Memory Bounded Open-Loop Planning in Large POMDPs using Thompson Sampling

Thomy Phan
LMU Munich
thomy.phan@ifi.lmu.de

Lenz Belzner
MaibornWolff
lenz.belzner@maibornwolff.de

Marie Kiermeier
LMU Munich
marie.kiermeier@ifi.lmu.de

Markus Friedrich
LMU Munich
markus.friedrich@ifi.lmu.de

Kyrill Schmid
LMU Munich
kyrill.schmid@ifi.lmu.de

Claudia Linnhoff-Popien
LMU Munich
linnhoff@ifi.lmu.de

Abstract

State-of-the-art approaches to partially observable planning like POMCP are based on stochastic tree search. While these approaches are computationally efficient, they may still construct search trees of considerable size, which could limit the performance due to restricted memory resources. In this paper, we propose *Partially Observable Stacked Thompson Sampling (POSTS)*, a memory bounded approach to open-loop planning in large POMDPs, which optimizes a fixed size stack of Thompson Sampling bandits. We empirically evaluate POSTS in four large benchmark problems and compare its performance with different tree-based approaches. We show that POSTS achieves competitive performance compared to tree-based open-loop planning and offers a performance-memory tradeoff, making it suitable for partially observable planning with highly restricted computational and memory resources.

Introduction

Many real-world problems can be modeled as *Partially Observable Markov Decision Process (POMDP)*, where the true state is unknown to the agent due to limited and noisy sensors. The agent has to reason about the *history* of past observations and actions, and maintain a *belief state* as a distribution of possible states. POMDPs have been widely used to model decision making problems in the context of planning and reinforcement learning (Ross et al. 2008).

Solving POMDPs exactly is computationally intractable for domains with enormous state spaces and long planning horizons. First, the space of possible belief states grows exponentially w.r.t. the number of states N, since that space is N-dimensional, which is known as *curse of dimensionality* (Kaelbling, Littman, and Cassandra 1998). Second, the number of possible histories grows exponentially w.r.t. the horizon length, which is known as the *curse of history* (Pineau, Gordon, and Thrun 2006).

In the last few years, *Monte-Carlo planning* has been proposed to break both curses with statistical sampling. These methods construct sparse trees over belief states and actions, representing the state-of-the-art for efficient planning in large POMDPs (Silver and Veness 2010; Somani et al. 2013; Bai et al. 2014). While these approaches avoid exhaustive search, the constructed *closed-loop* trees can still become arbitrarily large for highly complex domains, which could limit the performance due to restricted memory resources (Powley, Cowling, and Whitehouse 2017). In contrast, *open-loop* approaches only focus on searching action sequences and are independent of the history and belief state space. Open-loop approaches are able to achieve competitive performance compared to closed-loop planning, when the problem is too large to provide sufficient computational and memory resources (Weinstein and Littman 2013; Perez Liebana et al. 2015; Lecarpentier et al. 2018). However, open-loop planning has been a less popular choice for decision making in POMDPs so far (Yu et al. 2005).

In this paper, we propose *Partially Observable Stacked Thompson Sampling (POSTS)*, a memory bounded approach to open-loop planning in large POMDPs, which optimizes a fixed size stack of Thompson Sampling bandits.

To evaluate the effectiveness of POSTS, we formulate a tree-based approach, called *Partially Observable Open-Loop Thompson Sampling (POOLTS)* and show that POOLTS is able to find optimal open-loop plans with sufficient computational and memory resources.

We empirically test POSTS in four large benchmark problems and compare its performance with POOLTS and other tree-based approaches like POMCP. We show that POSTS achieves competitive performance compared to tree-based open-loop planning and offers a performance-memory tradeoff, making it suitable for partially observable planning with highly restricted computational and memory resources.

Background

Partially Observable Markov Decision Processes

A POMDP is defined by a tuple $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{O}, \Omega, b_0 \rangle$, where $\mathcal{S}$ is a (finite) set of states, $\mathcal{A}$ is the (finite) set of actions, $\mathcal{P}(s_{t+1}|s_t, a_t)$ is the transition probability function, $\mathcal{R}(s_t, a_t)$ is the scalar reward function, $\mathcal{O}$ is a (finite) set of observations, $\Omega(o_{t+1}|s_{t+1}, a_t)$ is the observation probability function, and b_0 is a probability distribution over initial states $s_0 \in \mathcal{S}$. It is always assumed, that $s_t, s_{t+1} \in \mathcal{S}$, $a_t \in \mathcal{A}$, and $o_t, o_{t+1} \in \mathcal{O}$ at time step t.

A *history* $h_t = [a_0, o_1, ..., a_{t-1}, o_t]$ is a sequence of actions and observations. A *belief state* $b_{h_t}(s_t)$ is a sufficient statistic for history h_t and defines a probability distribution over states s_t given h_t. $\mathcal{B}$ is the space of all possible belief states. b_0 represents the *initial belief state* b_{h_0}. The belief state can be updated by Bayes theorem:

$$b_{h_t}(s_t) = \eta \Omega(o_t|s_t, a) \sum_{s \in \mathcal{S}} \mathcal{P}(s_t|s, a) b_h(s_t) \tag{1}$$

where $\eta = \frac{1}{\Omega(o_{t+1}|b_h, a)}$ is a normalizing constant, $a = a_{t-1}$ is the last action, and h is the history without a and o_t.

The goal is to find a *policy* $\pi : \mathcal{B} \rightarrow \mathcal{A}$, which maximizes the return G_t at state s_t for a horizon T:

$$G_t = \sum_{k=0}^{T-1} \gamma^k \cdot \mathcal{R}(s_{t+k}, a_{t+k}) \tag{2}$$

where $\gamma \in [0, 1]$ is the discount factor. If $\gamma < 1$, then present rewards are weighted more than future rewards.

