Markovian State and Action Abstractions for MDPs via Hierarchical MCTS

Aijun Bai
UC Berkeley
aijunbai@berkeley.edu

Siddharth Srivastava
United Tech. Research Center
srivass@utrc.utc.com

Stuart Russell
UC Berkeley
russell@cs.berkeley.edu

Abstract

State abstraction is an important technique for scaling MDP algorithms. 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 POMDP and show that we can thereby take advantage of state abstraction without sacrificing the Markov property. We further 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 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

Markov decision processes (MDPs) provide a rich framework for planning and learning under uncertainty. In this paper, we focus on the problem of online planning in MDPs. An online planning algorithm finds the best action for the current state by exploring an expectimax search tree starting from the current state [Barto *et al.*, 1995; Hansen and Zilberstein, 2001; Kocsis and Szepesvári, 2006]. Take *Monte Carlo tree search* (MCTS) [Browne *et al.*, 2012] as an example. It has been observed that the performance of MCTS is typically dominated by the effective search depth [Kearns *et al.*, 2002; Hostetler *et al.*, 2014], which in turn is 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 space; for large problems, the branching factor leads to poor performance as the feasible search depth is too small.

State abstraction is an important technique for reducing the stochastic branching factor by treating a group of states as a unit [Dearden and Boutilier, 1997; Li *et al.*, 2006]. The space of abstract states is typically much smaller than the original concrete state space. However, abstraction results in a non-Markovian model, 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. The occupancy probability depends on the history of all past actions and abstract states. As an example, in the 3-state MDP depicted in Figure 1a, an edge represents a deterministic transition that is invoked by executing the labeled action. If we group states 1 and 2 into an abstract state $S_{1,2}$, then the probability of reaching state 3 after taking action a in $S_{1,2}$ equals the probability of being actually in state 1, which depends exactly on the number of times that the agent has executed action b in ground state 1 and action a in ground state 2 in the past history prior to entering state 3.

Safe state abstraction methods avoid the non-Markovian problem by ignoring only irrelevant state variables [Dietterich, 1999b; Andre and Russell, 2002] or exploiting particular structure in the transition function (e.g., bisimulation and homomorphism) [Dearden and Boutilier, 1997; Givan *et al.*, 2003; Jiang *et al.*, 2014; Anand *et al.*, 2015]. Such methods result in (near) lossless abstractions but are often inapplicable. One popular proposal to resolve this situation is to introduce an ad hoc *weighting function* (a.k.a. an *aggregation probability*), which functions like an occupancy probability for each concrete state given the abstract state [Bertsekas, 1995; Singh *et al.*, 1995; Li *et al.*, 2006]. 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 computation. 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 POMDP with the original ground MDP as the underlying MDP and the set of abstract states as the set of observations. Belief states in the resulting POMDP replace the otherwise necessary weighting function, 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.

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 *et al.*, 1999] to a POMDP setting, and develop a hierarchical MCTS algorithm that handles Markovian state and action abstractions for MDPs within 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; 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.

2 Background

2.1 MDPs and POMDPs

An MDP is a tuple $\langle S, A, T, R, \gamma \rangle$, where S and A are the state and action spaces, $T(s'|s,a)$ and $R(s,a)$ are the transition and reward functions, and γ is a discount factor [Bellman, 1957]. A solution for an MDP is an *optimal policy* $\pi^* : S \to A$ that maximizes the expected cumulative discounted reward (the *value function*) for all states. The optimal value function V^* satisfies the Bellman equation:

$$V^*(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V^*(s') \right\} . \quad (1)$$

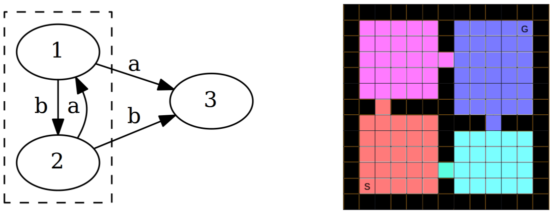
A partially observable MDP, or POMDP, is a tuple $\langle S, A, Z, T, R, \Omega, \gamma \rangle$, where Z is the observation space and $\Omega(z|s,a)$ is the observation function. A sequence of actions and observations defines a history $h = \{a_0, z_1, a_1, z_2 \cdots, a_{t-1}, z_t\}$, which determines uniquely a belief state b with $b(s|h)$ being the probability of being in state s given h. Let $end(h)$ be the last observation in h and let $\mathcal{B}$ and $\mathcal{H}$ be the set of all belief states and histories respectively. A POMDP can be equivalently transformed into an MDP defined over the belief space $\mathcal{B}$ (or the history space $\mathcal{H}$). The Bellman equation for a POMDP in terms of histories is as follows:

