xxx 1 (2017) x-xx Submitted x/xx; published x/xx

Markovian State and Action Abstractions for Markov Decision Processes via Hierarchical Monte Carlo Tree Search

Aijun Bai AIJUNBAI@BERKELEY.EDU
University of California at Berkeley

Siddharth Srivastava SRIVASS@UTRC.UTC.COM
United Technology Research Center

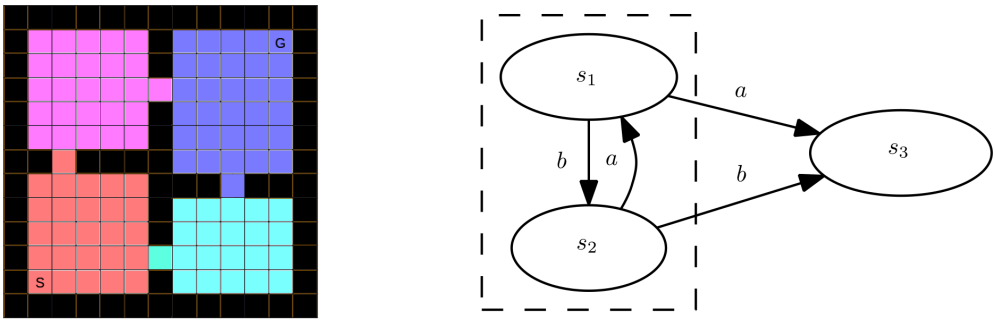
Stuart Russell RUSSELL@CS.BERKELEY.EDU
University of California at Berkeley

Abstract

Markov decision processes (MDPs) provide a rich framework for planning and learning under uncertainty. In the context of hierarchical planning and learning for MDPs, state abstraction is an important technique for scaling MDP algorithms by treating a group of states as a unit. As is well known, however, it introduces difficulties due to the non-Markovian nature of state-abstracted models. Whereas prior approaches rely upon ad hoc fixes for this issue, we propose instead to view the state-abstracted model as a partially observable MDP (POMDP) and show that we can thereby take advantage of state abstraction without sacrificing the Markov property. We bound the performance loss induced by the abstraction, and exploit the hierarchical structure introduced by state abstraction by extending the theory of options to a POMDP setting. In this context, we propose a hierarchical Monte Carlo tree search algorithm for approximately solving the abstracted POMDP and show that it converges to a recursively optimal hierarchical policy. Both theoretical and empirical results suggest that abstracting an MDP into a POMDP yields a scalable solution approach.

1. Introduction

The general task of sequential decision-making under uncertainty is of great interest to the *Artificial Intelligence* (AI) community (Russell & Norvig, 2003). It embraces a broad range of common problems found in planning and learning. Currently, the most general and clear fundamental formulation for these problems is achieved through the theory of *Markov decision processes* (MDPs), which provides a rich mathematical framework for planning and learning under uncertainty in fully observable environments (Puterman, 1994). State-of-the-art *offline* algorithms, such as linear programming, value iteration, and policy iteration, solve MDPs by computing a policy for the entire state space before the agent is deployed to interact with the environment. In practice, offline algorithms often suffer from the problem of scalability due to the well-known "curse of dimensionality" — the size of state space grows exponentially with the number of state variables (Littman, Dean, & Kaelbling, 1995). On the other hand, *online* algorithms alleviate this difficulty by computing a near-optimal policy merely for the current state. The basic observation is that an agent can only encounter a fraction of the overall states at run-time interacting with the environment. Typically, an online algorithm evaluates available actions for the current state and selects the seemingly best one by exploring the underlying expectimax search tree (Barto, Bradtke, & Singh, 1995; Hansen & Zilberstein, 2001; Kocsis & Szepesvári, 2006; Bai, Wu, & Chen, 2015).



(a) The rooms example.

(b) A 3-state MDP.

Figure 1: State abstraction examples.

In this paper, we consider the problem of approximately solving MDPs online. Take *Monte Carlo tree search* (MCTS), which is a popular online planning algorithm for MDPs, as an example. The idea of MCTS is to build a best-first search tree by simulating a *tree policy* and a *rollout policy* to estimate the optimal action values using sampled trajectories (Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, & Colton, 2012b). It has been observed that the performance of MCTS is typically dominated by the effective search depth before exhausting the total computation time within each action-selection step (Kearns, Mansour, & Ng, 1999; Hostetler, Fern, & Dietterich, 2014), which is in turn determined by the branching factor of the search tree. In MDPs, the branching factor consists of action branching and stochastic branching. Action branching depends on the number of available actions; stochastic branching depends on the number of possible outcomes for an action. Most online planning algorithms build search trees in the *ground state* (or *concrete state*) space; for large problems, however, the branching factor leads to poor performance as the feasible search depth is typically too small provided with limited computation resources.

As an important technique for scaling MDP algorithms, *state abstraction* (or *state aggregation*) reduces the stochastic branching factor by treating a group of states as a unit (Giunchiglia & Walsh, 1992; Dearden & Boutilier, 1997; Li, Walsh, & Littman, 2006; Konidaris, 2016). As an example, Figure 1a illustrates a *rooms* domain, where a robot needs to navigate from position S to position G. In the figure, black cells represent walls; all other cells are valid ground states. Cells sharing the same color correspond to the same abstract state, which in turn represents a room. The abstracted problem has typically a reduced abstract state space. However, abstraction results in a non-Markovian model (Srivastava, Russell, & Pinto, 2016), because the transition probability of reaching the next abstract state and the reward received by taking an action within an abstract state depend on the *occupancy probability* over concrete states represented by that abstract state. It is well known that the occupancy probability depends on the history of all past taken actions and visited abstract states (Jaakkola, Singh, & Jordan, 1995). Take the 3-state deterministic MDP depicted in Figure 1b as an example, where an edge represents a deterministic transition that is invoked by executing the labeled action. If we group states s_1 and s_2 into an abstract state $s_{1,2}$, then the probability of reaching state s_3 after taking action a in $s_{1,2}$ equals the probability of being actually in state s_1, which depends exactly on the initial state and the number of times that the agent has executed action b in ground state s_1 and action a in ground state s_2 in the past history prior to entering state s_3.

Safe state abstraction methods avoid the non-Markovian problem by eliminating only irrelevant state variables (Dietterich, 1999; Andre & Russell, 2002; Jong & Stone, 2005; Hengst, 2007) or exploiting particular symmetric structure within the transition and reward models (such as *bisimulation* or *homomorphism*) (Dean & Givan, 1997; Dearden & Boutilier, 1997; Dean, Givan, & Leach, 1997; Ravindran & Barto, 2002, 2003a; Givan, Dean, & Greig, 2003; Ravindran & Barto, 2003b; Pineau, Gordon, & Thrun, 2003; Ravindran & Barto, 2004; Wolfe & Barto, 2006; Taylor, Precup, & Panagaden, 2009; Ferns, Castro, Precup, & Panangaden, 2012; Jiang, Singh, & Lewis, 2014; Anand, Grover, Mausam, & Singla, 2015). Such methods compute near-lossless abstractions, which almost preserve the optimal policies for MDPs, but are often of limited usefulness, because near-lossless abstractions rarely exist and are computationally difficult to find. Popular proposals to resolve this situation are *bounded-parameter Markov decision processes* (BMDPs) and the *weighting function* approach. BMDP explicitly represent the abstract MDP with uncertain transition and reward functions resulting from doing state abstraction (Givan, Leach, & Dean, 1997). An interval value iteration algorithm is developed to compute polices for a BMDP that are optimal in terms of so-called optimistic and pessimistic criteria. However, general state abstraction usually introduces significantly high ranges in the parameter intervals defining the resulting BMDP, and thus the found policy has limited optimality guarantee in the original concrete MDP. For example, the abstracted BMDP resulting from doing state abstraction in the 3-state MDP depicted in Figure 1b would have a trivial bounded transition function stating that $T(s_3 \mid s_{1,2}, a) \in [0, 1]$. The weighting function approach introduces an ad hoc weighting function (also known as *aggregation probability*), which functions like an occupancy probability for each concrete state given an abstract state (Bertsekas, 1995; Singh, Jaakkola, & Jordan, 1995; Li et al., 2006; Hostetler et al., 2014). Superficially, this ensures that the abstract transition and reward functions can be written in a Markovian way. It is usually assumed that the weighting function is manually specified and remains constant in run-time. We argue that such approaches cannot be accurate enough to capture the true dynamics of the abstract system, where the occupancy probability is in fact non-stationary, depending on the whole history of past actions and abstract states, or in other words, the policy being computed/executed!

In this paper, we show that a ground MDP with state abstraction turns out to be a *partially observable Markov decision process* (POMDP). Generally speaking, POMDP extends MDP to partially observable environments (Kaelbling, Littman, & Cassandra, 1998). The POMDP resulting from doing state abstraction over an MDP has the original ground MDP as the underlying MDP and the set of abstract states as the set of observations (Bai, Srivastava, & Russell, 2016). Belief states in the resulting POMDP replace the otherwise necessary aggregation probability as in the weighting function approach, with the advantage that the belief state can be calculated by Bayesian updating. We show that algorithms such as POMCP can be naturally extended to do online planning for the ground MDP. Particularly, observing that the set of abstract states introduces automatically a hierarchical structure, we further define temporal transitions between abstract states as abstract actions by extending the theory of *options* (Sutton, Precup, & Singh, 1999) to a POMDP setting, and develop a hierarchical MCTS algorithm that handles Markovian state and action abstractions for MDPs following a POMDP formulation. Theoretically, we show that the performance loss in terms of action values due to approximation in state abstraction is bounded by a constant multiple of a state aggregation error introduced by grouping states with different optimal actions; and, the resulting algorithm converges to a recursively optimal hierarchical policy consistent with the input state and action abstractions. Perhaps counterintuitively, we find that a hierarchical MCTS algorithm solving the abstracted POMDP can outperform ground MCTS by orders of magnitude. The

main contributions of this paper are: 1) we formalize state abstraction on an MDP as a POMDP and bound its performance loss; and 2) we develop a hierarchical Monte Carlo tree search algorithm with action abstraction for approximately solving the resulting abstracted POMDP.

The remainder of this paper is organized as follows. Section 2 briefly reviews some background of this paper. Section 4 presents the main approach. Section 5 gives our theoretical findings in terms of performance loss, optimality guarantee and convergence. Section 6 shows the empirical results on the rooms domain and an extended continuous-rooms domain. Section 3 introduces some related work. Finally, we conclude with discussion of future work in Section 7.

2. Background

In this section, we briefly review background on Markov decision processes (MDPs), partially observable Markov decision processes (POMDPs), and Monte Carlo tree search (MCTS) algorithms. We also introduce some existing state- and action- abstraction techniques in the literature.

2.1 MDPs and POMDPs

MDPs provide a rich mathematical framework for planning and learning under uncertainty in fully observable environments (Puterman, 1994).

