23, 2010.

David Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 2016.

David Silver et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676), 2017.

David Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419), 2018.

Trey Smith and Reid Simmons. Heuristic Search Value Iteration for POMDPs. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 520–527, 2004.

Samuel Sokota, Caleb Y. Ho, Zaheen Ahmad, and J. Zico Kolter. Monte Carlo Tree Search with Iteratively Refining State Abstractions. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:18698–18709, 2021.

Jayakumar Subramanian et al. Approximate Information State for Approximate Planning and Reinforcement Learning in Partially Observed Systems. *Journal of Machine Learning Research*, 23(1):483–565, 2022.

Zachary N. Sunberg and Mykel J. Kochenderfer. Online Algorithms for POMDPs with Continuous State, Action, and Observation Spaces. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 28, 2018.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

Oriol Vinyals et al. Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning. *Nature*, 575(7782):350–354, 2019.

Yizheng Wang, Markus Zechner, John Michael Mern, Mykel J. Kochenderfer, and Jef Karel Caers. A sequential decision-making framework with uncertainty quantification for groundwater management. *Advances in Water Resources*, 166:104266, 2022.

Yizheng Wang, Markus Zechner, Gege Wen, Anthony Louis Corso, John Michael Mern, Mykel J. Kochenderfer, and Jef Karel Caers. Optimizing Carbon Storage Operations for Long-Term Safety. *arXiv preprint arXiv:2304.09352*, 2023.

Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. *University of North Carolina at Chapel Hill*, 1995.

Kyle Hollins Wray, Bernard Lange, Arec Jamgochian, Stefan J. Witwicki, Atsuhide Kobashi, Sachin Hagaribommanahalli, and David Ilstrup. POMDPs for Safe Visibility Reasoning in Autonomous Vehicles. In *International Conference on Intelligence and Safety for Robotics (ISR)*, pp. 191–195. IEEE, 2021.

Chenyang Wu, Guoyu Yang, Zongzhang Zhang, Yang Yu, Dong Li, Wulong Liu, and Jianye Hao. OpenReview Response: Adaptive Online Packing-guided Search for POMDPs. https://openreview.net/forum?id=0zvTBoQb5PA&noteId=bJTC_0sZPzr, 8 2021a.

Chenyang Wu, Guoyu Yang, Zongzhang Zhang, Yang Yu, Dong Li, Wulong Liu, and Jianye Hao. Adaptive Online Packing-guided Search for POMDPs. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:28419–28430, 2021b.

Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. DESPOT: Online POMDP Planning with Regularization. *Journal of Artificial Intelligence Research*, 58:231–266, 2017.

Timothy Yee, Viliam Lisỳ, Michael H. Bowling, and S. Kambhampati. Monte Carlo Tree Search in Continuous Action Spaces with Execution Uncertainty. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 690–697, 2016.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, et al. Deep Sets. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Appendix

This section contains material detailing the POMDP environments and experiments, the ablation studies, additional analysis of bootstrapping and double progressive widening, the network architectures, hyperparameters and tuning, computational resources, information regarding open-source code for reproducibility, and the full BetaZero algorithm pseudocode.

A POMDP Environments

This section describes the benchmark POMDPs in detail, including the heuristics used by the baseline POMDP algorithms and information regarding the particle filter belief used by BetaZero.

Light dark. The LightDark(y) POMDP is a one-dimensional localization problem (Platt Jr. et al., 2010). The objective is for the agent to execute the *stop* action at the goal, which is at ± 1 of the origin. The agent is awarded 100 for stopping at the goal and -100 for stopping anywhere else; using a discount of $\gamma = 0.9$. The agent receives noisy observations of their position, where the noise is minimized in the "light" region defined by y. In the LightDark(5) problem used by Wu et al. (2021b), the noise is a zero-mean Gaussian with standard deviation of $|y-5|/\sqrt{2}+10^{-2}$. For the LightDark(10) problem used by Sunberg & Kochenderfer (2018), the noise is a zero-mean Gaussian with standard deviation of $|y-10|+10^{-4}$. In both problems, we use a restricted action space of $\mathcal{A} = [-1, 0, 1]$ where 0 is the *stop* action. The expected behavior of the optimal policy is first to localize in the light region, then travel down to the goal. The BetaZero policy exhibits this behavior which can be seen in fig. 10 (where circles indicate the final location).

The approximately optimal solution to the light dark problems used *local approximation value iteration* (LAVI) (Kochenderfer, 2015) over the discretized belief-state space (i.e., mean and std). The belief mean was discretized between the range $[-12, 12]$ and the belief std was discretized between the range $[0, 5]$; each of length 100. The LAVI solver used 100 generative samples per belief state and ran for 100 value iterations with a Bellman residual of 1×10^{-3}.