$$V^*(h) = \max_{a \in A} \left\{ R(h,a) + \gamma \sum_{z \in Z} \Pr(z|h,a) V^*(haz) \right\} , \quad (2)$$

where $R(h,a) = \sum_{s \in S} R(s,a) b(s|h)$, $\Pr(z|h,a)$ is the probability of observing z after taking action a at history h, and haz denotes the history resulting from taking action a at history h and observing z afterwards.

2.2 Monte Carlo Tree Search

For online planning in (PO)MDPs, 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. *Upper confidence bounds for trees* (UCT) is one of the most popular implementations of MCTS for MDPs using the UCB action-selection heuristic to guide the tree search [Kocsis and Szepesvári, 2006]. *Partially observable Monte Carlo planning* (POMCP) extends UCT to POMDPs by employing particle filtering and root sampling, where each simulation starts with a state sampled from the belief state $b(s|h)$ represented as a set of particles at the root node [Silver and Veness, 2010].



(a) A 3-state MDP.

(b) The rooms example.

Figure 1: State abstraction examples.

3 The Approach

3.1 State Abstraction

Consider a ground MDP $M = \langle S, A, T, R, \gamma \rangle$. Let $X = \{x_1, x_2, \cdots\}$ be a partition on S, and $\varphi : S \to X$ be an abstraction function, such that $\varphi(s) \in X$ is the abstract state corresponding to ground state s. As an example, Figure 1b 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.

A so-called weighting function w has been introduced in the literature to ensure that the abstract transition and reward functions can be written in a Markovian way:

$$T_\varphi(x'|x,a) = \sum_{\varphi(s)=x} \sum_{\varphi(s')=x'} T(s'|s,a) w(s,x), \quad (3)$$

and

$$R_\varphi(x,a) = \sum_{\varphi(s)=x} R(s,a) w(s,x), \quad (4)$$

where $w(s,x)$ approximates the abstract-state-specific occupancy probability $\Pr(s|x)$ of being in ground state s given that the agent is in abstract state x [Bertsekas, 1995; Li *et al.*, 2006]. An abstract MDP $\langle X, A, T_\varphi, R_\varphi, \gamma \rangle$ is then defined on the abstract space X. Solving this abstract MDP gives a policy $\pi_\varphi : X \to A$, which can be translated to the ground MDP M. The problem here is that the true occupancy probability $\Pr(s|h)$, with h being the whole history of past actions and abstract states, is non-stationary. It cannot reasonably be approximated as a constant weighting function conditioned only on abstract state; in fact, it follows a Bayesian update

$$\Pr(s'|hax) = \eta \mathbf{1}[x = \varphi(s')] \sum_{s \in S} T(s'|s,a) \Pr(s|h), \quad (5)$$

where $\mathbf{1}$ is the indicator function and η is a normalizing factor. On the other hand, if we introduce Ω as a conditional probability function such that $\Omega(x|s) = \mathbf{1}[x = \varphi(s)]$ for any

$x \in X$ and $s \in S$, then the abstract problem 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.

From a POMDP perspective, the weighting function approach actually tries to approximate the belief state $b(s|h)$ using a constant 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 *et al.*, 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 infeasible 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.

In this paper, we employ a POMCP algorithm running within POMDP(M, φ) to find an online policy for the ground MDP with state abstraction, and refer to the resulting algorithm as POMCP(M, φ). The online policy produced by POMCP(M, φ) 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 *generative model* as a *simulator* of M is needed. Another important advantage with POMCP(M, φ) is that it can be extended naturally to problems with continuous state spaces without significant modifications, given suitable abstraction functions defined over continuous states.

3.2 Action Abstraction

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 (a.k.a. temporal abstraction) can increase the effective search depth by considering high-level actions composed from many concrete actions [Parr and Russell, 1998; Sutton *et al.*, 1999; Dietterich, 1999a; Barto and Mahadevan, 2003]. A given state abstraction naturally induces an action abstraction, where abstract actions connect abstract states in a one high-level step.