Definition 1 (Markov Decision Process)**.** *A Markov decision process (MDP) is a tuple $\langle S, A, T, R, \gamma \rangle$, where S is the finite state space, A is the finite action space, $T : S \times A \times S \to [0, 1]$ is the transition function with $T(s' \mid s, a)$ being the probability of reaching state $s' \in S$ after taking action $a \in A$ in state $s \in S$, $R : S \times A \to \mathbb{R}$ is the reward function with $R(s, a)$ being the immediate reward collected by taking action $a \in A$ in state $s \in S$, and $0 < \gamma \leq 1$ is a discount factor.*

Given an MDP $M = \langle S, A, T, R, \gamma \rangle$, a (stationary) *policy* $\pi : S \to A$ is a mapping from states to actions with $\pi(s)$ being the action to be taken in state s. The *value function* $V^\pi(s)$ of a policy π is defined as the expected cumulative reward received by following policy π starting from state s:

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))\right], \tag{1}$$

where s_t is the state at time step t. The corresponding *action-value function* $Q^\pi(s, a)$ is defined as the expected cumulative reward received by taking action a in state s and following π thereafter:

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s' \mid s, a) V^\pi(s'). \tag{2}$$

A solution for an MDP is an *optimal policy* π^* that maximizes the value function for all states. The corresponding optimal value function V^{π^*} (or V^* for short) satisfies the Bellman equation (Bellman, 1957):

$$V^*(s) = \max_{a \in A} \left\{ R(s, a) + \gamma \sum_{s' \in S} T(s' \mid s, a) V^*(s') \right\}. \tag{3}$$

In presence of partially observable environments, POMDPs extend MDPs by introducing an observation model (Kaelbling et al., 1998).

Definition 2 (Partially Observable Markov Decision Process)**.** *A partially observable Markov decision process (POMDP) is defined as a tuple $\langle S, A, Z, T, R, \Omega, \gamma \rangle$, where $\langle S, A, T, R, \gamma \rangle$ is the underlying MDP, Z is the finite observation space and $\Omega : S \times A \times Z \to [0, 1]$ is the observation function with $\Omega(z \mid s, a)$ being the probability of observing $z \in Z$ after having taken action $a \in A$ and resulting in state $s \in S$.*

Given a POMDP $P = \langle S, A, Z, T, R, \Omega, \gamma \rangle$, a *belief state* (or *belief* for short) $b : S \to [0, 1]$ is defined as a probability distribution over states with $b(s)$ being the probability of being in state s. Given current belief b, after taking action a and observing observation z, the new belief b' can be computed in a Bayesian filtering way $b' = \tau(b, a, z)$ such that:

$$b'(s') = \eta \Omega(z \mid s', a) \sum_{s \in S} T(s' \mid s, a) b(s), \tag{4}$$

where $\eta = 1/\Pr(z \mid b, a)$ is a normalizing factor with $\Pr(z \mid b, a) = \sum_{s' \in S} \Omega(z \mid s', a) \sum_{s \in S} T(s' \mid s, a) b(s)$ being the probability of observing observation z after taking action a in belief b. Let $\mathcal{B}$ be the set of all possible belief states. A POMDP can be equivalently transformed to a Bayesian-adaptive MDP (BAMDP) $\langle \mathcal{B}, A, T^+, R^+, \gamma \rangle$ defined over the belief space, with $\mathcal{B}$ being the state space, A being the action space, $R^+(b, a) = \sum_{s \in S} R(s, a) b(s)$ being the reward function and T^+ being the transition function such that:

$$T^+(b' \mid b, a) = \sum_{z \in Z} \mathbf{1}[b' = \tau(b, a, z)] \Pr(z \mid b, a), \tag{5}$$

where $\mathbf{1}$ is the indicator function. Let V^* be the optimal value function of the resulting MDP. The corresponding Bellman equation defined in belief space $\mathcal{B}$ can be written as:

$$V^*(b) = \max_{a \in A} \left\{ R^+(b, a) + \gamma \sum_{z \in Z} \Pr(z \mid b, a) V^*(\tau(b, a, z)) \right\}. \tag{6}$$

We define a history $h_t = [a_0, z_1, a_1, z_2 \cdots, a_{t-1}, z_t]$ at time step t as the sequence of all previous actions and observations, where a_t and z_t are the action and observation at time step t respectively. It can be seen from Equation 4 that, provided with an initial belief state b_0 at time step 0, a history h_t at time step t determines uniquely a belief state $b_t(s \mid h_t)$. Let $\mathcal{H}$ be the set of all possible histories, and $b = \zeta(h)$ be the corresponding belief state given history h and the initial belief state. The Bellman equation defined in history space $\mathcal{H}$ can be written in a similar way:

$$V^*(h) = \max_{a \in A} \left\{ R^+(h, a) + \gamma \sum_{z \in Z} \Pr(z \mid h, a) V^*(haz) \right\}, \tag{7}$$

where $R^+(h, a) = R^+(\zeta(h), a)$, $\Pr(z \mid h, a) = \Pr(z \mid \zeta(h), a)$, and haz denotes the history resulting from taking action a at history h and observing z afterwards, such that $\zeta(haz) = \tau(\zeta(h), a, z)$.

2.2 Monte Carlo Tree Search

The idea of Monte Carlo Tree Search (MCTS) is to build a best-first search tree by simulating a *tree policy* and a *rollout policy* to estimate the optimal action values using sampled trajectories. It is

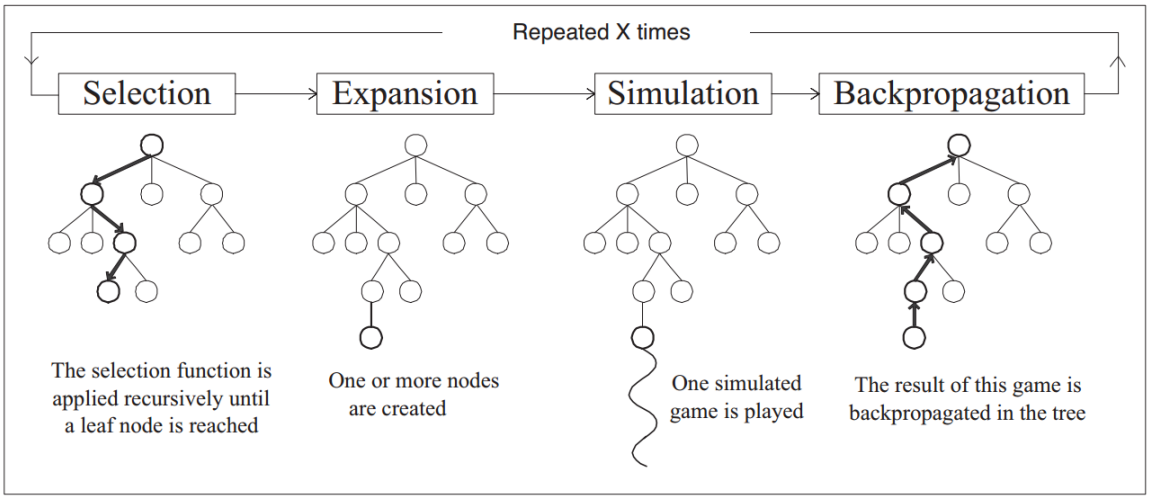


Figure 2: The main process of Monte Carlo tree search on game play (Chaslot et al., 2008).

model-free and requires only a *generative model* as a *simulator* of the underlying problem. MCTS evaluates a node, which corresponds to a state for an MDP or a belief state for a POMDP, in the search tree by: 1) selecting an action according to the tree policy; 2) executing the selected action in the simulator; 3) recursively evaluating the resulting state/belief if it is already in the search tree, otherwise inserting it into the search tree and playing the rollout policy based on Monte Carlo simulations; and 4) updating the statistics of tree nodes by backpropagating the simulation results up to the root node (Chaslot et al., 2008; Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, & Colton, 2012a). The main process of MCTS consisting of selection, expansion, simulation and backprogagation in game play domains is outlined in Figure 2. Iteratively repeating this process, MCTS builds an asymmetric best-first search tree. When interrupted at any time, MCTS reports the best action based on the current action values in the search tree. An example of the resulting search tree is illustrated in Figure 3. MCTS has shown to be computationally efficient, anytime and highly parallelisable. To date, great success has been achieved by MCTS in variety of domains, such as game play (Winands, Bjornsson, & Saito, 2010; Gelly & Silver, 2011; Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, Lanctot, et al., 2016), planning under uncertainty (Kocsis & Szepesvári, 2006; Silver & Veness, 2010), and Bayesian reinforcement learning (Guez, Silver, & Dayan, 2012; Asmuth & Littman, 2011; Vien, Ertel, Dang, & Chung, 2013).

In the context of multi-armed bandits, *Upper confidence bound* (UCB) is the most successful and widely-used algorithm to address the so-called *exploration and exploitation* dilemma (Auer, Cesa-Bianchi, & Fischer, 2002; Auer, 2003). *UCB applied to trees* (UCT) is perhaps one of the most popular implementations of MCTS for MDPs (Kocsis & Szepesvári, 2006; Gelly & Silver, 2007; Finnsson & Björnsson, 2008; Winands et al., 2010; Gelly & Silver, 2011; Keller & Eyerich, 2012; Barrett, Stone, Kraus, & Rosenfeld, 2013). UCT treats each state of the search tree as an

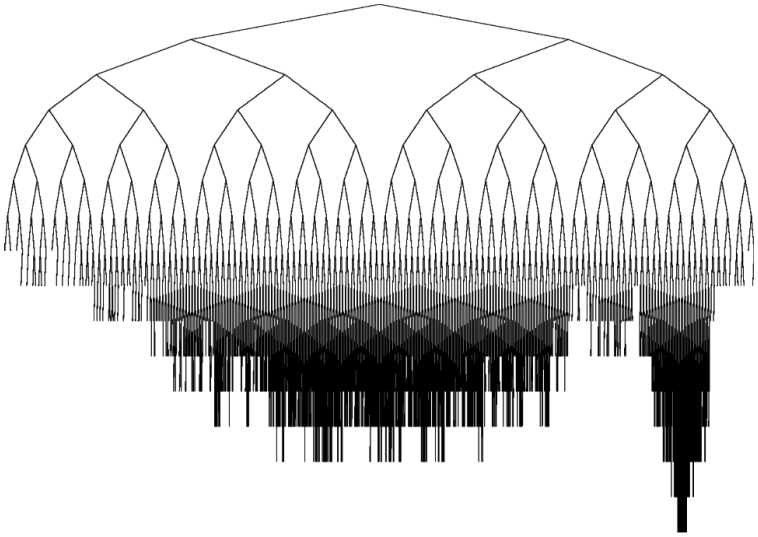


Figure 3: The resulting asymmetric search tree generated by MCTS (Coquelin & Munos, 2007).