The *value function* $V^{\pi}(b_t) = \mathbb{E}_{\pi}[G_t|b_t]$ is the expected return conditioned on belief states given a policy π. An optimal policy π^* has a value function $V^{\pi^*} = V^*$ with $V^*(b_t) \geq V^{\pi'}(b_t)$ for all $b_t \in \mathcal{B}$ and $\pi' \neq \pi^*$.

Multi-armed Bandits

Multi-armed Bandits (MABs or bandits) are fundamental decision making problems, where an agent has to repeatedly select an arm among a given set of arms in order to maximize its future pay-off. MABs can be considered as problems with a single state s, a set of actions $a \in \mathcal{A}$, and a stochastic reward function $\mathcal{R}(s, a) := X_a$, where X_a is a random variable with an unknown distribution $f_{X_a}(x)$. To solve a MAB, one has to determine the action, which maximizes the expected reward $\mathbb{E}[X_a]$. The agent has to balance between sufficiently trying out actions to accurately estimate their expected reward and to exploit its current knowledge on all arms by selecting the arm with the currently highest expected reward. This is known as the *exploration-exploitation dilemma*, where exploration can lead to actions with possibly higher rewards but requires time for trying them out, while exploitation can lead to fast convergence but possibly gets stuck in a local optimum. In this paper, we will cover UCB1 and Thompson Sampling as MAB algorithms.

UCB1 In *UCB1*, actions are selected by maximizing the upper confidence bound of action values $UCB1(a) = \overline{X_a} + c\sqrt{\frac{log(N_{total})}{N_a}}$, where $\overline{X_a}$ is the current average reward when choosing a, c is an exploration constant, N_{total} is the total number of action selections, and N_a is the number of times action a was selected. The second term represents the exploration bonus, which becomes smaller with increasing N_a (Auer, Cesa-Bianchi, and Fischer 2002).

UCB1 is a popular MAB algorithm and widely used in various challenging domains (Kocsis and Szepesvári 2006; Bubeck and Munos 2010; Silver et al. 2016; 2017).

Thompson Sampling *Thompson Sampling* is a Bayesian approach to balance between exploration and exploitation of actions (Thompson 1933). The unknown reward distribution of X_a of each action $a \in \mathcal{A}$ is modeled by a parametrized likelihood function $P_a(x|\theta)$ with a parameter vector θ. Given a prior distribution $P_a(\theta)$ and a set of past observed rewards $D_a = \{x_{a,1}, x_{a,2}, ..., x_{a,N_a}\}$, the posterior distribution $P_a(\theta|D_a)$ can be inferred by using Bayes rule $P_a(\theta|D_a) \propto \prod_i P_a(x_{a,i}|\theta)P_a(\theta)$. The expected reward of each action $a \in \mathcal{A}$ can be estimated by sampling $\theta \sim P_a(\theta|D_a)$ from the posterior. The action with the highest sampled expected reward $\mathbb{E}_{\theta}[X_a]$ is selected.

Thompson Sampling has been shown to be an effective and robust algorithm for making decisions under uncertainty (Chapelle and Li 2011; Kaufmann, Korda, and Munos 2012; Bai, Wu, and Chen 2013; Bai et al. 2014).

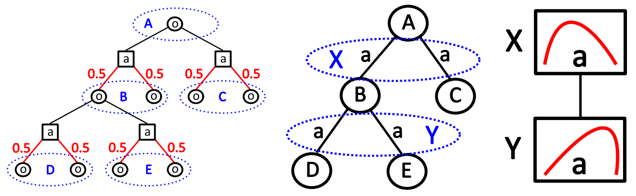
Planning in POMDPs

Planning searches for an (near-)optimal policy given a model $\hat{M}$ of the environment M, which usually consists of explicit probability distributions of the POMDP. Unlike *offline planning*, which searches the whole (belief) state space to find the optimal policy π^*, *local planning* only focuses on finding a policy π_t for the current (belief) state by taking possible future (belief) states into account (Weinstein and Littman 2013). Thus, local planning can be applied *online* at every time step at the current state to recommend the next action for execution. Local planning is usually restricted to a time or computation budget n_b due to strict real-time constraints (Bubeck and Munos 2010; Weinstein and Littman 2013; Perez Liebana et al. 2015).

In this paper, we focus on local *Monte-Carlo planning*, where $\hat{M}$ is a generative model, which can be used as black box simulator (Kocsis and Szepesvári 2006; Silver and Veness 2010; Weinstein and Littman 2013; Bai et al. 2014). Given s_t and a_t, the simulator $\hat{M}$ provides a sample $\langle s_{t+1}, o_{t+1}, r_t \rangle \sim \hat{M}(s_t, a_t)$. Monte-Carlo planning algorithms can approximate π^* and V^* by iteratively simulating and evaluating action sequences without reasoning about explicit probability distributions of the POMDP.

Local planning can be closed- or open-loop. *Closed-loop planning* conditions the action selection on histories of actions and observations. *Open-loop planning* only conditions the action selection on previous sequences of actions $p_T = [a_1, ..., a_T]$ (also called *open-loop plans* or simply *plans*) and summarized statistics about predecessor (belief) states (Bubeck and Munos 2010; Weinstein and Littman 2013; Perez Liebana et al. 2015). An example is shown in Fig. 1. A closed-loop tree for a domain with $\Omega(s_{t+1}|s_t, a_t) = 0.5$ is shown in Fig. 1a, while Fig. 1b shows the corresponding open-loop tree which summarizes the observation nodes of Fig. 1a within the blue dotted ellipses into history distribution nodes. Open-loop planning can be further simplified by only regarding statistics about the expected return of actions at specific time steps (Fig. 1c). In that case, only a stack of T statistics is used to sample plans for simulation and evaluation (Weinstein and Littman 2013).