We extend Sutton's options framework to a POMDP setting to model abstract actions within POMDP(M, φ). A temporal transition from an abstract state $x \in X$ to one of its neighbors $y \in X$ is considered a named option $o_{x \to y}$, which is defined as a tuple $\langle \mathcal{I}, \pi, \beta \rangle$. Here, $\mathcal{I}$ is the initiation set $\mathcal{I} = \{h \mid h \in \mathcal{H} \land end(h) = x\}$ indicating that $o_{x \to y}$ is executable only at a history ending with observation x, $\pi : \mathcal{H} \to A$ is a local policy for $o_{x \to y}$ defined over $\mathcal{H}$ and β is a termination condition with $\beta(h) = 1$ if $end(h) = y$, and $\beta(h) = 0$ otherwise. In the rooms domain, for example, $o_{A \to B}$ is an 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.

Let $\mathcal{O}$ be the set of options consisting of all possible direct transitions between abstract states. The set $\mathcal{O}$ can either be constructed manually by utilizing the neighboring relationship between abstract states, or learned incrementally from an empty set by introducing a new option each time a new abstract-state transition has been observed in a Monte Carlo simulation process. In this paper, we assume the former case for convenience, so the set of options is fixed in advance. The main results can also be extended to the latter case. It is not necessary to specify the local policy for each option beforehand. In fact, the proposed algorithm learns the high-level option-selection policy and the low-level, local option policies simultaneously via Monte Carlo simulation.

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 timestep. It has been shown that the polling execution of a hierarchical pol-

```
Agent ($s_0$ : initial state, $\varphi$ : $abstraction\ function$,
  $\Pi_{rollout}$ : 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

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 $\langle r, n+1, h'', s''\rangle$

GetGreedyPrimitive ($t$ : task, $h$ : history)
if $t$ is primitive then
    return $t$
else
    $a^* \leftarrow \text{argmax}_a\, Q[t, h, a]$
    return GetGreedyPrimitive ($a^*, h$)

GetPrimitive ($\Pi$ : policy, $t$ : task, $h$ : history)
if $t$ is primitive then
    return $t$
else
    return GetPrimitive ($\Pi, \pi_t(h), h$)

OnlinePlanning ($h$ : history, $\mathcal{T}$ : search tree,
  $\varphi$ : $abstraction\ function$, $\Pi_{rollout}$ : 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, $h$)

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 $\langle r, n' + n'', h'', s''\rangle$
```

Figure 2: POMCP($M, \varphi, \mathcal{O}$) — Markovian state and action abstractions for MDPs via hierarchical MCTS.

icy Π yields higher expected value than the hierarchical execution of the same hierarchical policy [Sutton *et al.*, 1999; Dietterich, 1999a]. 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 the 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

$$Q^{\mu}(h, o) = V^{\pi_o}(h) + \sum_{h' \in \mathcal{H}} \gamma^{|h'|-|h|} \Pr(h'|h, o) V^{\mu}(h'), \tag{6}$$

where $\Pr(h'|h, o)$ — the termination distribution of option o — gives the probability that o terminates at history h' after $|h'| - |h|$ timesteps. 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, i.e.

$$Q^{\pi_o}(h, a) = R(h, a) + \gamma \sum_{x \in X} \Pr(x|h, a) V^{\pi_o}(hax). \tag{7}$$

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 6 and 7. 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 by POMCP($M, \varphi, \mathcal{O}$) 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.

3.3 The Main Algorithm

The main algorithm POMCP($M, \varphi, \mathcal{O}$) is outlined in Figure 2. 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 **`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. 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 which 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.

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 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 6 for the root task or Equation 7 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 **`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. 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. 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 searching in the space of belief states due to the way the tree is expanded and the value functions are updated.