MAB, and selects the action that maximizes the UCB heuristic, defined as:

$$\mathrm{UCB}(s,a) = \bar{Q}(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}, \tag{8}$$

where $\bar{Q}(s,a)$ is the mean return of action a in state s from all previous simulations, $N(s,a)$ is the number of times that action a has been selected in state s, $N(s) = \sum_{a \in A} N(s,a)$ is the number of times that state s has been visited, and c is the exploration-exploitation factor. Kocsis and Szepesvári (2006) proved that with an appropriate choice of c the probability of selecting the optimal action converges to 1 as the number of samples grows to infinity.

Partially observable Monte Carlo planning (POMCP) is an extension of UCT to POMDPs (Silver & Veness, 2010). POMCP employs a *root sampling* technique to simulate a trajectory from a sampled state according to the belief state $\zeta(h)$ associated with history h at the root node. POMCP chooses the action that maximizes the UCB heuristic, defined in terms of histories and actions:

$$\mathrm{UCB}(h,a) = \bar{Q}(h,a) + c\sqrt{\frac{\log N(h)}{N(h,a)}}, \tag{9}$$

where $\bar{Q}(h,a)$ is the average outcome of applying action a at history node h over all previous simulations, $N(h,a)$ is the visitation count of action a at history h, $N(h) = \sum_{a \in A} N(h,a)$ is the overall visitation count of history h, and c is the exploration-exploitation factor. Additionally, POMCP uses a particle filter (Gordon, Salmond, & Smith, 1993; Thrun, 1999) to approximate the belief state, by conducting a Monte Carlo procedure to update particles based on sampled observations, rewards, and state transitions. Silver and Veness (2010) show that, for a suitable choice of c, the value function constructed by POMCP converges in probability 1 to the optimal value function. POMCP has been shown with success in various problems (Macindoe, Kaelbling, & Lozano-Pérez, 2012; Vien et al., 2013; Barrett, Agmon, Hazon, Kraus, & Stone, 2014).

2.3 State Abstraction

State abstraction for an MDP is associated with an abstraction function. We adopt the definition for abstraction function of Li et al. (2006).

Definition 3 (State Abstraction). *Let $M = \langle S, A, T, R, \gamma \rangle$ be a ground MDP, and $X = \{x_1, x_2, \cdots\}$ be a partition on state space S. An abstraction function $\varphi : S \to X$ is a mapping from states to abstract states, such that $\varphi(s) \in X$ is the abstract state corresponding to ground state s, and the inverse image $\varphi^{-1}(x)$, with $x \in X$, is the set of ground states that correspond to abstract state x under abstraction φ.*

Given a ground MDP $M = \langle S, A, T, R, \gamma \rangle$ and an abstraction function φ defined over abstract state space X, let $h_t = [a_0, x_1, a_1, x_2, \cdots, a_{t-1}, x_t]$ be the history of past taken actions and visited abstract states at time step t. An *occupancy probability* $\Pr(s \mid h)$ is defined as the probability of being in state s given history h. From a Bayesian perspective, the occupancy probability can be updated in a filtering way:

$$\Pr(s' \mid hax) = \eta \mathbf{1}[x = \varphi(s')] \sum_{s \in S} T(s' \mid s, a) \Pr(s \mid h), \tag{10}$$

where η is a normalizing factor, and hax is the history resulting from taking action a at history h and reaching abstract state x. Let $\Pr(h' \mid h, a)$ be the probability of reaching history h' after taking action a at history h, and $R(h, a)$ be the expected reward of taking action a at history h. Let $end(h)$ be the latest abstract state of history h. It turns out that:

$$\Pr(h' \mid h, a) = \mathbf{1}[x' = end(h')] \Pr(x' \mid h, a), \tag{11}$$

and

$$R(h, a) = \mathbf{1}[x = end(h)] \sum_{s \in \varphi^{-1}(x)} R(s, a) \Pr(s \mid h). \tag{12}$$

Here, $\Pr(x' \mid h, a)$ is the probability of reaching abstract state x' after taking action a at history h, which can further be expressed as:

$$\Pr(x' \mid h, a) = \mathbf{1}[x = end(h)] \sum_{s \in \varphi^{-1}(x)} \Pr(s \mid h) \sum_{s' \in \varphi^{-1}(x')} T(s' \mid s, a). \tag{13}$$

On condition that, for any $s \in \varphi^{-1}(x)$, $\sum_{s' \in \varphi^{-1}(x')} T(s' \mid s, a) \equiv \mathrm{C_T}(x, a, x')$ and $R(s, a) \equiv \mathrm{C_R}(x, a)$ are constants, we have:

$$\Pr(h' \mid h, a) = \mathbf{1}[x = end(h)] \mathbf{1}[x' = end(h')] \mathrm{C_T}(x, a, x'), \tag{14}$$

and

$$R(h, a) = \mathbf{1}[x = end(h)] \mathrm{C_R}(x, a). \tag{15}$$

That is to say that the abstract model is indeed Markovian in the space of abstract states. Thus we can define an abstract MDP $M_\varphi = \langle X, A, T_\varphi, R_\varphi, \gamma \rangle$ with $T_\varphi(x' \mid x, a) = \mathrm{C_T}(x, a, x')$ and $R_\varphi(x, a) = \mathrm{C_R}(x, a)$. Solving this abstract MDP gives an optimal abstract policy $\pi^*_\varphi : X \to A$, which can be translated to a policy $\pi^* : S \to A$ for the ground MDP M by following $\pi^*(s) = \pi^*_\varphi(\varphi(s))$ for any $s \in S$. It has been shown that π^* preserves the optimality for M as well. This

condition is called bisimulation (or homomorphism) in the context of *model minimization* (Dean & Givan, 1997; Dearden & Boutilier, 1997; Ravindran & Barto, 2002, 2003a; Givan et al., 2003). A special case of bisimulation for a factored MDP (FMDP) is that there exist some irrelevant state variables that can be safely removed from the representation (Dietterich, 1999; Andre & Russell, 2002; Jong & Stone, 2005).

Bisimulation enables safe state abstraction for MDPs. However, safe state abstraction rarely exists, and is computationally difficult to find. Approximations have been made in the literature to relax the bisimulation condition. For example, several similarity metrics in state space are proposed to do approximate bisimulation abstraction (Ferns, Panangaden, & Precup, 2004; Ravindran & Barto, 2004; Taylor et al., 2009). Alternatively, Jiang et al. (2014) and Anand et al. (2015) evaluate the bisimulation condition in a sampled state space utilizing UCT. These methods enable lossy state abstraction, where the bisimulation condition is not completely satisfied. The resulting abstract model with lossy state abstraction is in general non-Markovian in the space of abstract states. In the spirit of having an abstract MDP $M_\varphi = \langle X, A, T_\varphi, R_\varphi, \gamma \rangle$, without computing the exact occupancy probability $\Pr(s \mid h)$, the best we can know about the abstract transition and reward functions — $T_\varphi(x' \mid x, a)$ and $R_\varphi(x, a)$ — is that they are constrained within certain ranges as $\Pr(s \mid h)$ is varying within its own range. Givan et al. (1997) introduce the notion of bounded-parameter Markov decision process (BMDP) to explicitly represent the abstract MDP with uncertain transition and reward functions resulting from doing state abstraction. An interval value iteration algorithm is developed to compute polices for a BMDP that are optimal in terms of so-called optimistic and pessimistic criteria. In contrast, the weighting function approach introduces a weighting function (or aggregation probability) $w : S \times X \to [0, 1]$ to approximate the occupancy probability $\Pr(s \mid h)$ by dropping its dependency on past history and treating it as a stationary probability distribution conditioned only on the latest abstract state, such that $\Pr(s \mid h) \approx w(s, end(h))$. Superficially, this ensures that the abstract transition and reward functions can be written in a Markovian way:

$$T_\varphi(x' \mid x, a) = \sum_{s \in \varphi^{-1}(x)} w(s, x) \sum_{s' \in \varphi^{-1}(x')} T(s' \mid s, a), \tag{16}$$

and

$$R_\varphi(x, a) = \sum_{s \in \varphi^{-1}(x)} R(s, a) w(s, x). \tag{17}$$

Therefore, an abstract MDP $M_\varphi = \langle X, A, T_\varphi, R_\varphi, \gamma \rangle$ can be defined following the weighting function assumption (Bertsekas, 1995; Li et al., 2006; Hostetler et al., 2014). We argue in Section 4.1 that such approaches cannot be accurate enough to capture the true dynamics of the abstract system, where the occupancy probability is in fact non-stationary, depending on the whole history of past actions and abstract states, or in other words, the policy being computed/executed!

2.4 Action Abstraction

State abstraction aggregates a set of a states into a macro state. Action abstraction (or *temporal abstraction*) extends the macro idea to closed-loop policies, or more precisely, closed-loop partial policies, by introducing macro actions (*temporally-extend actions* or *options*) composed from many concrete actions (Hauskrecht, Meuleau, Kaelbling, Dean, & Boutilier, 1998). Following the options theory of Sutton et al. (1999), an option takes a variable amount of time to complete.

Definition 4 (MDP Options)**.** *An option defined over an MDP $\langle S, A, T, R, \gamma \rangle$ is a tuple $\langle \mathcal{I}, \pi, \beta \rangle$, where $\mathcal{I} \subseteq S$ is the initiation set, meaning that this option is available in state s if and only if $s \in \mathcal{I}$, $\pi : S \to A$ is the inner policy, and $\beta : S \to [0, 1]$ is the termination condition with $\beta(s)$ being the probability of terminating in state s.*

As a special case, a primitive action can be seen as an option that terminates deterministically in one time step. It has been shown that an MDP equipped with a set of options turns out to be a *semi-Markov decision process* (SMDP), where an action following its own course of control can take multiple time steps to terminate (Duff & Bradtke Michael, 1995; Parr & Russell, 1998).

Definition 5 (Semi-Markov Decision Process)**.** *A semi-Markov decision process (SMDP) is a tuple $\langle S, \mathcal{O}, T, R, \gamma \rangle$, where S is the finite state space, $\mathcal{O}$ is the finite option space, $T : S \times \mathcal{O} \times S \times \mathbb{N} \to [0, 1]$ is the transition function with $T(s', N \mid s, o)$ being the probability of reaching state $s' \in S$, $N \in \mathbb{N}$ time steps after taking option $o \in \mathcal{O}$ in state $s \in S$, $R : S \times \mathcal{O} \to \mathbb{R}$ is the reward function with $R(s, o)$ being the expected reward collected by taking option $o \in \mathcal{O}$ in state $s \in S$, and $0 < \gamma \leq 1$ is a discount factor.*

The definitions of policy and value function extend naturally to SMDPs. In the research of reinforcement learning, action abstraction has been adopted under the name of *hierarchical reinforcement learning* (HRL) (Barto & Mahadevan, 2003). HRL aims to learn a policy for an MDP efficiently by exploiting the underlying hierarchical structure of the problem. In this context, the approaches of *hierarchical abstract machines* (HAMs) (Parr & Russell, 1998) and *ALisp* (Andre & Russell, 2002; Marthi, Russell, Latham, & Guestrin, 2005) write the control of an HRL agent as a partial program with some choice points left unspecified, and then find the optimal "completion" of that program. More precisely, these methods model the interaction between the partial program and its environment as a SMDP, and apply a Q-learning algorithm within the resulting SMDP to learn the optimal policy selecting actions for choice points. Alternatively, the theory of options defines abstract actions as closed-loop policies for taking actions over a period of time, and models the HRL agent within a SMDP formulation (Sutton et al., 1999; Stolle & Precup, 2002). The MAXQ approach decomposes the original MDP into a hierarchy of subtasks, where each subtask is considered as a SMDP with its low-level subtasks as actions, and learns these subtasks simultaneously utilizing a recursive way of value function decomposition (Dietterich, 2000; Diuk, Strehl, & Littman, 2006; Jong & Stone, 2008).