Rock sample. In the RockSample(n, k) POMDP introduced by Smith & Simmons (2004), an agent has full observability of its position on an $n \times n$ grid but has to sense the k rocks to determine if they are "good" or "bad". The agent knows *a priori* the true locations of the rocks (i.e., the rock locations $\mathbf{x}_{\text{rock}}$ are a part of the problem, not the state). The observation noise is a function of the distance to the rock:

$$\frac{1}{2}\left(1 + \exp\left(-\frac{\|\mathbf{x}_{\text{rock}} - \mathbf{x}_{\text{agent}}\|_2 \log(2)}{c}\right)\right) \tag{9}$$

where $c = 20$ is the sensor efficiency. The agent can move in the four cardinal directions, sense the k rocks, or take the action to *sample* a rock when it is located under the agent. The agent receives a reward of 10 for sampling a "good" rock and a penalty of -10 for sampling a "bad" rock. The terminal state is the exit at the right edge of the map, where the agent gets a reward of 10 for exiting.

Mineral exploration. The Mineral Exploration POMDP introduced by Mern & Caers (2023) is an information gather problem with the goal of deciding whether a subsurface ore body is economical to mine or should be abandoned (calibrated so that 50% of cases are economical). The agent can drill every fifth cell of a 32×32 plot of land to determine the ore quality at that location. Therefore, the action space consists of the 36 drill locations and the final decisions to either *mine* or *abandon*. The agent receives a small cost for each *drill* action, a reward proportional to the extracted ore if chosen to *mine* (which is negative if uneconomical), and a reward of zero if chosen to *abandon*:

$$R(s,a) = \begin{cases} -c_{\text{drill}} & \text{if } a = \text{drill} \\ \sum \mathbb{1}(s_{\text{ore}} \geq h_{\text{massive}}) - c_{\text{extract}} & \text{if } a = \text{mine} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

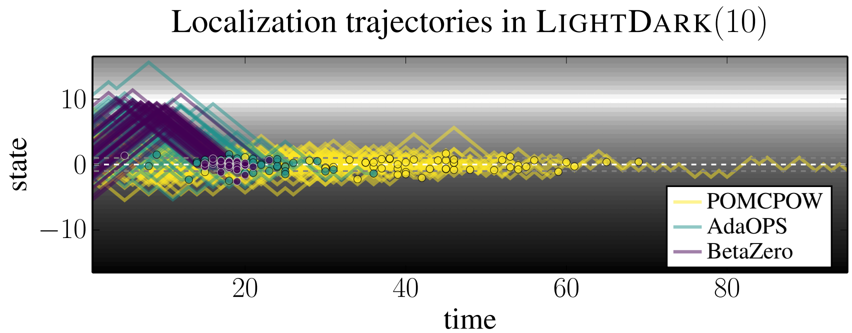

Figure 10: LightDark(10) trajectories from 50 episodes. BetaZero (dark blue) learned to first localize in the light region at $y = 10$ before heading to the goal (origin).

where $c_{\text{drill}} = 0.1$, $h_{\text{massive}} = 0.7$, and $c_{\text{extract}} = 71$. The term $\sum \mathbb{1}(s_{\text{ore}} \geq h_{\text{massive}})$ indicates the cells that have an ore quality value above some massive ore threshold h_{massive} (which are deemed valuable). Figure 11 and fig. 12 show an example of four steps of the mineral exploration POMDP.

A.1 Experiment details

Experiment parameters for each problem can be seen in tables 3 to 5 under the "online" column. For the baseline algorithms, the heuristics follow Wu et al. (2021b). Problems that failed to run due to memory limits followed suggestions from Wu et al. (2021a) to first use the MDP solution and then use a fixed upper bound of $r_{\text{correct}} = 100$ for the light dark problems and the following for the rock sample problems:

$$V_{\max} = r_{\text{exit}} + \sum_{t=1+n-k}^{2k-n} \gamma^{t-1} r_{\text{good}} \tag{11}$$

where $r_{\text{good}} = r_{\text{exit}} = 10$ and the sum computes an optimistic value assuming the rocks are directly lined between the agent and the goal and assuming $n \geq k$ for simplicity.