4 Theoretical Results

4.1 Optimality Results with State Abstraction

Inspired by the abstraction criteria introduced in [Hostetler *et al.*, 2014], we define aggregation error as follows

Definition 1. *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 1. *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 $E(a^*) \le 2He$ if $\gamma = 1$, else $E(a^*) \le 2\gamma \frac{1-\gamma^H}{1-\gamma} e$.*

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:

$$E_d(h, a) = \left| Q_d(h, a) - \sum_{s \in S} b(s|h) Q_d(s, a) \right|, \tag{8}$$

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

$$\begin{aligned} E_d(h, a) = \gamma & \left| \sum_{h' \in \mathcal{H}} \Pr(h'|h, a) V_{d+1}(h') \right. \\ & \left. - \sum_{s \in S} b(s|h) \sum_{s' \in S} T(s'|s, a) V_{d+1}(s') \right| \\ = \gamma & \left| \sum_{h' \in \mathcal{H}} \Pr(h'|h, a) V_{d+1}(h') \right. \\ & \left. - \sum_{s' \in S} V_{d+1}(s') \sum_{s \in S} b(s|h) T(s'|s, a) \right|. \end{aligned} \tag{9}$$

Noticing that

$$\begin{aligned} \Pr(s'|h, a) &= \sum_{s \in S} T(s'|s, a) b(s|h) \\ &= \sum_{h' \in \mathcal{H}} b(s'|h') \Pr(h'|h, a), \end{aligned} \tag{10}$$

it follows that

$$\begin{aligned} E_d(h, a) = \gamma & \left| \sum_{h' \in \mathcal{H}} \Pr(h'|h, a) V_{d+1}(h') \right. \\ & \left. - \sum_{h' \in \mathcal{H}} \Pr(h'|h, a) \sum_{s' \in S} b(s'|h') V_{d+1}(s') \right| \\ \le \gamma & \sum_{h' \in \mathcal{H}} \Pr(h'|h, a) \left| V_{d+1}(h') \right. \\ & \left. - \sum_{s' \in S} b(s'|h') V_{d+1}(s') \right|, \end{aligned} \tag{11}$$

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

$$\left| V_d(h) - \max_{a \in A} \sum_{s \in S} b(s|h) Q_d(s, a) \right| \le \max_{a \in A} E_d(h, a), \tag{12}$$

and

$$\begin{aligned} & \left| \max_{a \in A} \sum_{s \in S} b(s|h) Q_d(s, a) - \sum_{s \in S} b(s|h) V_d(s) \right| \\ & \le \sum_{s \in S} b(s|h) \left| Q_d(s, \mathring{a}) - V_d(s) \right| \le e, \end{aligned} \tag{13}$$

we have

$$\left| V_d(h) - \sum_{s \in S} b(s|h) V_d(s) \right| \le \max_{a \in A} E_d(h, a) + e. \tag{14}$$

Therefore,

$$\begin{aligned} E_d(h, a) & \le \gamma \left[\sum_{h' \in \mathcal{H}} \Pr(h'|h, a) \max_{a' \in A} E_{d+1}(h', a') + e \right] \\ & \le \gamma \left[\max_{h' \in \mathcal{H}, a' \in A} E_{d+1}(h', a') + e \right], \end{aligned} \tag{15}$$

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 $E(d) \le \gamma[E(d+1) + e]$. At terminal nodes, both $Q_H(h, \cdot)$ and $Q_H(s, \cdot)$ equal 0, so E(H) = 0. Therefore, $E(d) \le (H - d)e$ if $\gamma = 1$, otherwise $E(d) \le \gamma \frac{1-\gamma^{H-d}}{1-\gamma} e$. 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 $Q_0(h_0, a^*) \ge Q_0(h_0, b^*)$. Noticing that $\mathcal{P}(h_0) = \{s_0\}$, since $|Q_0(s_0, b^*) - Q_0(h_0, b^*)| \le E(0)$ and $|Q_0(s_0, a^*) - Q_0(h_0, a^*)| \le E(0)$, we get $E(a^*) = |Q_0(s_0, b^*) - Q_0(s_0, a^*)| \le 2E(0)$. □