Action abstraction has also been applied to speed up planning for MDPs, by exploiting the underlying hierarchical structure of the problem at hand. For instance, Hauskrecht et al. (1998) develop an *abstract MDP* model that works with macro-actions and macro-states by treating macro-actions as local policies that act in certain regions of state space, and restricting states in the abstract MDP to those at the boundaries of regions. *Variable influence structure analysis* (VISA) (Jonsson & Barto, 2006) performs hierarchical decomposition for an MDP by building *dynamic Bayesian network* (DBN) models for actions, and constructing causal graphs that capture dynamics between state variables, under the assumption that a factored MDP model is available. Barry, Kaelbling, and Lozano-Perez (2011) propose an offline algorithm called DetH* to solve large MDPs hierarchically by assuming that the transitions between macro-states are deterministic. Bai, Wu, and Chen (2012, 2015) develop a hierarchical online planning algorithm (namely MAXQ-OP) for MDPs based on MAXQ value function decomposition, provided with estimations of termination distribution for high-level subtasks *a prior*. Vien and Toussaint (2014) develop a hierarchical MCTS algorithm,

that can learn the termination distributions of subtasks while computing their policies, following the MAXQ approach.

3. Related Work

Hostetler et al. (2014) analyzed state aggregation in MDPs following the weighting function approach. They established a performance loss bound in terms of a state abstraction error and a weighting function error. Our method has removed the weighting function error since the POMDP formulation guarantees that the optimal weighting function will always be used in the algorithm.

Vien and Toussaint (2014) developed a similar hierarchical MCTS framework for MDPs and POMDPs according to the theory of MAXQ value function decomposition. The MAXQ framework is not completely applicable to exploit the hierarchical structure induced by abstract states. A MAXQ subtask is specified with a termination predicate which partitions the (belief) state space into a set of active (belief) states and a set of terminal (belief) states (Dietterich, 2000). Using MAXQ subtasks to model temporal transitions between abstract states as abstract actions results inevitability in a set of overlapping subtasks, in which case we have also to introduce a pseudo-reward function for each subtask. Taking the rooms domain as an example, let $x \rightarrow y$ be the subtask of moving from room x to room y. If we treat histories ending with x as the set of active belief states, then histories not ending with x have to be the respective terminal belief states, in which case supposedly different abstract actions $A \rightarrow B$ and $A \rightarrow C$ actually have the same termination condition, which leads them to have the same learned policy. One way to avoid this problem is to introduce a pseudo-reward function within each subtask to encourage the subtask to move to the abstract goal state by additionally collecting a pseudo-reward, e.g. $r(h) = 1$ if $end(h) = y$, otherwise $r(h) = 0$. On the other hand, if we treat histories ending with y as the set of terminal belief states, then histories not ending with y have to be the respective active belief states, in which case abstract actions $A \rightarrow B$ and $D \rightarrow B$ have the same active belief states such that they are executable in the same set of belief states, which is not desirable either. The options theory used in this paper does not have this overlapping-subtask problem thanks to the concepts of termination condition and initiation set, which can be seen as an extension of the MAXQ termination predicate.

Bai et al. (2012, 2015) also developed a hierarchical online planning algorithm for MDPs (namely MAXQ-OP) based on MAXQ value function decomposition. Their method needs to estimate the termination distribution for each subtask in order to evaluate the completion function recursively in a dynamic programming way. In this paper, the high-level and low-level policies are learned simultaneously via a hierarchical MCTS approach, without the need to estimate the termination distributions. In the context of hierarchical reinforcement learning, Castro and Precup (2011) present a mechanism for automatically constructing options in a finite MDP by computing a bisimulation metric between states identifying abstract states. In this paper, we constructs abstract actions that connect input abstract states automatically. One future work might be to incorporate the approach of Castro and Precup into our framework so that there is no need to specify abstract states manually beforehand.

4. The Main Approach

In this section, we present our main approach to Markovian state and action abstractions for MDPs via hierarchical MCTS.

4.1 State Abstraction within a POMDP Formulation

As aforementioned, state abstraction generally results into a non-Markovian system in the abstracted state space. The safe state abstraction approach avoids the non-Markovian problem by ignoring only irrelevant state variables, or exploiting some particular structure of the ground MDP. The bounded MDP approach explicitly consider the uncertain transition and reward models resulting from doing state abstraction, but can only provide optimality guarantee in very limited sense. The weighting-function approach approximates the occupancy probability as a fixed weighting function by dropping its condition on the whole past history, however this approach can not capture the true dynamics of the abstract system, since the occupancy probability is non-stationary anyway! Our approach takes an alternative POMDP perspective. We treat abstract states as observations, and develop an observation function Ω according to the abstraction function φ such that $\Omega(x \mid s) = \mathbf{1}[x = \varphi(s)]$ for any $x \in X$ and $s \in S$. The abstract system turns out to be a POMDP $\langle S, A, X, T, R, \Omega, \gamma \rangle$ with X and Ω being the observation space and the observation function respectively. We denote the resulting POMDP by POMDP(M, φ) indicating that it is created by applying abstraction function φ on the ground MDP M. Within the resulting POMDP, the belief state $b(s \mid h)$ is exactly the occupancy probability $\Pr(s \mid h)$.

From a POMDP perspective, the weighting function approach actually tries to approximate the belief state $b(s \mid h)$ using a fixed distribution $w(s, x)$ with $end(h) = x$, and finds a memoryless policy π_φ as a mapping from observation space X to action space A. It has been shown that a memoryless policy for a POMDP can be arbitrarily worse than an optimal policy for the POMDP, which in turn can be arbitrarily worse than an optimal policy for the underlying MDP (Singh, Jaakkola, & Jordan, 1994). Thus, the weighting function approach is not well motivated from a POMDP point of view. In contrast, finding directly a near-optimal policy for POMDP(M, φ) could be a considerably better choice for planning with state abstraction over the ground MDP M. Additionally, bounded optimality results within the POMDP formulation can also be established given a bounded state abstraction.

Exactly solving POMDP(M, φ) via a dynamic programming method such as value iteration is usually intractable due to the continuous nature of the belief space. However, from an online planning point of view, it can be observed that the search tree in POMDP(M, φ) typically has a much lower branching factor than in the ground MDP M. This makes it feasible to use approximate, search-based solution techniques for solving POMDP(M, φ). More precisely, a search-based online planning algorithm running in M starting from state s_0 builds an expectimax tree $\mathcal{T}(s_0)$ with actions as the expectation nodes and states as the maximization nodes. The branching factor of $\mathcal{T}(s_0)$ is bounded by $|A|B$, where B is the maximal number of possible outcomes for any state–action pair. A similar expectimax tree $\mathcal{T}(b_0)$ with the same actions as the expectation nodes and belief states as the maximization nodes can be built by running a search-based online planning algorithm in POMDP(M, φ) starting from belief state b_0 with $b_0(s) = \mathbf{1}[s = s_0]$. The branching factor of $\mathcal{T}(b_0)$ is bounded by $|A|B'$ where B' is the maximal number of possible observations (i.e., abstract states) for any belief–action pair. Generally, we have $|A|B' < |A|B$, if $B' < B$ which holds for most abstractions due to the fact that $B \leq |S|$, $B' \leq |X|$ and $|X| \ll |S|$. Therefore, a search-based online planning algorithm running in POMDP(M, φ) resulting from abstraction could be much more efficient than running directly in the ground MDP M in terms of exploring the underlying expectimax search tree. It can also be observed that the observation function in the resulting POMDP is actually deterministic, which also makes it easy to be solved.

In this paper, we employ a POMCP algorithm running within POMDP(M, φ) to find an online policy for the ground MDP consistent with state abstraction φ, and refer to the resulting algorithm as POMCP$_\varphi$. The online policy produced by POMCP$_\varphi$ can be translated naturally to M, given the fact that when the agent is in ground state s, it can conclude that the respective belief state satisfies $b(s') = \mathbf{1}[s' = s]$ for any $s' \in S$. Since we are using a Monte Carlo algorithm to build the search tree, it is not necessary to have explicit representations of the underlying transition and reward functions for the ground MDP M. Only a simulator of M is needed. Another important advantage with POMCP$_\varphi$ is that it can be extended naturally to problems with continuous state spaces without significant modifications, given suitable abstraction functions defined over continuous states.

4.2 Automatic Action Abstraction in Belief Space

The proposed approach of state abstraction, we suggest, typically results in a search tree with a lower stochastic branching factor. The complementary approach of action abstraction can increase the effective search depth by considering high-level actions composed from many concrete actions. A given state abstraction naturally induces an action abstraction, where abstract actions connect abstract states in a one high-level step. We extend the options framework (Sutton et al., 1999) to a POMDP setting to model abstract actions within POMDP(M, φ) — the abstracted POMDP resulting from applying state abstraction φ over MDP M. Common results of options with respect to MDPs can be naturally extended to POMDPs as well.

Definition 6 (POMDP Options)**.** *Consider a POMDP $\langle S, A, Z, T, R, \Omega, \gamma \rangle$. Let $\mathcal{H}$ be the history space. An option is defined as a tuple $\langle \mathcal{I}, \pi, \beta \rangle$, where $\mathcal{I} \subseteq \mathcal{H}$ is the initiation set, meaning that this option is available at history h if and only if $h \in \mathcal{I}$, $\pi : \mathcal{H} \to A$ is the inner policy, and $\beta : \mathcal{H} \to [0, 1]$ is the termination condition with $\beta(h)$ being the probability of terminating this option at history h.*

Provided with the abstraction function φ, a temporally-extended transition (with possibly multiple time steps) in POMDP(M, φ) from a history ending with abstract state $x \in X$ to a history ending with one of x's neighbors $y \in X$ is considered a named option $o_{x \to y}$, represented as a tuple $\langle \mathcal{I}, \pi, \beta \rangle$. Here, $\mathcal{I}$ is the initiation set $\mathcal{I} = \{h \mid h \in \mathcal{H} \wedge end(h) = x\}$ indicating that $o_{x \to y}$ is executable only at a history ending with observation x, $\pi : \mathcal{H} \to A$ is an inner policy for $o_{x \to y}$ defined over $\mathcal{H}$ and β is a termination condition that $\beta(h) = 1$ if $end(h) = y$, and $\beta(h) = 0$ otherwise. Intuitively, $o_{x \to y}$ connects histories ending with abstract state x to histories ending with abstract state y. In the rooms domain, for example, $o_{A \to B}$ is the option moving the agent from room A to room B, which is executable only if the agent observes that it is in room A, and terminates when the agent observes that it is in room B. In fact, any two neighboring abstract states introduce automatically a high-level option that connects them. Let $\mathcal{O}$ be the set of options consisting of all possible direct transitions between histories ending with different abstract states. The set $\mathcal{O}$ can either be constructed manually by enumerating all the neighboring abstract states, or learned incrementally from an empty set by introducing a new option each time a new high-level transition has been observed. In this paper, we assume the former case for convenience. The main results can be easily extended to the latter case. Either way, it is not necessary to specify the local policy for each option beforehand. The proposed algorithm learns the high-level option-selection policy and the low-level, local option policies simultaneously via nested Monte Carlo planning.