For problems not studied by Wu et al. (2021b), we use the same heuristics as their easier counterpart (i.e., LightDark(10) uses LightDark(5) heuristics and RockSample(20, 20) uses RockSample(15, 15) heuristics). For mineral exploration, the baselines used the following heuristics. POMCPOW used a value estimator of $\max(0, R(s, a = \text{mine}))$ and when using "no heuristic" used a random rollout policy to estimate the value. Both AdaOPS and DESPOT used a lower bound computed as the returns if fully drilled all locations, then made the decision to abandon:

$$V_{\min} = -\sum_{t=1}^{T-1} \gamma^{t-1} c_{\text{drill}} \tag{12}$$

The upper bound comes from an oracle π_{truth} taking the correct final action without drilling, computed over 10,000 states. Note that there is no state transition in this problem.

$$V_{\max} = \mathop{\mathbb{E}}_{s \in \mathcal{S}} \left[\max\left(0, R\big(s, \pi_{\text{truth}}(s)\big)\right) \right] \tag{13}$$

$$\approx \frac{1}{n} \sum_{i=1}^{n} \max\left(0, R\big(s^{(i)}, \pi_{\text{truth}}(s)\big)\right) \tag{14}$$

Particle filtering. Both BetaZero and the baseline algorithms update their belief with a bootstrap particle filter using a low-variance resampler (Gordon et al., 1993), with $n_{\text{particles}} \in [500, 1000, 1000]$ for the light dark, rock sample, and mineral exploration problems, respectively. The particle filter follows an update procedure of first reweighting then resampling. In mineral exploration, the

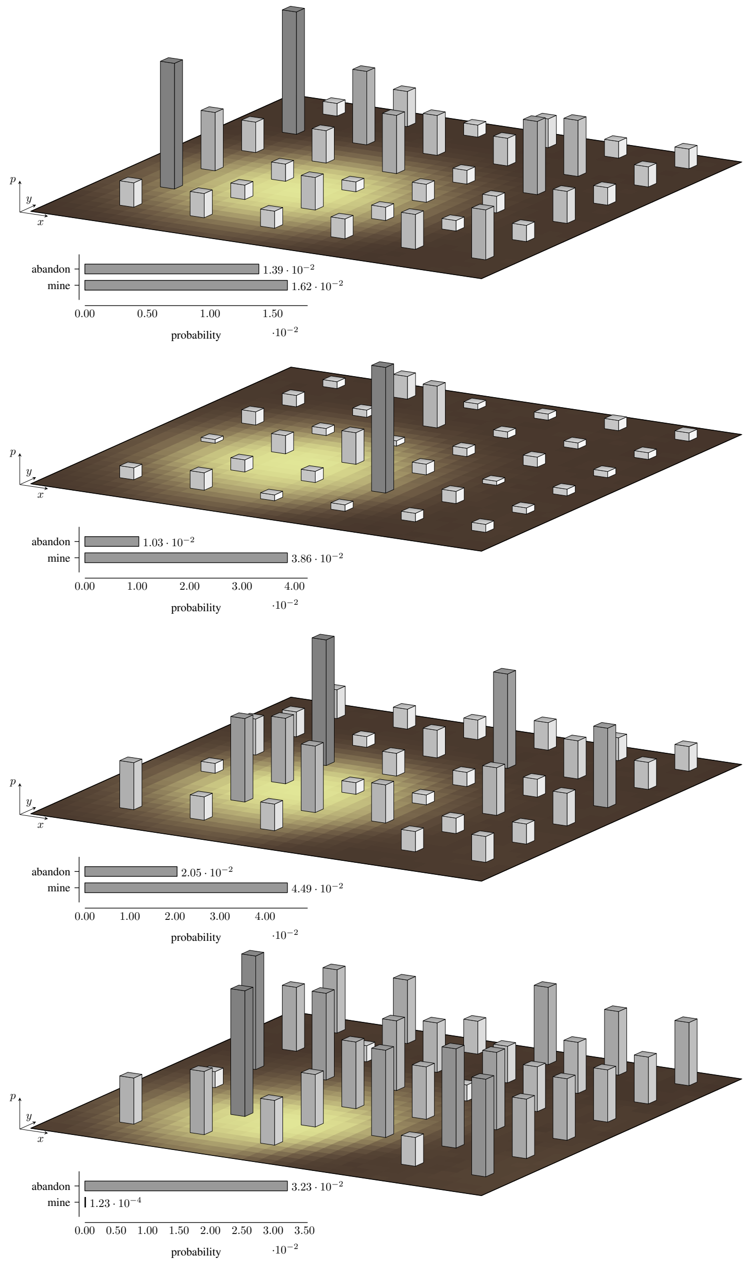

Figure 11: The BetaZero policy shown over belief mean for four steps. BetaZero first prioritizes the edges of the belief mean, corresponding to the belief uncertainty (right-most plots), then explores the outer regions of the subsurface; ultimately gathering information from actions with high mean and std, matching heuristics. At the initial step, abandoning and mining have near-equal probability (bottom left graphs) but by the fourth action, abandoning is much more likely.