Partially Observable Monte-Carlo Planning (POMCP) is



(a) closed-loop tree (b) open-loop tree (c) stacked

Figure 1: Illustration of closed- and open-loop planning schemes. (a) Closed-loop tree with state observations (circular nodes) and actions (rectangular nodes). Red links correspond to stochastic observations made with a probability of 0.5. (b) Open-loop tree with links as actions and history distribution nodes according to the blue dotted ellipses in Fig. 1a. (c) Open-loop approach with stack of action distributions according to the blue dotted ellipses in Fig. 1b.

a closed-loop approach based on *Monte-Carlo Tree Search (MCTS)* (Silver and Veness 2010). POMCP uses a search tree of histories with *o-nodes* representing observations and *a-nodes* representing actions (Fig. 1a). Each *o-node* has a visit count $N(h_t)$ and a value estimate $V(h_t) = V(b_t)$ for history h_t and belief state b_t. Each *a-node* has a visit count $N(h_t)$ and a value estimate $Q(h_t, a_t) = \mathbb{E}[G_t|h_t, a_t]$ for action a_t and history h_t. A simulation starts at the current belief state and is divided into two stages: In the first stage, a tree policy π_{tree} is used to traverse the tree until a leaf node is reached. Actions are selected via $\pi_{tree}(h_t)$ and simulated in $\hat{M}$ to determine the next nodes to visit. π_{tree} can be implemented with MABs, where each *o-node* represents a MAB. In the second stage, a rollout policy $\pi_{rollout}$ is used to sample action sequences until a terminal state or a maximum search depth T is reached. The observed rewards are accumulated to returns G_t (Eq. 2), propagated back to update the value estimate of every node in the simulated path, and a new leaf node is added to the search tree. $\pi_{rollout}$ can be used to integrate domain knowledge into the planning process to focus the search on promising states (Silver and Veness 2010). The original version of POMCP uses UCB1 for π_{tree} and is shown to converge to the optimal best-first tree with sufficient computation (Silver and Veness 2010).

(Lecarpentier et al. 2018) formulates an open-loop variant of MCTS using UCB1 as π_{tree}, called *Open-Loop Upper Confidence bound for Trees (OLUCT)*, which could be easily extended to POMDPs by constructing a tree, which summarizes all *o-nodes* to history distribution nodes (Fig. 1b).

Open-loop planning generally converges to sub-optimal solutions in stochastic domains, since it ignores (belief) state values $V(b_t)$ and optimizes the node values $V(N_t)$ (Fig. 1b) instead (Lecarpentier et al. 2018). If the problem is too complex to provide sufficient computation budget n_b or memory capacity, then open-loop approaches are competitive to closed-loop approaches, since they need to explore a much smaller search space to converge to an appropriate solution (Weinstein and Littman 2013; Lecarpentier et al. 2018).

Related Work

Tree-based approaches to open-loop planning condition the action selection on previous action sequences as shown in Fig. 1b (Bubeck and Munos 2010; Perez Liebana et al. 2015; Lecarpentier et al. 2018). Such approaches have been thoroughly evaluated for fully observable problems, but have been less popular for partially observable problems so far (Yu et al. 2005). POSTS is based on stacked open-loop planning, where a stack of T distributions over actions is maintained to generate open-loop plans with high expected return (Weinstein and Littman 2013; Belzner and Gabor 2017). Unlike previous approaches, POSTS is a *memory-bounded* open-loop approach to *partially observable planning*.

(Yu et al. 2005) proposed an open-loop approach to planning in POMDPs by using hierarchical planning. An open-loop plan is constructed at an abstract level, where uncertainty w.r.t. particular actions is ignored. A low-level planner controls the actual execution by explicitly dealing with uncertainty. POSTS is more general, since it performs planning directly on the *original problem* and does not require the POMDP to be transformed for hierarchical planning.

(Powley, Cowling, and Whitehouse 2017) proposed a memory bounded version of MCTS with a state pool to add, discard, or reuse states depending on their visitation frequency. However, this approach cannot be easily adapted to tree-based open-loop approaches, because it requires (belief) states to be identifiable. POSTS does not require a pool to reuse states or nodes, but only maintains a *fixed size stack* of Thompson Sampling bandits, which adapt according to the *temporal dependencies* between actions.

Open-Loop Search with Thompson Sampling

Generalized Thompson Sampling We use a variant of Thompson Sampling, which works for arbitrary reward distributions as proposed in (Bai, Wu, and Chen 2013; Bai et al. 2014) by assuming that X_{a_t} follows a Normal distribution $\mathcal{N}(\mu, \frac{1}{\tau})$ with unknown mean μ and precision $\tau = \frac{1}{\sigma^2}$, where σ^2 is the variance. $\langle \mu, \tau \rangle$ follows a Normal Gamma distribution $\mathcal{NG}(\mu_0, \lambda, \alpha, \beta)$ with $\lambda > 0$, $\alpha \geq 1$, and $\beta \geq 0$. The distribution over τ is a Gamma distribution $\tau \sim \mathit{Gamma}(\alpha, \beta)$ and the conditional distribution over μ given τ is a Normal distribution $\mu \sim \mathcal{N}(\mu_0, \frac{1}{\lambda\tau})$.

Given a prior distribution $P(\theta) = \mathcal{NG}(\mu_0, \lambda_0, \alpha_0, \beta_0)$ and n observations $D = \{x_1, ..., x_n\}$, the posterior distribution is defined by $P(\theta|D) = \mathcal{NG}(\mu_1, \lambda_1, \alpha_1, \beta_1)$, where $\mu_1 = \frac{\lambda_0\mu_0 + n\overline{X}}{\lambda_0 + n}$, $\lambda_1 = \lambda_0 + n$, $\alpha_1 = \alpha_0 + \frac{n}{2}$, and $\beta_1 = \beta_0 + \frac{1}{2}(n\sigma^2 + \frac{\lambda_0 n(\overline{X} - \mu_0)^2}{\lambda_0 + n})$. $\overline{X}$ is the mean of all values in D and $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{X})^2$ is the variance.