The overall option-selection policy μ is defined as a mapping from histories to options $\mu : \mathcal{H} \to \mathcal{O}$. Let π_o be the local policy of option o. The hierarchical policy as a set of policies

$\Pi = \{\mu, \pi_{o_1}, \pi_{o_2}, \cdots\}$ represents a hierarchical solution for POMDP(M, φ), where μ corresponds to the root task and the π_os correspond to its subtasks. Given Π, in a *hierarchical control mode*, the agent selects an option $o = \mu(h_t)$ when initiated in a history h_t, and follows the option o according to π_o until it terminates in h_{t+k} ($k \geq 1$), at which point a new option $\mu(h_{t+k})$ is selected; in a *polling control mode*, the agent executes the action suggested by the current option $\mu(h_t)$ selected by μ at history h_t, regardless of which option is selected at the last time step. It has been shown that the polling execution of a hierarchical policy Π yields higher expected value than the hierarchical execution of the same hierarchical policy (Sutton et al., 1999; Dietterich, 2000). In this paper, we develop a hierarchical MCTS algorithm with state and action abstractions according to the value function decomposition as in the hierarchical control mode, but run the algorithm empirically as in the polling control mode.

In hierarchical control mode, let $V^\mu(h)$ be the value of following μ starting from history h, and let $Q^\mu(h, o)$ be the value of executing option o at history h and following μ thereafter. Let $|h|$ be the number of action–observation pairs of history h. It turns out that $V^\mu(h) = Q^\mu(h, \mu(h))$, and

$$Q^\mu(h, o) = V^{\pi_o}(h) + \sum_{h' \in \mathcal{H}} \gamma^{|h'|-|h|} \Pr(h' \mid h, o) V^\mu(h'), \tag{18}$$

where $\Pr(h' \mid h, o)$ — the termination distribution of option o — gives the probability that o terminates at history h' after $|h'| - |h|$ time steps. Here, $V^{\pi_o}(h)$ gives the value of following option o starting from history h, which can be further expressed as $V^{\pi_o}(h) = Q^{\pi_o}(h, \pi_o(h))$, where $Q^{\pi_o}(h, a)$ gives the value of executing the primitive action a at history h and following o thereafter, which is expressed as:

$$Q^{\pi_o}(h, a) = R^+(h, a) + \gamma \sum_{x \in X} \Pr(x \mid h, a) V^{\pi_o}(hax). \tag{19}$$

Combining policy evaluation with policy improvement, an *optimal hierarchical policy* $\Pi^* = \{\mu^*, \pi^*_{o_1}, \pi^*_{o_2}, \cdots\}$ can be computed in principle by iteratively applying $V^{\mu^*}(h) = \max_o Q^{\mu^*}(h, o)$ and $V^{\pi^*_o}(h) = \max_a Q^{\pi^*_o}(h, a)$. The problem here is that the termination distributions are unknown for the agent, because complete options with local policies are not assumed to be provided in advance, and estimating the termination distribution of an option implies that the local policy of the option is known. Instead, the agent has to find near-optimal policies for the root task and its subtasks simultaneously. We alleviate this problem by conducting a series of nested MCTS (namely POMCP) processes over the hierarchy. A high-level search tree for the root task is built by running MCTS with options as the macro actions. Each option builds its own sub-search tree via a nested MCTS process. In the search step, when simulating an option, its own sub-search tree is used to evaluate its Q-value. The leaf nodes of a sub-search tree serve as the next nodes of the high-level search tree. The simulation inside the sub-search tree is directed by its own tree and rollout policies, which guide the simulation in accomplishing the subtask corresponding to this option. It is also possible to design option-specific informative (rather than purely random) rollout policies for particular options, which is usually easier than designing an informative rollout policy for the ground MDP. In the back-propagation step, Q-values are updated according to Equations 18 and 19. This learning-by-simulation process ensures that the local policies of options as well as their termination distributions converge simultaneously. Given converged low-level policies and their termination distributions, the high-level policy converges in the limit as well. The resulting algorithm is denoted

Algorithm 1: The main algorithm

Agent (s_0 : *initial state,* φ : *abstraction function,* $\Pi_{rollout}$: *rollout policy*)
$h \leftarrow \varnothing$
$\mathcal{P}(h) \leftarrow \{s_0\}$
repeat
  $\mathcal{T} \leftarrow$ an empty search tree
  $a \leftarrow$ **OnlinePlanning** ($h, \mathcal{T}, \varphi, \Pi_{rollout}$)
  Execute a and observe abstract state x
  $h \leftarrow hax$
  $\mathcal{P}(h) \leftarrow$ **ParticleFilter** ($\mathcal{P}(h), a, x$)
until *termination conditions*;

OnlinePlanning (h : *history,* $\mathcal{T}$: *search tree,* φ : *abstraction function,* $\Pi_{rollout}$: *rollout policy*)
repeat
  $s \sim \mathcal{P}(h)$
  Search (*root task*, $s, h, 0, \mathcal{T}, \varphi, \Pi_{rollout}$)
until *resource budgets reached*;
return GetGreedyPrimitive (*root task*, h)

by HPOMCP$_\varphi$ indicating that it is a hierarchical MCTS algorithm running in POMDP(M, φ) resulting from doing state and action abstractions on the ground MDP M with φ as the state abstraction function and $\mathcal{O}$ as the set of abstract actions.

4.3 The Main Algorithm

The main algorithm HPOMCP$_\varphi$ is outlined in Algorithms 1 to 4. In the algorithm, the root task, its subtasks (modeled as options) and primitive actions are considered uniformly as *tasks*. For example, a task as a parameter of the **Search** function could be either the root task, or one of the options, or one of the primitive actions. The algorithm builds a search tree for each task, up to the maximal planning horizon H, in the space of histories consistent with the hierarchy defined by the input abstraction function φ. The belief state corresponding to a history h is represented as a set of particles, denoted by $\mathcal{P}(h)$, which are updated using a particle filter. In the implementation, we do not need to explicitly maintain h as a full sequence of actions and observations. Instead, we use a hash value of h as an index, and update it incrementally, i.e. $hash(haz) = hash\_combine(hash(h), a, z)$.

We now describe the subroutines in more detail. The **Agent** function is the overall procedure interacting with the environment in a polling control mode. It calls **OnlinePlanning** to select a primitive action, executes it, observes the resulting abstract state, and updates particles repeatedly. The **OnlinePlanning** function runs in an anytime fashion. At each iteration, it samples a state s from the belief state represented as a set of particles corresponding to h, and invokes a hierarchical search process for the root task from history h and sampled state s by calling **Search**. It finally returns a greedy primitive action according to the current search tree and action values. It is worth noting that in practice the algorithm can also take advantage of the fact that at the root node of the search tree the agent actually has access to the true ground state, in which case the set of particles at the root node contains only one single state. This does not change the fact that the algorithm is

Algorithm 2: The tree search algorithm

Search ($t : task$, $s : state$, $h : history$, $d : depth$, $\mathcal{T} : search\ tree$, $\varphi : abstraction\ function$, $\Pi_{rollout} : rollout\ policy$)
if t *is primitive* **then**
  $\langle s', r\rangle \sim$ **Simulate** (s, t)
  $x \leftarrow \varphi(s')$
  return $\langle r, 1, htx, s'\rangle$
else
  if $d \geq H$ **or** t *terminates at* h **then**
    return $\langle 0, 0, h, s\rangle$
  else
    if *node* $\langle t, h\rangle$ *is not in tree* $\mathcal{T}$ **then**
      Insert node $\langle t, h\rangle$ to $\mathcal{T}$
      return **Rollout** $(t, s, h, d, \varphi, \Pi_{rollout})$
    else
      $a^* \leftarrow \text{argmax}_a \left\{ Q[t, h, a] + c\sqrt{\frac{\log N[t,h]}{N[t,h,a]}} \right\}$
      $\langle r', n', h', s'\rangle \leftarrow$ **Search** $(a^*, s, h, d, \mathcal{T}, \varphi, \Pi_{rollout})$
      $\langle r'', n'', h'', s''\rangle \leftarrow$ **Search** $(t, s', h', d + n', \mathcal{T}, \varphi, \Pi_{rollout})$
      $N[t, h] \leftarrow N[t, h] + 1$
      $N[t, h, a^*] \leftarrow N[t, h, a^*] + 1$
      $r \leftarrow r' + \gamma^{n'} r''$
      $Q[t, h, a^*] \leftarrow Q[t, h, a^*] + \frac{r - Q[t,h,a^*]}{N[t,h,a^*]}$
      return $\langle r, n' + n'', h'', s''\rangle$

Algorithm 3: The pollling rollout algorithm

Rollout ($t : task$, $s : state$, $h : history$, $d : depth$, $\varphi : abstraction\ function$, $\Pi_{rollout} : rollout\ policy$)
if $d \geq H$ **or** t *terminates at* h **then**
  return $\langle 0, 0, h, s\rangle$
else
  $a \leftarrow$ **GetPrimitive** $(\Pi_{rollout}, t, h)$
  $\langle s', r'\rangle \leftarrow$ **Simulate** (s, a)
  $x \leftarrow \varphi(s')$
  $\langle r'', n, h'', s''\rangle \leftarrow$ **Rollout** $(t, s', hax, d + 1, \varphi, \Pi_{rollout})$
  $r \leftarrow r' + \gamma r''$
  return $\langle r, n + 1, h'', s''\rangle$

searching in the space of belief states due to the way the tree is expanded and the value functions are updated.