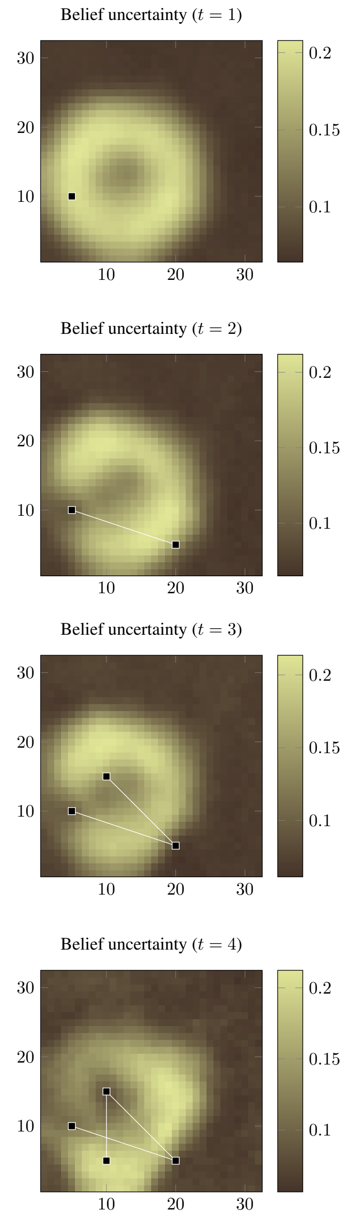

Figure 12: The selected drill actions over belief uncertainty, showing that uncertainty collapses after drilling.

observations are noiseless which could quickly result in particle depletion. Therefore, approximate Bayesian computation (ABC) is used to reweight each particle using a Gaussian distribution centered at the observation with a standard deviation of $\sigma_{\text{abc}} = 0.1$ (Csilléry et al., 2010).

The belief representation takes the mean and standard deviation across the $n_{\text{particles}}$. In the light dark problems, this is computed across the 500 sampled y-state values that make up the belief. The initial y-value state distribution—which makes up the initial belief—follows a Gaussian distribution and thus the parametric representation is a good approximation of the belief.

For the rock sample problem, the belief is represented as the mean and standard deviation of the good rocks from the 1000 sampled states (appending the true position as it is deterministic). The rock qualities are sampled uniformly in $\{0, 1\}$ indicating if they are "good", which makes the problem non-Gaussian, but the parametric belief approximation can model a uniform distribution by placing the mean at the center of the uniform range and stretching the variance to match the uniform.

Lastly, the mineral exploration problem flattens the 1000 subsurface 32×32 maps that each have associated ore quality per-pixel between $[0, 1]$ into two images: a mean and standard deviation image of the ore quality that is stacked and used as input to a CNN. The initial state distribution for the massive ore quantity closely follows a Gaussian, making the parametric belief approximation well suited.

For problems where Gaussian approximations do not capture the belief, the parameters of other distributions could be used as a belief representation or the particles themselves could be input into a network—first passing through an order-invariant layer (Igl et al., 2018). Scaling to larger observation spaces will not be an issue as BetaZero plans over belief states instead of observations.

B Additional Analysis

This section briefly describes additional analyses omitted from the main body of the paper. This includes analysis of bootstrapping the initial Q-values using a one-step lookahead with the value network and sensitivity analysis of double progressive widening on belief-states and actions.

B.1 Bootstrapping analysis

When adding a belief-action pair (b, a) to the MCTS tree, initializing the Q-values via bootstrapping with the value network may improve performance when using a small MCTS budget. Table 2 shows the results of an analysis comparing BetaZero with bootstrapping $Q_0(b, a) = R_b(b, a) + \gamma V_\theta(\tilde{b}')$ where $\tilde{b}' = \phi(b')$ and without bootstrapping $Q_0(b, a) = 0$. Each domain used the online parameters described in tables 3 to 5. Results indicate that bootstrapping was only helpful in the rock sample problems and incurs additional compute time due to the belief update done in $b' \sim T_b(b, a)$. Note that bootstrapping was not used during offline training. In problems with high stochasticity in the belief-state transitions, bootstrapping may be noisy during the initial search due to the transition T_b sampling a single state from the belief. Further analysis could investigate the use of multiple belief transitions to better estimate the value, at the expense of additional computation. The value estimate of b could instead be used as the bootstrap but we would expect similar results to the one-step bootstrap as many problems we study have sparse rewards.