The posterior is inferred for each action $a_t \in \mathcal{A}$ to sample an estimate μ_{a_t} for the expected return. The action with the highest estimate is selected. The complete formulation is given in Algorithm 1.

The prior should ideally reflect knowledge about the underlying model, especially for initial turns, where only a small amount of data has been observed (Honda and Takemura 2014). If no knowledge is available, then *uninformative priors* should be chosen, where all possibilities can be

Algorithm 1 Generalized Thompson Sampling

```
procedure $ThompsonSampling(N_t)$
    for $a_t \in \mathcal{A}$ do
        Infer $\langle \mu_1, \lambda_1, \alpha_1, \beta_1 \rangle$ from prior and $\overline{X_{a_t}}, \sigma_a^2, n_{a_t}$
        $\mu_{a_t}, \tau_{a_t} \sim \mathcal{NG}(\mu_1, \lambda_1, \alpha_1, \beta_1)$
    return $argmax_{a_t \in \mathcal{A}}(\mu_{a_t})$

procedure $UpdateBandit(N_t, G_t)$
    $n_{a_t} \leftarrow n_{a_t} + 1$
    $\langle \overline{X_{old,a_t}}, \overline{X_{a_t}} \rangle \leftarrow \langle \overline{X_{a_t}}, (n_{a_t}\overline{X_{old,a_t}} + G_t)/(n_{a_t}+1) \rangle$
    $s_{a_t} \leftarrow [(n_{a_t}-1)s_{a_t} + (G_t - \overline{X_{old,a_t}})(G_t - \overline{X_{a_t}})]/n_{a_t}$
```

sampled (almost) uniformly. This can be achieved by choosing the priors such that the variance of the resulting Normal distribution $\mathcal{N}(\mu_0, \frac{1}{\lambda_0 \tau_0})$ becomes infinite ($\frac{1}{\lambda_0 \tau_0} \rightarrow \infty$ and $\lambda_0 \tau \rightarrow 0$). Since τ follows a Gamma distribution $Gamma(\alpha_0, \beta_0)$ with expectation $\mathbb{E}(\tau) = \frac{\alpha_0}{\beta_0}$, α_0 and β_0 should be chosen such that $\frac{\alpha_0}{\beta_0} \rightarrow 0$. Given the hyperparameter space $\lambda_0 > 0$, $\alpha_0 \geq 1$, and $\beta_0 \geq 0$, it is recommended to set $\alpha_0 = 1$ and $\mu_0 = 0$ to center the Normal distribution. λ_0 should be chosen small enough and β_0 should have a sufficiently large value (Bai et al. 2014).

Monte Carlo Belief State Update The belief state can be updated exactly according to Eq. 1. However, exact Bayes updates may be computationally infeasible in POMDPs with large state spaces due to the curse of dimensionality. For this reason, we approximate the belief state b_{h_t} for history h_t with a particle filter as described in (Silver and Veness 2010). The belief state b_{h_t} is represented by a set $\hat{b}_{h_t}$ of K sample states or particles. After execution of a_t and observation of o_{t+1}, the particles are updated by Monte Carlo simulation. Sampled states $s_t \sim \hat{b}_{h_t}$ are simulated with a_t such that $\langle s_{t+1}, o'_{t+1}, r_t \rangle \sim \hat{M}(s_t, a_t)$. If $o'_{t+1} = o_{t+1}$, then s_{t+1} is added to $\hat{b}_{h_t a_t o_{t+1}} = \hat{b}_{h_{t+1}}$.

POOLTS To evaluate the effectiveness of POSTS compared to other open-loop planners, we first define *Partially Observable Open-Loop Thompson Sampling (POOLTS)* and show that POOLTS is able to converge to an optimal open-loop plan, if sufficient computational and memory resources are provided. POOLTS is a tree-based approach based on OLUCT from (Lecarpentier et al. 2018). Each node N_t represents a Thompson Sampling bandit and stores $\overline{X_{a_t}}$, s_a, and n_{a_t} for each action $a_t \in \mathcal{A}$.

A simulation starts at a state s_t, which is sampled from the current belief state b_{h_t}. The belief state is approximated by a particle filter $\hat{b}_{h_t}$ as described above. An open-loop tree (Fig. 1b) is iteratively constructed by traversing the current tree in a selection step by using Thompson Sampling to select actions. When a leaf node is reached, it is expanded by a child node N_{new} and a rollout is performed by using a policy $\pi_{rollout}$ until a terminal state is reached or a maximum search depth T is exceeded. The observed rewards are accumulated to returns G_t (Eq. 2) and propagated back to update the corresponding bandit of every node in the simulated path. When the computation budget n_b has run out, the action a_t with the highest expected return $\overline{X_{a_t}}$ is selected for execution. The complete formulation of POOLTS is given in Algorithm 2.

Algorithm 2 POOLTS Planning

```
procedure $POOLTS(h_t, T, n_b)$
    Create $N_0$ for $h_t$
    while $n_b > 0$ do
        $n_b \leftarrow n_b - 1$
        $s_t \sim \hat{b}_{h_t}$
        $TreeSearch(N_0, s_t, T, 0)$
    return $argmax_{a_t \in \mathcal{A}}(\overline{X_{a_t}})$

procedure $TreeSearch(N_t, s_t, T, d)$
    if $d \geq T$ or $s_t$ is a terminal state then return 0
    if $N_t$ is a leaf node then
        Expand $N_t$
        Perform rollout with $\pi_{rollout}$ to sample $G_t$
        return $G_t$
    $a_t \leftarrow ThompsonSampling(N_t)$
    $\langle s_{t+1}, r_t, o_{t+1} \rangle \sim \hat{M}(s_t, a_t)$
    $R_t \leftarrow TreeSearch(N_{t+1}, s_{t+1}, T, d+1)$
    $G_t \leftarrow r_t + \gamma R_t$
    $UpdateBandit(N_t, G_t)$
    return $G_t$
```