The **Search** function builds a search tree for each task in the space of histories constrained by the hierarchy. For a primitive action, it simply runs a one-step simulation and returns the result

Algorithm 4: The primitive-action selection algorithms

```
GetGreedyPrimitive (t : task, h : history)
if t is primitive then
    return t
else
    a* ← argmax_a Q[t, h, a]
    return GetGreedyPrimitive (a*, h)

GetPrimitive (Π : policy, t : task, h : history)
if t is primitive then
    return t
else
    return GetPrimitive (Π, π_t(h), h)
```

encoded in a tuple. For a non-primitive action t which could either be the root task or one of its subtasks, it returns the result returned by a **Rollout** subroutine if the node $\langle t, h\rangle$ is not already in the tree, otherwise it 1) selects a greedy (macro) action a^* for task t according to the UCB action-selection heuristic, 2) invokes a nested search process for the selected (macro) action a^* with the same state s, history h and depth d as the input parameters, 3) recursively explores the current search tree starting from the history h' and state s' returned by the search of a^*, and 4) updates estimated Q-values according to Equation 18 for the root task or Equation 19 for an option. More precisely, $Q[\text{root task}, h, a] \approx Q^{\mu}(h, a)$, where μ is the high-level policy represented by the high-level search tree, and a is an option executable at history h; and $Q[t, h, a] \approx Q^{\pi_t}(h, a)$, where t is an option, π_t is the local policy represented by the nested search tree of t, and a is a primitive action.

The **Rollout** function uses a hierarchical rollout policy $\Pi_{rollout}$ to run a sequence of Monte Carlo simulations in a polling control mode according to the generative model of the ground MDP that is encoded in the function **Simulate**. It returns a tuple $\langle r, n, h, s\rangle$ where r is the sum of sampled rewards, n is the total number of steps, h is the resulting history and s is the final state which follows the belief state corresponding to h. In the algorithm, $\Pi_{rollout}$ is assumed to be a hierarchical policy defined over the history space. The **GetPrimitive** function returns a primitive action at history node h for task t suggested by a hierarchical policy Π. It recursively finds the right action suggested by the current task t according to $\pi_t \in \Pi$ until it reaches a primitive action. In particular, the **GetGreedyPrimitive** function returns the greedy action at history h for task t suggested by the hierarchical policy represented by the current search tree and action values.

5. Theoretical Results

In this section, we present our main theoretical results in terms of performance loss, optimality guarantee and convergence.

5.1 Optimality Results with State Abstraction

Theorem 1. *POMCP$_\varphi$ finds the optimal policy consistent with abstraction φ.*

Proof. This follows directly from the optimality results of POMCP which states that POMCP finds the optimal policy for the underlying POMDP (Silver & Veness, 2010). Applying state abstraction φ

on a ground MDP M results into a POMDP POMDP(M, φ) with the observations being the abstract states. POMCP$_\varphi$ further finds the optimal policy of this POMDP. Thus, POMCP$_\varphi$ finds the optimal policy consistent with the input state abstraction φ. □

Inspired by the abstraction criteria introduced in (Hostetler et al., 2014), we define aggregation error as follows.

Definition 7 (Aggregation Error)**.** *The aggregation error of state abstraction $\langle X, \varphi\rangle$ for a ground MDP $M = \langle S, A, T, R, \gamma\rangle$ is e, if $\exists \mathring{a} \in A$, such that for all $x \in X$, $\varphi(s) = x$ and $d \in [0, H]$, $|V_d(s) - Q_d(s, \mathring{a}) \ | \leq e$, where V_d and Q_d are the optimal value and action-value functions at depth d in the search tree of M, and H is the maximal planning horizon.*

A bounded aggregation error requires that the action-value of $\mathring{a}$ is close to the optimal value for all ground states within the same abstract state x. Particularly, $e = 0$ implies that all ground states within the same abstract state share the same optimal action. Computing exactly the aggregation error implies solving the entire ground MDP completely which is usually infeasible, but this doesn't change the fact that such aggregation error exists and measures the approximation error introduced by grouping states that have different optimal actions when doing state abstraction.

Theorem 2. *For state abstraction $\langle X, \varphi\rangle$ for a ground MDP $M = \langle S, A, T, R, \gamma\rangle$ with aggregation error e, let s_0 be the current state in the ground MDP M and let h_0 with $\mathcal{P}(h_0) = \{s_0\}$ be the corresponding history in POMDP(M, φ). Let $Q^*(s, \cdot)$ and $Q^*(h, \cdot)$ be the optimal action values of M and POMDP(M, φ) respectively. Let $a^* = \operatorname{argmax}_{a\in A} Q^*(h_0, a)$ be the optimal primitive action found in POMDP(M, φ) at history h_0, and define an action-value error as $E(a^*) = |\max_{a\in A} Q^*(s_0, a) - Q^*(s_0, a^*)|$. Suppose the maximal planning horizon is H, then $E(a^*)$ is bounded by $E(a^*) \leq 2He$ if $\gamma = 1$, else $E(a^*) \leq 2\gamma\frac{1-\gamma^H}{1-\gamma}e$.*

Proof. Consider the search trees of M and POMDP(M, φ). We define a specific action-value error for history h and action a at depth d in the search tree of POMDP(M, φ) to be:

$$E_d(h, a) = \left| Q_d(h, a) - \sum_{s\in S} b(s \mid h) Q_d(s, a) \right|, \tag{20}$$

where $Q_d(s, \cdot)$ and $Q_d(h, \cdot)$ are optimal action values at depth d in the search trees of M and POMDP(M, φ) respectively. By applying Bellman equations, we have

$$\begin{aligned} E_d(h, a) &= \gamma \left| \sum_{h'\in\mathcal{H}} \Pr(h' \mid h, a) V_{d+1}(h') \right. \\ &\qquad \left. - \sum_{s\in S} b(s \mid h) \sum_{s'\in S} T(s' \mid s, a) V_{d+1}(s') \right| \\ &= \gamma \left| \sum_{h'\in\mathcal{H}} \Pr(h' \mid h, a) V_{d+1}(h') \right. \\ &\qquad \left. - \sum_{s'\in S} V_{d+1}(s') \sum_{s\in S} b(s \mid h) T(s' \mid s, a) \right|. \end{aligned} \tag{21}$$

Noticing that

$$\begin{aligned}\Pr(s' \mid h, a) &= \sum_{s \in S} T(s' \mid s, a) b(s \mid h) \\ &= \sum_{h' \in \mathcal{H}} b(s' \mid h') \Pr(h' \mid h, a), \end{aligned} \tag{22}$$

it follows that

$$\begin{aligned} E_d(h,a) = \gamma \Bigg| & \sum_{h' \in \mathcal{H}} \Pr(h' \mid h, a) V_{d+1}(h') \\ & - \sum_{h' \in \mathcal{H}} \Pr(h' \mid h, a) \sum_{s' \in S} b(s' \mid h') V_{d+1}(s') \Bigg| \\ \leq \gamma \sum_{h' \in \mathcal{H}} \Pr(h' \mid h, a) & \Bigg| V_{d+1}(h') \\ & - \sum_{s' \in S} b(s' \mid h') V_{d+1}(s') \Bigg|, \end{aligned} \tag{23}$$

by applying the triangle inequality. On the other hand, since

$$\left| V_d(h) - \max_{a \in A} \sum_{s \in S} b(s \mid h) Q_d(s,a) \right| \leq \max_{a \in A} E_d(h,a), \tag{24}$$

and

$$\begin{aligned} & \left| \max_{a \in A} \sum_{s \in S} b(s \mid h) Q_d(s,a) - \sum_{s \in S} b(s \mid h) V_d(s) \right| \\ & \leq \sum_{s \in S} b(s \mid h) \left| Q_d(s, \mathring{a}) - V_d(s) \right| \leq e, \end{aligned} \tag{25}$$

we have

$$\left| V_d(h) - \sum_{s \in S} b(s \mid h) V_d(s) \right| \leq \max_{a \in A} E_d(h,a) + e. \tag{26}$$

Therefore,

$$\begin{aligned} E_d(h,a) &\leq \gamma \left[\sum_{h' \in \mathcal{H}} \Pr(h' \mid h, a) \max_{a' \in A} E_{d+1}(h', a') + e \right] \\ &\leq \gamma \left[\max_{h' \in \mathcal{H}, a' \in A} E_{d+1}(h', a') + e \right], \end{aligned} \tag{27}$$

for all $h \in \mathcal{H}$ and $a \in A$. Let $E(d) = \max_{h \in \mathcal{H}, a \in A} E_d(h,a)$. We get $E(d) \leq \gamma[E(d+1) + e]$. At terminal nodes, both $Q_H(h, \cdot)$ and $Q_H(s, \cdot)$ equal 0, so E(H) = 0. Therefore, $E(d) \leq (H-d)e$ if $\gamma = 1$, otherwise $E(d) \leq \gamma \frac{1-\gamma^{H-d}}{1-\gamma} e$. At the root node, let $a^* = \operatorname{argmax}_a Q_0(h_0, a)$, and $b^* = \operatorname{argmax}_{a \in A} Q_0(s_0, a^*)$. If $a^* = b^*$, $E(a^*) = 0$; otherwise, we must have $Q_0(h_0, a^*) \geq Q_0(h_0, b^*)$. Noticing that $\mathcal{P}(h_0) = \{s_0\}$, since $|Q_0(s_0, b^*) - Q_0(h_0, b^*)| \leq E(0)$ and $|Q_0(s_0, a^*) - Q_0(h_0, a^*)| \leq E(0)$, we get $E(a^*) = |Q_0(s_0, b^*) - Q_0(s_0, a^*)| \leq 2E(0)$. □

5.2 Convergence Results with Action Abstraction

Theorem 3. *With probability 1, HPOMCP$_\varphi$ converges to a recursively optimal hierarchical policy for POMDP(M, φ) over the hierarchy defined by the input state and action abstractions.*