	LightDark(5)		LightDark(10)		RockSample(15, 15)		RockSample(20, 20)		Mineral Exploration	
	returns	time [s]	returns	time [s]	returns	time [s]	returns	time [s]	returns	time [s]
BetaZero (bootstrap)	4.22 ± 0.31	0.014	14.45 ± 1.15	0.34	$\mathbf{20.15 \pm 0.71}$	0.48	$\mathbf{13.09 \pm 0.55}$	1.11	10.32 ± 2.38	6.27
BetaZero (no bootstrap)	$\mathbf{4.47 \pm 0.28}$	0.014	$\mathbf{16.77 \pm 1.28}$	0.33	19.50 ± 0.71	0.42	11.00 ± 0.54	0.57	$\mathbf{10.67 \pm 2.25}$	4.46

Reporting mean ± standard error over 100 seeds (i.e., episodes); timing is average per episode.

Table 2: Effect of Q-value bootstrapping in online *BetaZero* performance (returns and online timing).

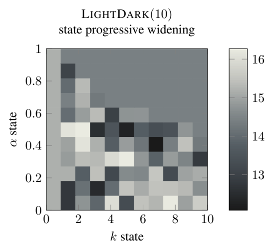

Figure 13: Sensitivity analysis of *belief-state* progressive widening in LightDark(10).

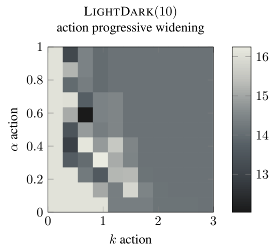

Figure 14: Sensitivity analysis of *action* progressive widening in LightDark(10).

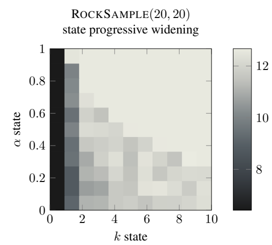

Figure 15: Sensitivity analysis of *belief-state* progressive widening in RockSample(20, 20).

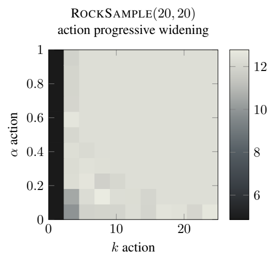

Figure 16: Sensitivity analysis of *action* progressive widening in RockSample(20, 20).

B.2 Limitations of double progressive widening

Double progressive widening (DPW) is a straightforward approach to handle large or continuous state and action spaces in Monte Carlo tree search. It is easy to implement and only requires information available in the tree search, i.e., number of children nodes and number of node visits. It is known that MCTS performance can be sensitive to DPW hyperparameter tuning and Sokota et al. (2021) show that DPW ignores information about the relation between states that could provide more intelligent branching. Sokota et al. (2021) introduce *state abstraction refinement* that uses a distance metric between states to determine if a similar state should be added to the tree; requiring a state transition every time a state-action node is visited. For our work, we want to reduce the number of expensive belief-state transitions in the tree and avoid the use of problem-specific heuristics required when defining distance metrics. Using DPW in BetaZero was motivated by simplicity and allows future work to innovate on the components of belief-state and action branching.

To analyze the sensitivity of DPW, figs. 13 and 14 show a sweep over the α and k parameters for DPW in LightDark(10). Figure 13 shows that the light dark problem is sensitive to belief-state widening and fig. 14 indicates that this problem may not require widening on all actions—noting that when $k = 0$, the only action expanded on is the one prioritized from the policy head $a \sim P_\theta(\tilde{b}, \cdot)$. The light dark problems have a small action space of $|\mathcal{A}| = 3$, therefore this prioritization leads to good performance when only a single action is evaluated (left cells in fig. 14 when $k = 0$).

In RockSample(20, 20), figs. 15 and 16 indicates that this problem benefits from a higher widening factor (top right of the figures) as the action space $|\mathcal{A}| = 25$ is larger and the belief-state transitions operate over a much larger state space. DPW uses a single branching factor throughout the tree search and research into methods that adapt the branching based on learned information would be a valuable direction to explore.

Lim et al. (2023) introduce a class of POMDP planning algorithms that use a fixed number of samples to branch on instead of progressive widening. The bottom row of figs. 13 to 16 (where $\alpha = 0$) can be interpreted as a fixed branching factor compared to progressive widening in the other cells. The analysis in the figures shows that there are cases where BetaZero has better performance when using progressive widening (show in the lighter colors).

C Network Architectures

Figures 17 to 19 specify the neural network architectures for the three problem domains. The networks were designed to be simple so that future work could focus on incorporating more complicated architectures such as residual networks. Mineral exploration does not normalize the inputs and is the only problem where the input is treated as an image, thus we use a convolutional neural network (CNN). Training occurs on normalized returns and an output denormalization layer is added to the value head to ensure proper magnitude of the predicted values.