(Kocsis and Szepesvári 2006; Bubeck and Munos 2010; Lecarpentier et al. 2018) have shown that tree search algorithms using UCB1 converge to the optimal closed-loop or open-loop plan respectively, if the computation budget n_b is sufficiently large. This is because the expected state or node values in the leaf nodes become stationary, given a stationary rollout policy $\pi_{rollout}$. This enables the values in the preceding nodes to converge as well, leading to state- or node-wise optimal actions. By replacing UCB1 with Thompson Sampling, the tree search should still converge to the optimal closed-loop or open-loop plan, since Thompson Sampling also converges to the optimal action, if the return distribution of G_t becomes stationary (Agrawal and Goyal 2013). (Chapelle and Li 2011; Bai et al. 2014) empirically demonstrated that Thompson Sampling converges faster than UCB1, when rewards are sparse and when the number of arms is large.

POSTS *Partially Observable Stacked Thompson Sampling (POSTS)* is an open-loop approach, which optimizes a stack of T Thompson Sampling bandits to search for high-quality open-loop plans (Fig. 1c). Each bandit N_t stores $\overline{X_{a_t}}$, s_a, and n_{a_t} for each action $a_t \in \mathcal{A}$.

Similarly to POOLTS, a simulation starts at a state s_t, which is sampled from a particle filter $\hat{b}_{h_t}$, representing the current belief state b_{h_t}. Unlike POOLTS, a fixed size stack of bandits $N_1, ..., N_T$ is used to sample plans $p_T = [a_1, ..., a_T]$. p_T is evaluated with the generative model $\hat{M}$ to observe immediate rewards $r_1, ..., r_T$, which are accumulated to returns $G_1, ..., G_T$ (Eq. 2). Each bandit N_t is then updated with the corresponding return G_t. When the computation budget n_b has run out, the action a_t with the highest expected return $\overline{X_{a_t}}$ is selected for execution. The complete formulation of POSTS is given in Algorithm 3.

Algorithm 3 POSTS Planning

```
procedure $POSTS(h_t, T, n_b)$
    while $n_b > 0$ do
        $n_b \leftarrow n_b - 1$
        $s_t \sim \hat{b}_{h_t}$
        $Simulate(N_0, s_t, T, 0)$
    return $argmax_{a_t \in \mathcal{A}}(\overline{X_{a_t}})$
procedure $Simulate(s_t, T, d)$
    if $d \geq T$ or $s_t$ is a terminal state then return $0$
    $a_t \leftarrow ThompsonSampling(N_t)$
    $\langle s_{t+1}, r_t, o_{t+1} \rangle \sim \hat{M}(s_t, a_t)$
    $R_t \leftarrow Simulate(s_{t+1}, T, d + 1)$
    $G_t \leftarrow r_t + \gamma R_t$
    $UpdateBandit(N_t, G_t)$
    return $G_t$
```

The idea of POSTS is to only regard the temporal dependencies between the actions of an open-loop plan. The bandit stack is used to learn these dependencies with the expected (discounted) return. When a bandit N_t samples an action a_t with a resulting reward of r_t, then all preceding bandits N_{t-k} with $k < t$ are updated with $G_{t-k} = r_{t-k} + \gamma r_{t-k+1} + ... + \gamma^k r_t$, using a discounted value of r_t. This is because the actions sampled by all preceding bandits are possibly relevant for obtaining the reward r_t. By only regarding these temporal dependencies, POSTS is *memory bounded*, not requiring a search tree to model dependencies between histories or history distributions (Fig. 1a and 1b).

Experiments

Evaluation Environments

We tested POSTS in the *RockSample*, *Battleship*, and *PocMan* domains, which are well-known POMDP benchmark problems for decision making in POMDPs (Silver and Veness 2010; Somani et al. 2013; Bai et al. 2014). For each domain, we set the discount factor γ as proposed in (Silver and Veness 2010). The results were compared with POMCP, POOLTS, and a partially observable version of OLUCT, which we call POOLUCT. The problem-size features of all domains are shown in Table 1.

The *RockSample(n,k)* problem simulates an agent moving in an $n \times n$ grid containing k rocks (Smith and Simmons 2004). Each rock can be *good* or *bad* but the true state of each rock is unknown. The agent has to sample good rocks, while avoiding to sample bad rocks. It has a noisy sensor, which produces an observation $o_t \in \{good, bad\}$ for a particular rock. The probability of sensing the correct state of the rock decreases exponentially with the agent's distance to that rock. Sampling gives a reward of $+10$, if the rock is good and -10 otherwise. If a good rock was sampled, it becomes bad. Moving and sensing do not give any rewards. Moving past the east edge of the grid gives a reward of $+10$ and the episode terminates. We set $\gamma = 0.95$.

In *Battleship* five ships of size 1, 2, 3, 4, and 5 respectively are randomly placed into a 10×10 grid, where the agent has to sink all ships without knowing their actual positions (Silver and Veness 2010). Each cell hitting a ship gives a reward of $+1$. There is a reward of -1 per time step and a terminal reward of $+100$ for hitting all ships. We set $\gamma = 1$.

PocMan is a partially observable version of *PacMan* (Silver and Veness 2010). The agent navigates in a 17×19 maze and has to eat randomly distributed food pellets and power pills. There are four ghosts moving randomly in the maze. If the agent is within the visible range of a ghost, it is getting chased by the ghost and dies, if it touches the ghost, terminating the episode with a reward of -100. Eating a power pill enables the agent to eat ghosts for 15 time steps. In that case, the ghosts will run away, if the agent is under the effect of a power pill. At each time step a reward of -1 is given. Eating food pellets gives a reward of $+10$ and eating a ghost gives $+25$. The agent can only perceive ghosts, if they are in its direct line of sight in each cardinal direction or within a hearing range. Also, the agent can only sense walls and food pellets, which are adjacent to it. We set $\gamma = 0.95$.