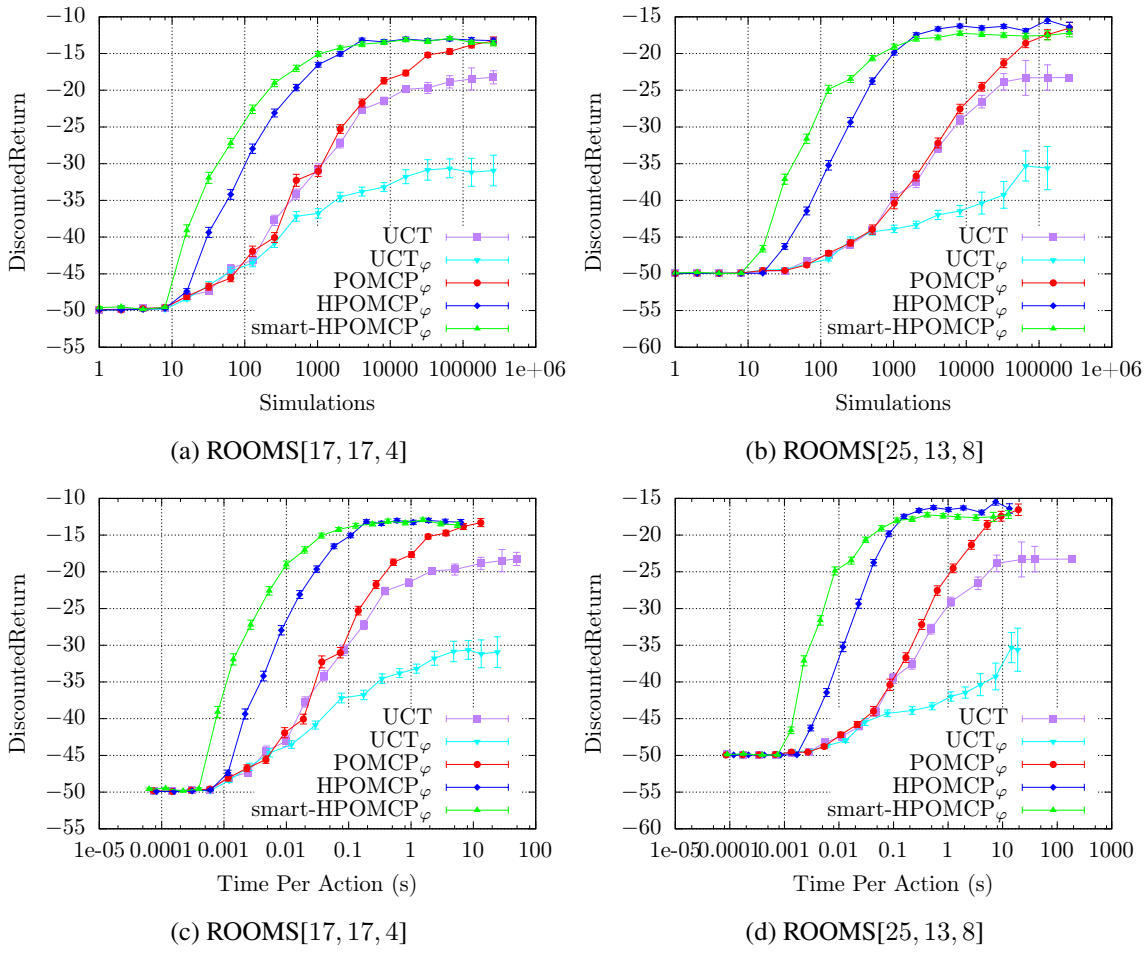
Proof. For a particular option that connects two abstract states, HPOMCP$_\varphi$ finds the optimal policy with probability 1 following the convergence results of POMCP with sufficient explorations in the limit. Given the fact that, when options converge, their termination distributions also converge, the root task reduces to a stationary semi-Markov Decision Process (SMDP) defined over the belief space. HPOMCP$_\varphi$ then finds the optimal policy in the limit for the root task over the converged SMDP by extending the convergence results of POMCP to SMDPs over belief space. Since the high-level policy is optimal given optimal low-level policies, HPOMCP$_\varphi$ converges to a recursively optimal hierarchical policy. □

6. Experiments

This section presents the experiments conducted on the rooms domain and an extended continuous-rooms domain comparing with several baseline algorithms.

6.1 Rooms Domain

The ROOMS[m, n, k] problem simulates a robot navigating in a $m \times n$ grid map containing k rooms, as depicted in Figure 1a. The 8 primitive actions are E, SE, S, SW, W, NW, N and NE. Each action has a probability 0.2 of mistake, in which case a random action is executed instead. If any movement is going to collide with the wall, the agent stays at its current position. Each primitive action has a reward of -1. Moving into the goal has a reward of 10. In the ground MDP, the stochastic branching factor for an action is up to 8. With state abstraction by assuming a room as an abstract state, the stochastic branching factor for an action reduces to 2. An option is defined as a transition from a room to one of its neighbors. The discount factor is $\gamma = 0.98$. The maximal planning horizon is determined as $H = \lfloor \log_\gamma \epsilon \rfloor = 341$, where ϵ is set to be 0.001. Figure 4 with x axis in log scale shows the results of running UCT, UCT$_\varphi$, POMCP$_\varphi$, HPOMCP$_\varphi$ and smart-HPOMCP$_\varphi$ in ROOMS[$17, 17, 4$] and ROOMS[$25, 13, 8$] problems, where UCT runs directly in the ground state space, UCT$_\varphi$ is a UCT algorithm running entirely in the abstract state space, POMCP$_\varphi$ is a POMCP algorithm running on POMDP(M, φ) resulting from doing state abstraction on the ground MDP M, HPOMCP$_\varphi$ is the proposed hierarchical MCTS algorithm running on POMDP(M, φ), and smart-HPOMCP$_\varphi$ is a HPOMCP$_\varphi$ algorithm equipped with hand-coded informative rollout policies for options. UCT, UCT$_\varphi$, POMCP$_\varphi$ and HPOMCP$_\varphi$ are all developed with purely random rollout policies. The performance is evaluated using averaged discounted return in terms of the number of simulations and the averaged computation time per action. Each data point is averaged over 100 runs (or up to 2 hours of total computation time). It can be seen from the results that POMCP$_\varphi$ outperforms UCT or has at least the same performance, indicating that modeling a ground MDP with state abstraction as a POMDP and solving the POMDP via approximated, search-base online planning algorithms is feasible. UCT$_\varphi$ uses the empirical distributions of $\Pr(s \mid x)$ to approximate $w(s, x)$ and finds a memoryless policy as a mapping from abstract states to actions following the weighting function approach. It has easily the worst performance in all cases, confirming that finding memoryless policies might not be the right way to do state abstractions since too much information on the abstract level is ignored. The main algorithm — HPOMCP$_\varphi$ — outperforms



(a) ROOMS[17, 17, 4]

(b) ROOMS[25, 13, 8]

(c) ROOMS[17, 17, 4]

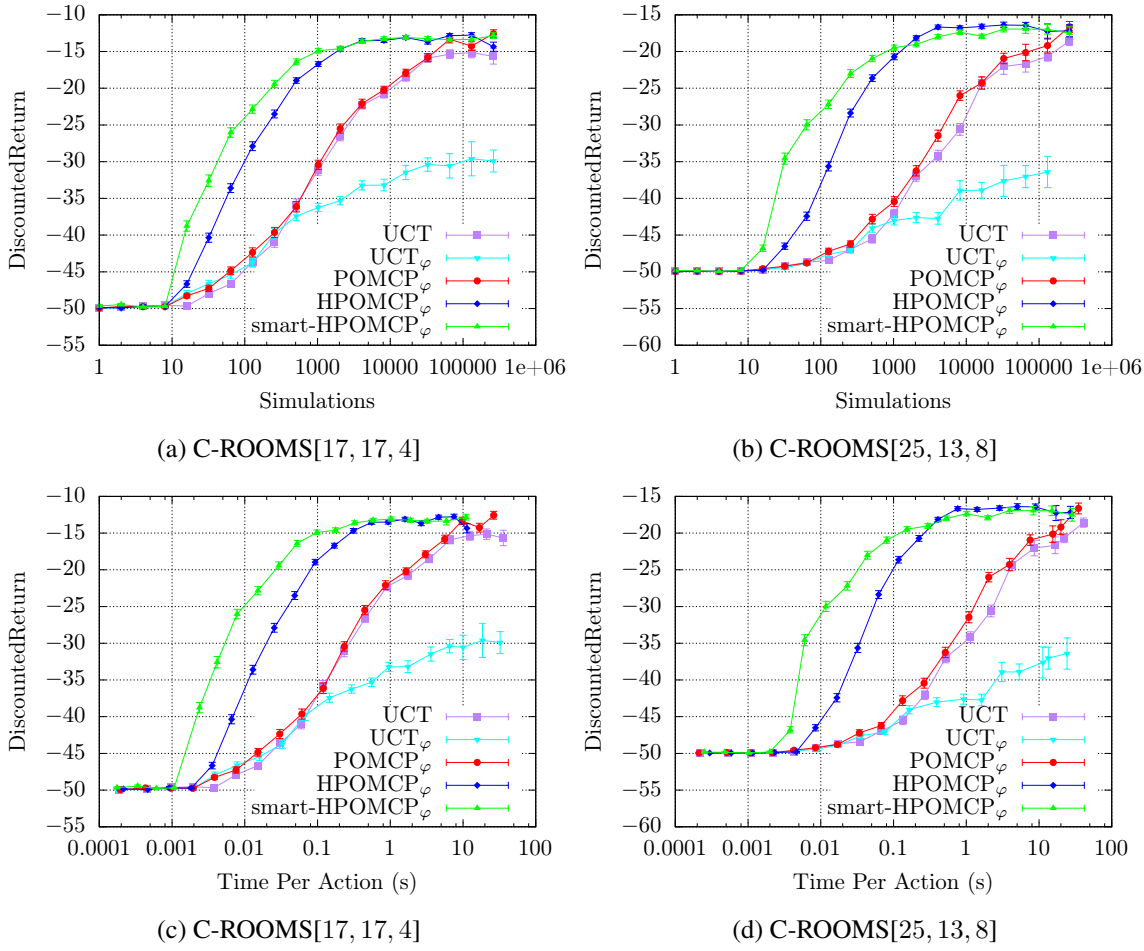
(d) ROOMS[25, 13, 8]

Figure 4: Empirical results on the rooms domains.

UCT by orders of magnitude. HPOMCP$_\varphi$ also outperforms POMCP$_\varphi$ substantially suggesting that exploiting the hierarchical structure introduced by doing state abstraction contributes the main improvement. With the help of an option-specific rollout policy which is designed to near-optimally move to the intersection area of two connected rooms, smart-HPOMCP$_\varphi$ improves on HPOMCP$_\varphi$ significantly. The possibility of introducing option-specific rollout policies can also be considered as an advantage of HPOMCP$_\varphi$.

6.2 Continuous Rooms Domain

We further extend ROOMS$[m, n, k]$ into a continuous state space and propose a C-ROOMS$[m, n, k]$ problem, where each cell has a size of 1 (m^2). The position of the agent is represented using continuous (x, y) coordinates. A primitive action moves the agent by a distance of 1 (m) in expectation, augmented with a Gaussian error. The agent finishes this task if the distance to the goal is within 0.5 (m). UCT in such continuous domains reduces to a depth-1 search which can be seen as a single step of policy improvement over the rollout policy. To run POMCP$_\varphi$ and HPOMCP$_\varphi$ algorithms in



(a) C-ROOMS[17, 17, 4]

(b) C-ROOMS[25, 13, 8]

(c) C-ROOMS[17, 17, 4]

(d) C-ROOMS[25, 13, 8]

Figure 5: Empirical results on the continuous rooms domains.

this domain, we need only provide the appropriate observation function defined over the continuous state space. Figure 5 shows the results of running UCT, UCT$_\varphi$, POMCP$_\varphi$, HPOMCP$_\varphi$ and smart-HPOMCP$_\varphi$ in C-ROOMS[17, 17, 4] and C-ROOMS[25, 13, 8] problems, confirming that POMCP$_\varphi$ and HPOMCP$_\varphi$ algorithms have the ability to run in continuous domains without significant modifications. Similar trends can be seen in the results. It is also interesting to see that although UCT has reduced to a depth-1 search in this continuous domain, it still has rather good performance. This might be because the value function of a random policy in this domain provides a good heuristic, thus a greedy policy over this heuristic can work well.