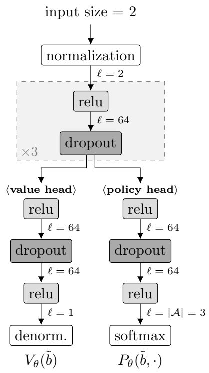

Figure 17: Light dark neural network architecture.

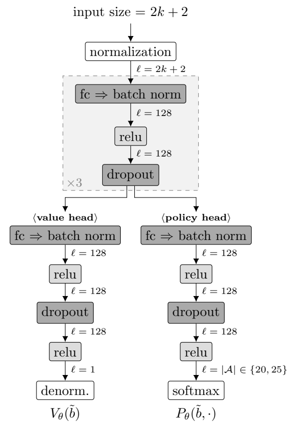

Figure 18: Rock sample neural network architecture.

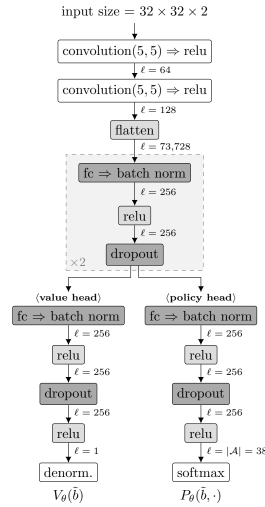

Figure 19: Mineral exploration CNN architecture.

C.1 Return scaling for output normalization

For general POMDPs, the return can be an unbounded real-value and not conveniently in $[0, 1]$ or $[-1, 1]$; as is often the case with two player games. Schrittwieser et al. (2020) use a categorical representation of the value split into a discrete support to make learning more robust (Schrittwieser, 2020). We instead simply normalize the target before training as

$$\bar{g}_t = \frac{g_t - \mathbb{E}[G_{\text{train}}]}{\sqrt{\text{Var}[G_{\text{train}}]}} \tag{15}$$

where G_{train} is the set of returns used during training; keeping running statistics of all training data. Intuitively, this ensures that the target values have zero mean and unit variance which is known to stabilize training (LeCun et al., 2002). After training, a denormalization layer is added to the normalized output $\bar{v}$ of the value network as

$$v_t = \bar{v}\sqrt{\text{Var}[G_{\text{train}}]} + \mathbb{E}[G_{\text{train}}] \tag{16}$$

to properly scale value predictions when the network is evaluated (which is done entirely internal to the network).

D Hyperparameters and Tuning

The hyperparameters used during offline training and online execution are described in tables 3 to 5. Offline training refers to the BetaZero policy iteration steps that collect parallel MCTS data (*policy evaluation*) and then retrain the network (*policy improvement*). The online execution refers to using the BetaZero policy after offline training to evaluate its performance through online tree search. The main difference between these two settings is the final criteria used to select the root node action in MCTS. During offline training of problems with large action spaces (e.g., rock sample and mineral exploration), sampling root node actions according to the Q-weighted visit counts with a temperature τ ensures exploration. To evaluate the performance online, root node action selection takes the maximizing action of the Q-weighted visit counts. During training, we also evaluate a holdout set that uses the argmax criteria to monitor the true performance of the learned policy.

The MCTS parameters for the mineral exploration problem were tuned using Latin hypercube sampling based on the lower-confidence bound of the returns. During training, the rock sample

	Parameter*	LightDark(5)		LightDark(10)		Description
		Offline	Online	Offline	Online	
BetaZero policy iteration parameters	$n_{\text{iterations}}$	30	—	30	—	Number of offline BetaZero policy iterations.
	n_{data}	500	—	500	—	Number of parallel MCTS data gen. episodes per policy iteration.
	bootstrap Q_0	false	false	false	false	Use bootstrap estimate for initial Q-value in MCTS.
Neural network parameters	n_{epochs}	50	—	50	—	Number of training epochs.
	α	1×10^{-4}	—	1×10^{-4}	—	Learning rate.
	λ	1×10^{-5}	—	1×10^{-5}	—	L_2-regularization parameter.
MCTS parameters	n_{online}	100	1300	100	1000	Number of tree search iterations of MCTS.
	c	1	1	1	1	PUCT exploration constant.
	k_a	2.0	2.0	2.0	2.0	Multiplicative action progressive widening value.
	α_a	0.25	0.25	0.25	0.25	Exponential action progressive widening value.
	k_b	2.0	2.0	2.0	2.0	Multiplicative belief-state progressive widening value.
	α_b	0.1	0.1	0.1	0.1	Exponential belief-state progressive widening value.
	d	10	10	10	10	Maximum tree depth.
	τ	0	0	0	0	Exploration temperature for final root node action selection.
	z_q	1	1	1	1	Influence of Q-values in final criteria.
	z_n	1	1	1	1	Influence of visit counts in final criteria.