Methods

POMCP We use the POMCP implementation from (Silver and Veness 2010). π_{tree} selects actions from a set of legal actions $a_t \in \mathcal{A}_{legal}(s_t)$ with UCB1. $\pi_{rollout}$ randomly selects actions from $\mathcal{A}_{legal}(s')$, depending on the currently simulated state $s' \in \mathcal{S}$.

In each simulation step, there is at most one expansion step, where new nodes are added to the search tree. Thus, tree size should increase linearly w.r.t. n_b in large POMDPs.

POOLTS and POOLUCT POOLTS is implemented according to Algorithm 2, where actions are selected from a set of legal actions $a_t \in \mathcal{A}_{legal}(s_t)$ with Thompson Sampling (Algorithm 1) in the first stage. $\pi_{rollout}$ randomly selects actions from $\mathcal{A}_{legal}(s')$, depending on the currently simulated state $s' \in \mathcal{S}$. POOLUCT is similar to POOLTS but uses UCB1 as action selection strategy in the first stage. Since, open-loop planning can encounter different states at the same node (Fig. 1), the set of legal actions may vary for each state $s_t \in \mathcal{S}$. We always mask out the statistics of currently illegal actions, regardless of whether they have high average action values, to avoid selecting them.

Similarly to POMCP, the search tree size should increase linearly w.r.t. n_b, but with less nodes, since open-loop trees store summarized information about history distributions.

POSTS POSTS is implemented as a stack of Thompson Sampling bandits N_i with $1 \leq i \leq T$ according to Algorithm 3. Starting at s_t, all bandits N_i apply Thompson Sampling to a set of legal actions $\mathcal{A}_{legal}(s')$, depending on the currently simulated state $s' \in \mathcal{S}$. Similarly to POOLTS and POOLUCT, we mask out the statistics of currently illegal actions and only regard the value statistics of legal actions for selection during planning.

Given a horizon of T, POSTS always maintains T Thompson Sampling bandits, independently of the computation budget n_b.

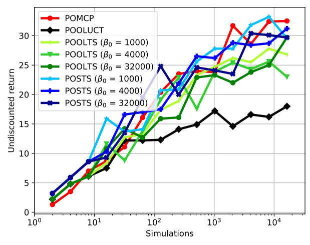
Table 1: The problem-size features of the benchmark domains *RockSample*, *Battleship*, and *PocMan*.

	RockSample(11,11)	*RockSample(15,15)*	*Battleship*	*PocMan*
# States $\|\mathcal{S}\|$	$247,808$	$7,372,800$	$\sim 10^{12}$	$\sim 10^{56}$
# Actions $\|\mathcal{A}\|$	16	20	100	4
# Observations $\|\mathcal{O}\|$	3	3	2	1024

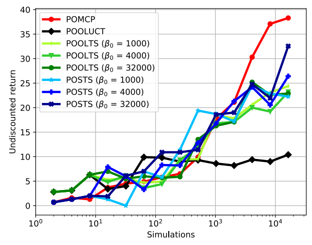
Results

We ran each approach on *RockSample*, *Battleship*, and *PocMan* with different settings for 100 times or at most 12 hours of total computation. We evaluated the performance of each approach with the *undiscounted return* ($\gamma = 1$), because we focus on the actual effectiveness instead of the quality of optimization (Bai et al. 2014). For POMCP and POOLTS we set the UCB1 exploration constant c to the reward range of each domain as proposed in (Silver and Veness 2010).

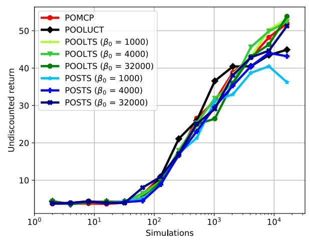
Prior Sensitivity Since we assume no additional domain knowledge, we focus on uninformative priors with $\mu_0 = 0$, $\alpha_0 = 1$, and $\lambda_0 = 0.01$ as proposed in (Bai et al. 2014). With this setting, β_0 controls the degree of initial exploration during the planning phase, thus its impact on the performance of POOLTS and POSTS is evaluated. The results are shown in Fig. 2 for $\beta_0 = 1000, 4000, 32000$ for POOLTS and POSTS.



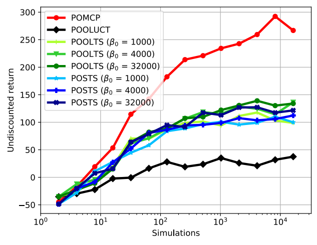
(a) *RockSample(11,11)*



(b) *RockSample(15,15)*



(c) *Battleship*

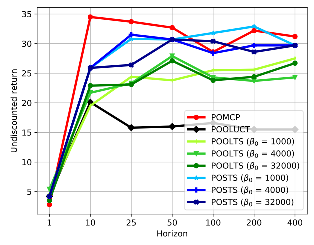


(d) *PocMan*

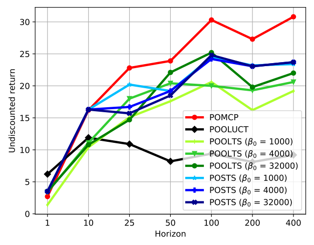
Figure 2: Average performance of POSTS, POOLTS, POOLUCT, and POMCP with different prior values for β_0, different computation budgets n_b, and a horizon of $T = 100$.

In *RockSample*, POSTS slightly outperforms POOLTS and keeps up in performance with POMCP. POOLTS slightly outperforms POSTS and POMCP in *Battleship* with POSTS only being able to keep up when $n_b < 10^4$ or when $\beta_0 = 32000$. POMCP clearly outperforms all open-loop approaches in *PocMan*. POOLTS slightly outperforms POSTS in *PocMan* with POSTS only being to keep up, if $\beta_0 = 32000$. POOLUCT performed worst in all domains except *Battleship*, where it performs best with a computation budget of $n_b \leq 1024$. POSTS performs slightly better, if β_0 is large, but POOLTS seem to be insensitive to the choice of β_0 except in *PocMan*, where it performs better, if β_0 is large.