7. Conclusion

In this paper, we propose that state- and action-abstracted MDPs can be viewed as POMDPs. We bound the performance loss induced by the abstraction and we develop a hierarchical MCTS algorithm with automatically constructed abstract actions for approximately solving the abstract

POMDP. The algorithm converges to a recursively optimal hierarchical policy for the ground MDP consistent with the input state abstractions. Empirical results show that the proposed approach improves ground MCTS by orders of magnitude on benchmark domains. In future work, we plan to extend this approach to general reinforcement learning algorithms with features (such as those introduced by various state-space function approximators). The non-Markovianess introduced by features can be overcome by using a POMDP formulation; the hierarchical structure in the feature space can be exploited by using a similar hierarchical MCTS approach as shown in this paper.

Acknowledgments

Funding for this research was provided by ONR under contract N00014-12-1-0609, by DARPA under contract N66001-15-2-4048, and by the United Technologies Research Center. Opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the funding agencies.

References

Anand, A., Grover, A., Mausam, M., & Singla, P. (2015). ASAP-UCT: abstraction of state-action pairs in UCT. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pp. 1509–1515.

Andre, D., & Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. In *Proceedings of the 8th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence*, pp. 119–125.

Asmuth, J., & Littman, M. L. (2011). Learning is planning: near Bayes-optimal reinforcement learning via Monte-Carlo tree search. In *Uncertainty in Artificial Intelligence*, pp. 19–26.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, *47*(2), 235–256.

Auer, P. (2003). Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, *3*, 397–422.

Bai, A., Srivastava, S., & Russell, S. (2016). Markovian state and action abstractions for MDPs via hierarchical MCTS. In *25th International Joint Conference on Artificial Intelligence (IJCAI)*, New York, United States.

Bai, A., Wu, F., & Chen, X. (2012). Online planning for large MDPs with MAXQ decomposition (extended abstract). In *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012)*.

Bai, A., Wu, F., & Chen, X. (2015). Online planning for large Markov decision processes with hierarchical decomposition. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *6*(4), 45.

Barrett, S., Agmon, N., Hazon, N., Kraus, S., & Stone, P. (2014). Communicating with unknown teammates. In *Proc. of 13th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012)*.

Barrett, S., Stone, P., Kraus, S., & Rosenfeld, A. (2013). Teamwork with limited knowledge of teammates. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Barry, J., Kaelbling, L., & Lozano-Perez, T. (2011). Deth*: Approximate hierarchical solution of large Markov decision processes. In *International Joint Conference on Artificial Intelligence*, pp. 1928–1935.

Barto, A., Bradtke, S., & Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*(1-2), 81–138.

Barto, A., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, *13*(4), 341–379.

Bellman, R. (1957). *Dynamic Programming* (1 edition). Princeton University Press, Princeton, NJ, USA.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, Vol. 1. Athena Scientific Belmont, MA.

Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012a). A survey of Monte Carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, *4*(1), 1–43.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012b). A survey of Monte Carlo tree search methods. *IEEE Trans. on Computational Intelligence and AI in Games*, *4*, 1–43.

Castro, P. S., & Precup, D. (2011). Automatic construction of temporally extended actions for mdps using bisimulation metrics. In *Recent Advances in Reinforcement Learning*, pp. 140–152. Springer.

Chaslot, G., Bakkes, S., Szita, I., & Spronck, P. (2008). Monte-carlo tree search: A new framework for game AI. In Darken, C., & Mateas, M. (Eds.), *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, October 22-24, 2008, Stanford, California, USA*. The AAAI Press.

Coquelin, P.-A., & Munos, R. (2007). Bandit algorithms for tree search. In *Uncertainty in Artificial Intelligence*.

Dean, T., & Givan, R. (1997). Model minimization in markov decision processes. In *AAAI/IAAI*, pp. 106–111.

Dean, T., Givan, R., & Leach, S. (1997). Model reduction techniques for computing approximately optimal solutions for markov decision processes. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pp. 124–131. Morgan Kaufmann Publishers Inc.

Dearden, R., & Boutilier, C. (1997). Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, *89*(1), 219–283.

Dietterich, T. G. (1999). State abstraction in MAXQ hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 994–1000.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal Artificial Intelligence Research(JAIR)*, *13*, 227–303.

Diuk, C., Strehl, A. L., & Littman, M. L. (2006). A hierarchical approach to efficient reinforcement learning in deterministic domains. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 313–319. ACM.

Duff, S. J., & Bradtke Michael, O. (1995). Reinforcement learning methods for continuous-time markov decision problems. *Adv Neural Inf Process Syst*, *7*, 393.

Ferns, N., Panangaden, P., & Precup, D. (2004). Metrics for finite markov decision processes. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pp. 162–169. AUAI Press.

Ferns, N., Castro, P. S., Precup, D., & Panangaden, P. (2012). Methods for computing state similarity in markov decision processes. *arXiv preprint arXiv:1206.6836*.

Finnsson, H., & Björnsson, Y. (2008). Simulation-based approach to general game playing.. In *AAAI*, Vol. 8, pp. 259–264.

Gelly, S., & Silver, D. (2011). Monte-Carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, *175*(11), 1856–1875.

Gelly, S., & Silver, D. (2007). Combining online and offline knowledge in uct. In *Proceedings of the 24th international conference on Machine learning*, pp. 273–280. ACM.

Giunchiglia, F., & Walsh, T. (1992). A theory of abstraction. *Artificial Intelligence*, *57*(2), 323–389.

Givan, R., Dean, T., & Greig, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, *147*(1), 163–223.

Givan, R., Leach, S., & Dean, T. (1997). Bounded parameter markov decision processes. In *European Conference on Planning*, pp. 234–246. Springer.

Gordon, N. J., Salmond, D. J., & Smith, A. F. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, Vol. 140, pp. 107–113. IET.

Guez, A., Silver, D., & Dayan, P. (2012). Efficient Bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Information Processing Systems*, pp. 1034–1042.

Hansen, E., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*(1-2), 35–62.

Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pp. 220–229. Morgan Kaufmann Publishers Inc.

Hengst, B. (2007). Safe state abstraction and reusable continuing subtasks in hierarchical reinforcement learning. *AI 2007: Advances in Artificial Intelligence*, 58–67.

Hostetler, J., Fern, A., & Dietterich, T. (2014). State aggregation in Monte Carlo tree search. In *Proceedings of 28th AAAI Conference on Artificial Intelligence*.

Jaakkola, T., Singh, S. P., & Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In Tesauro, G., Touretzky, D., & Leen, T. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 7, pp. 345–352. The MIT Press.

Jiang, N., Singh, S., & Lewis, R. (2014). Improving UCT planning via approximate homomorphisms. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, pp. 1289–1296.

Jong, N. K., & Stone, P. (2005). State abstraction discovery from irrelevant state variables. In Kaelbling, L. P., & Saffiotti, A. (Eds.), *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pp. 752–757. Professional Book Center.

Jong, N., & Stone, P. (2008). Hierarchical model-based reinforcement learning: R-max + MAXQ. In *Proceedings of the 25th international conference on Machine learning*, pp. 432–439. ACM.

Jonsson, A., & Barto, A. (2006). Causal graph based decomposition of factored mdps. *The Journal of Machine Learning Research*, *7*, 2259–2301.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1-2), 99–134.

Kearns, M., Mansour, Y., & Ng, A. (1999). A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, pp. 1324–1331. Morgan Kaufmann Publishers Inc.

Keller, T., & Eyerich, P. (2012). Prost: Probabilistic planning based on UCT. In *ICAPS12*.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pp. 282–293.

Konidaris, G. (2016). Constructing abstraction hierarchies using a skill-symbol loop. *The 25th International Joint Conference on Artificial Intelligence (IJCAI)*.

Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs.. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*.

Littman, M., Dean, T., & Kaelbling, L. (1995). On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 394–402. Citeseer.

Macindoe, O., Kaelbling, L. P., & Lozano-Pérez, T. (2012). POMCoP: Belief space planning for sidekicks in cooperative games. In Riedl, M., & Sukthankar, G. (Eds.), *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-12, Stanford, California, October 8-12, 2012*. The AAAI Press.

Marthi, B., Russell, S. J., Latham, D., & Guestrin, C. (2005). Concurrent hierarchical reinforcement learning. In *IJCAI*, pp. 779–785.

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, Vol. 10.

Pineau, J., Gordon, G., & Thrun, S. (2003). Policy-contingent abstraction for robust robot control. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 477 – 484.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Ravindran, B., & Barto, A. G. (2002). Model minimization in hierarchical reinforcement learning. In *Abstraction, Reformulation, and Approximation*, pp. 196–211. Springer.

Ravindran, B., & Barto, A. G. (2003a). Relativized options: Choosing the right transformation. In *ICML*, pp. 608–615.

Ravindran, B., & Barto, A. G. (2003b). SMDP homomorphisms: An algebraic approach to abstraction in semi-markov decision processes. In Gottlob, G., & Walsh, T. (Eds.), *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pp. 1011–1018. Morgan Kaufmann.

Ravindran, B., & Barto, A. G. (2004). Approximate homomorphisms: A framework for non-exact minimization in markov decision processes. In *In Proceedings of the International Conference on Knowledge Based Computer Systems.*

Russell, S. J., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2 edition). Pearson Education.

Silver, D., & Veness, J. (2010). Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, pp. 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.

Singh, S. P., Jaakkola, T., & Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the 11th International Conference on Machine Learning*, pp. 284–292.

Singh, S. P., Jaakkola, T., & Jordan, M. I. (1995). Reinforcement learning with soft state aggregation. *Advances in Neural Information Processing Systems*, 361–368.

Srivastava, S., Russell, S. J., & Pinto, A. (2016). Metaphysics of planning domain descriptions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pp. 1074–1080.

Stolle, M., & Precup, D. (2002). Learning options in reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, pp. 212–223. Springer.

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*(1), 181–211.

Taylor, J., Precup, D., & Panagaden, P. (2009). Bounding performance loss in approximate mdp homomorphisms. In Koller, D., Schuurmans, D., Bengio, Y., & Bottou, L. (Eds.), *Advances in Neural Information Processing Systems 21*, pp. 1649–1656. Curran Associates, Inc.

Thrun, S. (1999). Monte carlo pomdps.. In *NIPS*, Vol. 12, pp. 1064–1070.

Vien, N. A., Ertel, W., Dang, V.-H., & Chung, T. (2013). Monte-carlo tree search for bayesian reinforcement learning. *Applied intelligence*, *39*(2), 345–353.

Vien, N. A., & Toussaint, M. (2014). Hierarchical Monte-Carlo planning. In *Proceedings of the 29th AAAI Conference on Artificial*.

Winands, M. H., Bjornsson, Y., & Saito, J. (2010). Monte Carlo tree search in lines of action. *IEEE Transactions on Computational Intelligence and AI in Games*, *2*(4), 239–250.

Wolfe, A. P., & Barto, A. G. (2006). Decision tree methods for finding reusable mdp homomorphisms. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, Vol. 21, p. 530. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.