* Entries with "—" denote non-applicability and "·" denotes they are disabled.

Table 3: *BetaZero* parameters for the LightDark problems.

	Parameter	RockSample(15, 15)		RockSample(20, 20)		Description
		Offline	Online	Offline	Online	
BetaZero policy iteration parameters	$n_{\text{iterations}}$	50	—	50	—	Number of offline BetaZero policy iterations.
	n_{data}	500	—	500	—	Number of parallel MCTS data gen. episodes per policy iteration.
	bootstrap Q_0	false	true	false	true	Use bootstrap estimate for initial Q-value in MCTS.
Neural network parameters	n_{epochs}	10	—	10	—	Number of training epochs.
	α	1×10^{-3}	—	1×10^{-3}	—	Learning rate.
	λ	1×10^{-5}	—	1×10^{-5}	—	L_2-regularization parameter.
MCTS parameters	n_{online}	100	100	100	100	Number of tree search iterations of MCTS.
	c	50	50	50	50	PUCT exploration constant.
	k_a	·	5.0	·	·	Multiplicative action progressive widening value.
	α_a	·	0.9	·	·	Exponential action progressive widening value.
	k_b	·	1.0	1.0	1.0	Multiplicative belief-state progressive widening value.
	α_b	·	0.0	0.0	0.0	Exponential belief-state progressive widening value.
	d	15	15	4	4	Maximum tree depth.
	τ	1.0	0	1.5	0	Exploration temperature for final root node action selection.
	z_q	1	0.4	1	0.5	Influence of Q-values in final criteria.
	z_n	1	0.9	1	0.8	Influence of visit counts in final criteria.

Table 4: *BetaZero* parameters for the RockSample problems.

	Parameter	Offline	Online	Description
BetaZero policy iteration parameters	$n_{\text{iterations}}$	20	—	Number of offline BetaZero policy iterations.
	n_{data}	100	—	Number of parallel MCTS data gen. episodes per policy iteration.
	bootstrap Q_0	false	false	Use bootstrap estimate for initial Q-value in MCTS.
Neural network parameters	n_{epochs}	10	—	Number of training epochs.
	α	1×10^{-6}	—	Learning rate.
	λ	1×10^{-4}	—	L_2-regularization parameter.
MCTS parameters	n_{online}	50	50	Number of tree search iterations of MCTS.
	c	57	57	PUCT exploration constant.
	k_a	41.09	41.09	Multiplicative action progressive widening value.
	α_a	0.57	0.57	Exponential action progressive widening value.
	k_b	37.13	37.13	Multiplicative belief-state progressive widening value.
	α_b	0.94	0.94	Exponential belief-state progressive widening value.
	d	5	5	Maximum tree depth.
	τ	1.0	0	Exploration temperature for final root node action selection.
	z_q	1	1	Influence of Q-values in final criteria.
	z_n	1	1	Influence of visit counts in final criteria.

Table 5: *BetaZero* parameters for the Mineral Exploration problem.

problems disabled progressive widening to expand on all actions and transition to a single belief state. Then for online execution, we tuned the DPW parameters as shown in figs. 13 to 16. The problems train with a batch size of 1024 over 80% of 100,000 samples from one round of data collection ($n_{\text{buffer}} = 1$) using p_{dropout} of 0.2, 0.5, 0.7, respectively. The neural network optimizer Adam (Kingma & Ba, 2014) was used in LIGHTDARK(y) while RMSProp (Hinton et al., 2014) was used in the others. A value function loss of MAE was used in mineral exploration (MSE otherwise), each using $n_{\text{samples}} = 100{,}000$ during training.

E Compute Resources

BetaZero was designed to use a single GPU to train the network and parallelize MCTS evaluations across available CPUs. Evaluating the networks on the CPU is computationally inexpensive due to the size of the networks (see figs. 17 to 19). This design was chosen to enable future research without a computational bottleneck. For network training, a single NVIDIA A100 was used with 80GB of memory on an Ubuntu 22.04 machine with 500 GB of RAM. Parallel data collection processes were run on 50 processes split evenly over two separate Ubuntu 22.04 machines: (1) with 40 Intel Xeon 2.3 GHz CPUs, and (2) with 56 Intel Xeon 2.6 GHz CPUs. Algorithm 5 (line 3) shows where CPU parallelization occurs. In practice, the MCTS data generation simulations are the bottleneck of the offline component of BetaZero and not the network training—thus, parallelization is useful.