4.2 Convergence Results with Action Abstraction

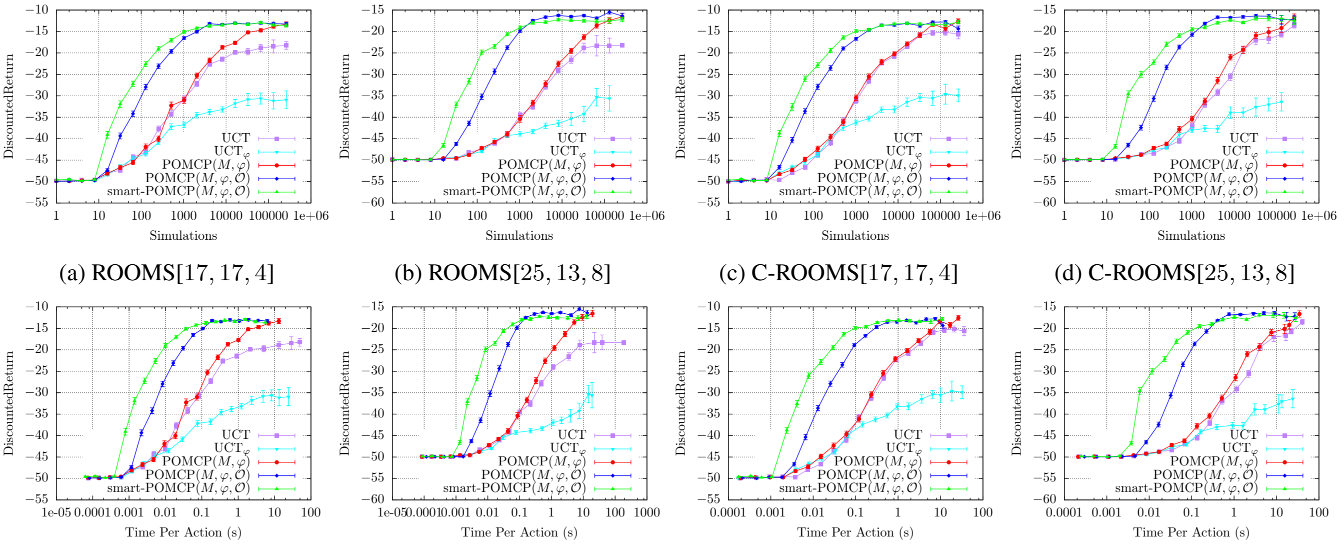
Theorem 2. *With probability 1, POMCP($M, \varphi, \mathcal{O}$) converges to a recursively optimal hierarchical policy for POMDP(M, φ) over the hierarchy defined by the input state and action abstractions.*

Proof. (Sketch) For a particular option, POMCP($M, \varphi, \mathcal{O}$) finds the optimal policy with probability 1 following the convergence results of POMCP in the limit. Given the fact that, when an option converges, its termination distribution also converges, the root task reduces to a stationary semi-Markov Decision Process (SMDP) defined over the belief space. POMCP($M, \varphi, \mathcal{O}$) thus finds the optimal policy in the limit for the root task within the converged SMDP by extending the convergence results of POMCP to SMDPs. Since the high-level policy is optimal given optimal low-level policies, POMCP($M, \varphi, \mathcal{O}$) converges to a recursively optimal hierarchical policy. □

5 Experiments

5.1 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 1b. 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



(a) ROOMS$[17, 17, 4]$ (b) ROOMS$[25, 13, 8]$ (c) C-ROOMS$[17, 17, 4]$ (d) C-ROOMS$[25, 13, 8]$

(e) ROOMS$[17, 17, 4]$ (f) ROOMS$[25, 13, 8]$ (g) C-ROOMS$[17, 17, 4]$ (h) C-ROOMS$[25, 13, 8]$

Figure 3: Empirical results on the rooms and continuous rooms domains.

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. Figures 3a, 3e, 3b and 3f with x axis in log scale show the results of running UCT, UCT$_\varphi$, POMCP(M, φ), POMCP($M, \varphi, \mathcal{O}$) and smart-POMCP($M, \varphi, \mathcal{O}$) 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(M, φ) is a POMCP algorithm running on POMDP(M, φ) resulting from doing state abstraction on the ground MDP M, POMCP($M, \varphi, \mathcal{O}$) is the proposed hierarchical MCTS algorithm running on POMDP(M, φ), and smart-POMCP($M, \varphi, \mathcal{O}$) is a POMCP($M, \varphi, \mathcal{O}$) algorithm equipped with hand-coded informative rollout policies for options. UCT, UCT$_\varphi$, POMCP(M, φ) and POMCP($M, \varphi, \mathcal{O}$) 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) in terms of the number of simulations and the averaged computation time per action. It can be seen from the results that POMCP(M, φ) 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 $\Pr(s|x)$ 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 — POMCP($M, \varphi, \mathcal{O}$) — outperforms UCT by orders of magnitude. POMCP($M, \varphi, \mathcal{O}$) also outperforms POMCP(M, φ) 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-POMCP($M, \varphi, \mathcal{O}$) improves on POMCP($M, \varphi, \mathcal{O}$) significantly. The possibility of introducing option-specific rollout policies can also be considered as an advantage of POMCP($M, \varphi, \mathcal{O}$).