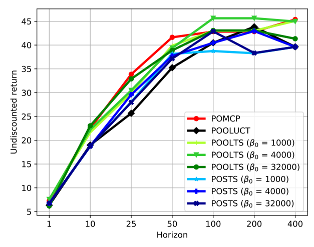
Horizon Sensitivity We evaluated the sensitivity of all approaches w.r.t. different horizons T. The results are shown in Fig. 3 for $n_b = 4096$ [1] and $\beta_0 = 1000, 4000, 32000$ for POOLTS and POSTS.



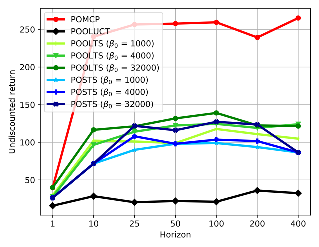
(a) *RockSample(11,11)*



(b) *RockSample(15,15)*



(c) *Battleship*



(d) *PocMan*

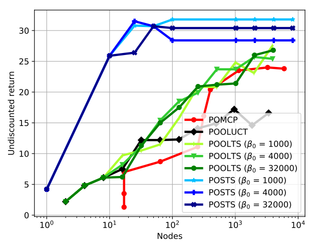
Figure 3: Average performance of POSTS, POOLTS, POOLUCT, and POMCP with different planning horizons T and a computation budget of $n_b = 4096$.

In *RockSample(11,11)*, there is a performance peak at $T = 10$ for POMCP and POOLUCT, while for POSTS and POOLTS it is about $T = 50$. In all other domains, there seems to be a performance peak at $T = 100$ for most approaches. If $T > 100$, there is no significant improvement or even degrading performance for most approaches except for POMCP, which slightly improves in all domains but *RockSample(11,11)*, if $T = 400$.

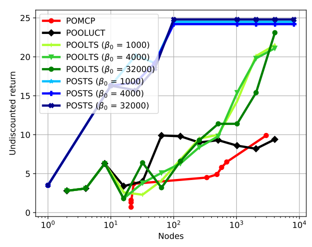
Performance-Memory Tradeoff We evaluated the performance-memory tradeoff of all approaches by introducing a memory capacity n_{mem}, where the computation is interrupted, when the number of nodes exceeds n_{mem}. For

[1]Using computation budgets between 1024 and 16384 led to similar plots, thus we stick to $n_b = 4096$ with all approaches requiring less than one second per action (Silver and Veness 2010).

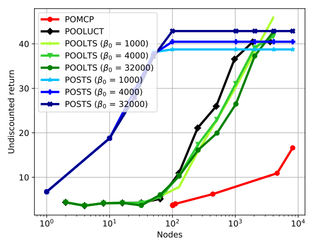
POMCP, we count the number of *o-nodes* and *a-nodes* (Fig. 1a). For POOLTS and POOLUCT, we count the number of history distribution nodes (Fig. 1b). For POSTS, we count the number of Thompson Sampling bandits, which is always $min\{n_{mem}, T\}$. The results are shown in Fig. 4 for $n_b = 4096$, $T = 100$, and $\beta_0 = 1000, 4000, 32000$ for POOLTS and POSTS. POSTS never uses more than $10^2 = 100$ nodes in each setting.



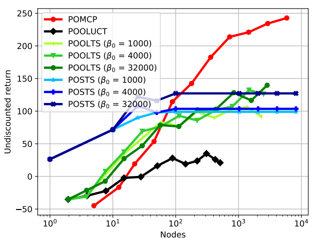
(a) *RockSample(11,11)*



(b) *RockSample(15,15)*



(c) *Battleship*



(d) *PocMan*

Figure 4: Average performance of POSTS, POOLTS, POOLUCT, and POMCP with memory bounds, a computation budget of $n_b = 4096$ and a horizon of $T = 100$.

In *Rocksample* and *Battleship*, POMCP is outperformed by POSTS and POOLTS (and also POOLUCT in *Battleship*). POSTS always performs best in these domains, when $n_{mem} < 1000$. POMCP performs best in *PocMan* by outperforming POSTS, when $n_{mem} > 100$ and POOLTS keeps up with the best POSTS setting, when $n_{mem} > 1000$. POOLUCT performs worst except in *Battleship*, improving less and slowest with increasing n_{mem}. It outperforms POMCP in *Rocksample(15,15)*, when $n_{mem} \leq 1000$ though. In *PocMan*, POOLUCT creates less than 550 nodes, when $n_b = 4096$, indicating that the search tree construction has converged and does not improve any further.

Discussion

The experiments show that partially observable open-loop planning can be a good alternative to closed-loop planning, when the action space is large, stochasticity is low, and when computational and memory resources are highly restricted. Especially approaches based on Thompson Sampling like POOLTS and POSTS seem to be very effective and robust w.r.t. the hyperparameter choice. Setting a large value for β_0 seems to be beneficial for large problems (Fig. 2). This is because an enormous search space needs to be explored, while avoiding premature convergence to poor solutions. However, if β_0 is too large, POSTS and POOLTS might converge too slowly, thus requiring much more computation (Bai et al. 2014). If T is too large, then the value estimates have very high variance, making bandit adaptation more difficult. This could explain the performance stagnation or degradation for most approaches in Fig. 3, when $T > 100$. The performance of POSTS scales similarly to POOLTS w.r.t. n_b and T (Fig. 2 and 3). POSTS is also more robust than POOLUCT w.r.t. changes to n_b and T except in *Battleship*, where both approaches scale similarly, when β_0 is sufficiently large.