F Open-Sourced Code and Experiments

The BetaZero algorithm has been open sourced and incorporated into the Julia programming language POMDPs.jl ecosystem (Egorov et al., 2017). Fitting into this ecosystem allows BetaZero to access existing POMDP models and can easily be compared to various POMDP solvers. The user constructs a `BetaZeroSolver` that takes parameters for policy iteration and data generation, parameters for neural network architecture and training, and parameters for MCTS (described in the tables above). The user may choose to define a method that inputs the belief b and outputs the belief representation $\tilde{b}$ used by the neural network (the default computes the belief mean and std). Given a `pomdp::POMDP` structure, a `solver::BetaZeroSolver` is constructed and solved using:

```
policy = solve(solver, pomdp)
```

which runs *offline* policy iteration (algorithm 4). Once you have a trained neural network, an action can then be generated *online* from the policy given a belief b using the following (algorithm 6):

```
a = action(policy, b)
```

All experiments, including the experiment setup for the baseline algorithms with their heuristics, are included for reproducibility. Code to run MCTS data collection across parallel processes is also included. The code and experiments presented in this work are available online.[3]

G BetaZero Algorithm

The following algorithms 4 to 6 detail the full BetaZero policy iteration algorithm that iterates between *policy evaluation* and *policy improvement* for a total of $n_{\text{iterations}}$. The offline policy evaluation stage, or data collection process (algorithm 5), runs n_{data} parallel MCTS simulations over the original POMDP and collects a dataset $\mathcal{D}$ of beliefs b_t, policy vectors $\boldsymbol{\pi}_t$, and returns g_t (computed after each episode terminates). The top-level Q-weighted MCTS algorithm is shown in algorithm 6, which iteratively runs MCTS simulations for n_{online} iterations to a specified depth d. The final root node action selection policy follows the Q-weighted visit counts from eq. (6). The descriptions of parameters ψ used in offline training and online tree search are listed in tables 3 to 5.

[3] `https://github.com/sisl/BetaZero.jl`

Algorithm 4: BetaZero offline policy iteration.

Require: $\mathcal{P} \stackrel{\text{def}}{=} \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$: POMDP
Require: ψ: Parameters (includes $n_{\text{iterations}}$, n_{data}, n_{online}, and d)

```
function BetaZero(𝒫, ψ)
    f_θ ← InitializeNetwork(ψ)
    P_θ, V_θ ← f_θ                                ▷ where (p, v) ← (P_θ(b̃), V_θ(b̃))
    for i ← 1 to n_iterations
        𝒟 ← CollectData(𝒫, f_θ, ψ)                ▷ policy evaluation
        f_θ ← Train(f_θ, 𝒟)                       ▷ policy improvement
    return β_0^π(𝒫, f_θ)                          ▷ BetaZero online policy (uses alg. 6)
```

Algorithm 5: Collect MCTS data offline for policy evaluation.

```
function CollectData(𝒫, f_θ, ψ)
    𝒟 = ∅
    parallel for i ← 1 to n_data                  ▷ parallelize MCTS runs across available CPUs
        for t ← 1 to T
            a_t ← MonteCarloTreeSearch(𝒫, f_θ, b_t, ψ)    ▷ select next action through online planning
            𝒟_i^(t) ← 𝒟_i^(t) ∪ {(b_t, π_tree^(t), g_t)}  ▷ collect belief and policy data (placeholder for returns)
            s_{t+1} ~ T(· | s_t, a_t)              ⎫
            o_t ~ O(· | a_t, s_{t+1})              ⎬ transition the original POMDP
            b_{t+1} ← Update(b_t, a_t, o_t)        ⎪
            r_t ← R(s_t, a_t) or R(s_t, a_t, s_{t+1})  ⎭
        g_t ← Σ_{k=t}^T γ^(k−t) r_k   for  t ← 1 to T   ▷ compute returns from observed rewards
    return 𝒟
```

Algorithm 6: Monte Carlo tree search algorithm using Q-weighed visit counts.

```
function MonteCarloTreeSearch(𝒫, f_θ, b, ψ)
    ℳ ← ⟨ℬ, 𝒜, T_b, R_b, γ⟩ converted from the POMDP 𝒫    ▷ plan using the belief-state MDP
    for i ← 1 to n_online
        Simulate(f_θ, b, d)                   ▷ MCTS simulated planning to a depth d (algorithm 3)
    π_tree(b, a) ∝ (( exp Q(b,a) / Σ_a' exp Q(b,a') )^z_q ( N(b,a) / Σ_a' N(b,a') )^z_n )^(1/τ)   ▷ Q-weighted visit counts eq. (6)
    π_tree(b, a_i) ← π_tree(b, a_i) / Σ_j π_tree(b, a_j)    ▷ normalize to get a valid probability distribution
    return a ~ π_tree(b, ·)                   ▷ sample root node action (let τ → 0 to get argmax)
```