5.2 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(M, φ) and POMCP($M, \varphi, \mathcal{O}$) algorithms in this domain, we need only provide the appropriate observation function defined over the continuous state space. Figures 3c, 3g, 3d and 3h show the results of running UCT, UCT$_\varphi$, POMCP(M, φ), POMCP($M, \varphi, \mathcal{O}$) and smart-POMCP($M, \varphi, \mathcal{O}$) in C-ROOMS$[17, 17, 4]$ and C-ROOMS$[25, 13, 8]$ problems, confirming that POMCP(M, φ) and POMCP($M, \varphi, \mathcal{O}$) 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.

6 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 original MAXQ decomposition is not completely applicable to exploit the hierarchical structure defined by abstract states. A MAXQ subtask is designed to be 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, 1999a]. 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 $x \to y$ 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 $A \to B$ and $A \to C$ 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 $A \to B$ and $D \to B$ 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 advance.

7 Conclusion

In this paper, we propose state- and action-abstracted MDPs can be viewed as POMDPs. We bound the performance loss induced by the abstraction and we describe a hierarchical MCTS algorithm for approximately solving the abstract POMDP. The algorithm converges to a recursively optimal hierarchical policy for the ground MDP consistent with the input state and action abstractions. Empirical results show that the proposed approach improves ground MCTS by orders of magnitude. In future work, we plan to extend this approach to 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 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. The authors would also like to thank Richard Doan and the anonymous reviewers for their valuable comments and suggestions.

References

[Anand *et al.*, 2015] Ankit Anand, Aditya Grover, Mausam Mausam, and Parag Singla. ASAP-UCT: abstraction of state-action pairs in UCT. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1509–1515, 2015.

[Andre and Russell, 2002] David Andre and Stuart J. Russell. 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*, pages 119–125, 2002.

[Bai *et al.*, 2012] Aijun Bai, Feng Wu, and Xiaoping Chen. Online planning for large MDPs with MAXQ decomposition. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, June 2012.

[Bai *et al.*, 2015] Aijun Bai, Feng Wu, and Xiaoping Chen. Online planning for large Markov decision processes with hierarchical decomposition. *ACM Transactions on Intelligent Systems and Technology*, 6(4):45, 2015.

[Barto and Mahadevan, 2003] A.G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13:341–379, 2003.

[Barto *et al.*, 1995] A.G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138, 1995.

[Bellman, 1957] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.

[Bertsekas, 1995] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Trans. on Computational Intelligence and AI in Games*, 4:1–43, 2012.

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

[Dietterich, 1999a] Thomas G Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Machine Learning Research*, 13(1):63, May 1999.

[Dietterich, 1999b] Thomas G. Dietterich. State abstraction in MAXQ hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, pages 994–1000, 1999.

[Givan *et al.*, 2003] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1):163–223, 2003.

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

[Hostetler *et al.*, 2014] Jesse Hostetler, Alan Fern, and Tom Dietterich. State aggregation in Monte Carlo tree search. In *Proceedings of 28th AAAI Conference on Artificial Intelligence*, 2014.

[Jiang *et al.*, 2014] Nan Jiang, Satinder Singh, and Richard Lewis. Improving UCT planning via approximate homomorphisms. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1289–1296, 2014.

[Kearns *et al.*, 2002] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near optimal planning in large Markov Decision Processes. *Machine Learning*, 49(2-3):193–208, 2002.

[Kocsis and Szepesvári, 2006] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

[Li *et al.*, 2006] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*, 2006.

[Parr and Russell, 1998] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, volume 10, 1998.

[Silver and Veness, 2010] D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, pages 2164–2172, 2010.

[Singh *et al.*, 1994] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the 11th International Conference on Machine Learning*, pages 284–292, 1994.

[Singh *et al.*, 1995] Satinder P Singh, Tommi Jaakkola, and Michael I Jordan. Reinforcement learning with soft state aggregation. *Advances in Neural Information Processing Systems*, pages 361–368, 1995.

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

[Vien and Toussaint, 2014] Ngo Anh Vien and Marc Toussaint. Hierarchical Monte-Carlo planning. In *Proceedings of the 29th AAAI Conference on Artificial*, 2014.