POSTS is competitive to POOLTS and superior to POOLUCT in all settings except in *Battleship* (when n_b is large) with POOLTS and POOLUCT being shown to theoretically converge to optimal open-loop plans, given sufficient computation budget n_b and memory capacity n_{mem}. POSTS is shown to be superior to all other approaches in *RockSample* and *Battleship*, when memory resources are highly restricted, only being outperformed by the tree-based approaches in *Battleship* after thousands of nodes were created, consuming much more memory than POSTS, which only uses 100 nodes at most. This might be due to the relatively large action space of these domains (Table 1), where all tree-based planners construct enormous trees with high branching factors, when exploring the effect of each action. *RockSample* and *Battleship* have low stochasticity, since state transitions are deterministic. In both domains the agent is primarily uncertain about the real state, thus the planning quality only depends on the belief state approximation and the uncertainty about observations (only in *RockSample*).

POMCP performs best in *PocMan*. This might be due to the small action space (Table 1) and high stochasticity (where all ghosts primarily move randomly), since open-loop planning is known to converge to sub-optimal solutions in such domains (Weinstein and Littman 2013; Lecarpentier et al. 2018). However, POMCP has the highest memory consumption, since it constructs larger trees than open-loop approaches with the same computation budget (Fig 1a). In *PocMan*, POSTS is able to keep up with POOLTS, while being much more memory-efficient (Fig. 2d and 4d).

Conclusion and Future Work

In this paper, we proposed *Partially Observable Stacked Thompson Sampling (POSTS)*, a memory bounded approach to open-loop planning in large POMDPs, which optimizes a fixed size stack of Thompson Sampling bandits.

To evaluate the effectiveness of POSTS, we formulated a tree-based approach, called POOLTS and showed that POOLTS is able to find optimal open-loop plans with sufficient computational and memory resources.

We empirically tested POSTS in four large benchmark problems and showed that POSTS achieves competitive performance compared to tree-based open-loop planners like POOLTS and POOLUCT, if sufficient resources are provided. Unlike tree-based approaches, POSTS offers a performance-memory tradeoff by performing best, if computational and memory resources are highly restricted, making it suitable for efficient partially observable planning.

For the future, we plan to extend POSTS to multi-agent systems (Phan et al. 2018).

References

Agrawal, S., and Goyal, N. 2013. Further Optimal Regret Bounds for Thompson Sampling. In *Artificial Intelligence and Statistics*, 99–107.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine learning* 47(2-3):235–256.

Bai, A.; Wu, F.; Zhang, Z.; and Chen, X. 2014. Thompson Sampling based Monte-Carlo Planning in POMDPs. In *Proceedings of the Twenty-Fourth International Conferenc on International Conference on Automated Planning and Scheduling*, 29–37. AAAI Press.

Bai, A.; Wu, F.; and Chen, X. 2013. Bayesian Mixture Modelling and Inference based Thompson Sampling in Monte-Carlo Tree Search. In *Advances in Neural Information Processing Systems*, 1646–1654.

Belzner, L., and Gabor, T. 2017. Stacked Thompson Bandits. In *Proceedings of the 3rd International Workshop on Software Engineering for Smart Cyber-Physical Systems*, 18–21. IEEE Press.

Bubeck, S., and Munos, R. 2010. Open Loop Optimistic Planning. In *COLT*, 477–489.

Chapelle, O., and Li, L. 2011. An Empirical Evaluation of Thompson Sampling. In *Advances in neural information processing systems*, 2249–2257.

Honda, J., and Takemura, A. 2014. Optimality of Thompson Sampling for Gaussian Bandits depends on Priors. In *Artificial Intelligence and Statistics*, 375–383.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artificial intelligence* 101(1):99–134.

Kaufmann, E.; Korda, N.; and Munos, R. 2012. Thompson Sampling: An Asymptotically Optimal Finite-Time Analysis. In *International Conference on Algorithmic Learning Theory*, 199–213. Springer.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo Planning. In *ECML*, volume 6, 282–293. Springer.

Lecarpentier, E.; Infantes, G.; Lesire, C.; and Rachelson, E. 2018. Open Loop Execution of Tree-Search Algorithms. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2362–2368. IJCAI Organization.

Perez Liebana, D.; Dieskau, J.; Hunermund, M.; Mostaghim, S.; and Lucas, S. 2015. Open Loop Search for General Video Game Playing. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 337–344. ACM.

Phan, T.; Belzner, L.; Gabor, T.; and Schmid, K. 2018. Leveraging Statistical Multi-Agent Online Planning with Emergent Value Function Approximation. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '18, 730–738. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Pineau, J.; Gordon, G.; and Thrun, S. 2006. Anytime Point-based Approximations for Large POMDPs. *Journal of Artificial Intelligence Research* 27:335–380.

Powley, E.; Cowling, P.; and Whitehouse, D. 2017. Memory Bounded Monte Carlo Tree Search. *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-Draa, B. 2008. Online Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32:663–704.

Silver, D., and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In *Advances in neural information processing systems*, 2164–2172.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529(7587):484–489.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the Game of Go without Human Knowledge. *Nature* 550(7676):354–359.

Smith, T., and Simmons, R. 2004. Heuristic Search Value Iteration for POMDPs. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, 520–527. AUAI Press.

Somani, A.; Ye, N.; Hsu, D.; and Lee, W. S. 2013. DESPOT: Online POMDP Planning with Regularization. In *Advances in neural information processing systems*, 1772–1780.

Thompson, W. R. 1933. On the Likelihood that One Unknown Probability exceeds Another in View of the Evidence of Two Samples. *Biometrika* 25(3/4):285–294.

Weinstein, A., and Littman, M. L. 2013. Open-loop Planning in Large-Scale Stochastic Domains. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 1436–1442. AAAI Press.

Yu, C.; Chuang, J.; Gerkey, B.; Gordon, G.; and Ng, A. 2005. Open-Loop Plans in Multi-Robot POMDPs. Technical report, Stanford CS Dept.