Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Hierarchical Monte-Carlo Planning

Ngo Anh Vien
Machine Learning and Robotics Lab
University of Stuttgart, Germany
vien.ngo@ipvs.uni-stuttgart.de

Marc Toussaint
Machine Learning and Robotics Lab
University of Stuttgart, Germany
marc.toussaint@ipvs.uni-stuttgart.de

Abstract

Monte-Carlo Tree Search, especially UCT and its POMDP version POMCP, have demonstrated excellent performance on many problems. However, to efficiently scale to large domains one should also exploit hierarchical structure if present. In such hierarchical domains, finding rewarded states typically requires to search deeply; covering enough such informative states very far from the root becomes computationally expensive in flat non-hierarchical search approaches. We propose novel, scalable MCTS methods which integrate a task hierarchy into the MCTS framework, specifically leading to hierarchical versions of both, UCT and POMCP. The new method does not need to estimate probabilistic models of each subtask, it instead computes subtask policies purely sample-based. We evaluate the hierarchical MCTS methods on various settings such as a hierarchical MDP, a Bayesian model-based hierarchical RL problem, and a large hierarchical POMDP.

Introduction

Monte-Carlo Tree Search (MCTS) (Coulom 2006) has become a standard approach to a wide variety of planning problems. In particular UCT (Upper Confidence Bound applied to Trees) for MDPs (Kocsis and Szepesvári 2006) and its extension to POMDPs (POMCP) (Silver and Veness 2010) became the state-of-the-art for many challenging large problems such as Computer Go and other games. POMCP and UCT have shown to be promising in tackling two notorious problems in planning: the *curse of dimensionality*, which means the exponential growth of complexity with the size of the state space, and the *curse of history* which means the exponential growth of complexity with the length of the planning horizon.

However, the complexity of non-hierarchical planning is still exponential in the horizon and the performance of POMCP and UCT degrades when not given enough computational time. The exponential growth of necessarily visited nodes, which is $O(|\mathcal{A}|^T)$ (if T is the horizon), is due to the action branching after each step. If macro actions are used instead of primitive actions, the branching number would decrease to $O(|\mathcal{A}|^{T/L})$ (if L is the maximum number of steps executed by macro actions) (He, Brunskill, and Roy 2010). However, if policies were restricted to an action hierarchy which is a hierarchy of macro actions with multiple abstract levels, the complexity of planning would be enormously reduced. Hence, the integration of an action hierarchy into planning algorithms would let them be scalable to very large problems (Dietterich 2000; Pineau 2004).

In this paper, we propose new MCTS methods for hierarchical planning, which seamlessly integrates an action hierarchy in POMCP (as well as UCT). Our formulation is generally made for POMDPs. To demonstrate the efficiency of the novel MCTS method, we evaluate it on various hierarchical planning domains in MDPs, POMDPs and a Bayesian reinforcement learning setting.

Related Work

MCTS algorithms (Coulom 2006; Browne et al. 2012) have long been applied to planning under uncertainty, and recently received wide attention from many other communities. Among those, UCT (Kocsis and Szepesvári 2006) is one of the most successful examples and has been extensively applied, e.g., to Computer Go (Gelly and Silver 2011), Amazon (Lorentz 2008), Game Playing (Finnsson and Björnsson 2008). UCT was previously combined with macro actions (Powley, Whitehouse, and Cowling 2013) in a simple form that repeats executing the same actions for a fixed period of time steps. Existing MCTS algorithms, including UCT, cannot efficiently cope with long-horizon problems. When the terminal states can only be reached after many time steps, MCTS algorithms must extensively build a very deep search tree which requires a large number of rollout simulations which is exponential in the tree depth.

In order to tackle long-horizon problems, planning algorithms are usually integrated with macro actions or a task hierarchy, which constitutes a hierarchical planning style. Macro actions and action hierarchies have previously been used for planning and learning in MDP (Barto and Mahadevan 2003). Later the idea was extended for POMDPs (Hansen and Zhou 2003; Theocharous and Kaelbling 2003; Pineau 2004; He, Brunskill, and Roy 2010; Lim, Hsu, and Sun 2011; Ma and Pineau 2014). These work range from offline planning (Hansen and Zhou 2003; Theocharous and Kaelbling 2003; Pineau 2004; Lim, Hsu, and Sun 2011; Ma and Pineau 2014) to online planning (He, Brunskill, and Roy 2010), and with assumption of known subgoals (Hansen and Zhou 2003; Theocharous and Kaelbling 2003;

Pineau 2004; Lim, Hsu, and Sun 2011) to subgoal discovery (He, Brunskill, and Roy 2010; Ma and Pineau 2014). The work of (Pineau 2004; Lim, Hsu, and Sun 2011) assumed macro policies are partially defined in terms of the respective subgoals. The work of (Hansen and Zhou 2003) optimized finite state controllers, and (Toussaint, Charlin, and Poupart 2008) used an expectation maximization (EM) algorithm to learn both hierarchical controller and macro policies. On the other hand, the approaches in (Pineau 2004) and Theocharous and Kaelbling (Theocharous and Kaelbling 2003) went to approximate all subtasks' models to then use a standard POMDP solver to find the policies. It is clear that these approximation approaches can not scale up well for large problems.

The work (Bai, Wu, and Chen 2012) also extends MAXQ to online hierarchical planning, however they need to estimate the model (transition $\Pr(s', t|s, a)$) of macro actions in order to evaluate completion functions. To the best of our knowledge, our paper is the first to propose and demonstrate a purely sample-based MCTS extension for hierarchical planning.

Background

POMDP

POMDPs are a generalization of MDPs where states are partially observable. A POMDP is defined as a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{Z}, \mathcal{R}, b_0\}$; where $\mathcal{S}$ is an unobservable state space; $\mathcal{O}$ is an observation space; $\mathcal{A}$ is an action space; $\mathcal{T}$ defines a state transition function, $\mathcal{T}(s, a, s') = \Pr(s'|s, a)$; $\mathcal{Z}$ is an observation function, $\mathcal{Z}(s', a, o) = \Pr(o|s', a)$; b_0 is an initial belief which represents a probability distribution over states $b_0(s) = \Pr(s_0 = s)$; and $\mathcal{R}$ is a reward function, $\mathcal{R}(s, a, s')$. A policy $\pi : \mathcal{B} \mapsto \mathcal{A}$ is defined as a mapping from the belief space to the action space. We denote the belief update operator as $b_{t+1} = \tau(b_t, a, o)$.

MAXQ

The MAXQ framework was first introduced in the MDP setting and assumes that we are given an action hierarchy $\mathcal{H} = \{a_0, a_1, \ldots, a_n\}$, which is a tree of primitive actions (at leaf nodes) and compound actions (internal nodes). MAXQ is an instance of semi-MDP formulation (SMDP). A core concept of MAXQ is a hierarchical value decomposition. In contrast to plain MDPs, in SMDPs the transition function is defined as the joint probability distribution $p(s', k|s, a)$ over the next state and the number of execution time steps when taking action a in state s. The Bellman equations for SMDPs can be written as follow

$$V^\pi(s) = \bar{R}(s, \pi(s)) + \sum_{s',k} \gamma^k p(s', k|s, \pi(s)) V^\pi(s') \quad (1)$$

where $\bar{R}(s, \pi(s))$ is the expected reward received when taking action $\pi(s)$ at state s; π_i is a policy of the task a_i, and $\pi = \pi_0$. When defining the *reward term* $V^\pi(i, s)$ as the expected value of following a policy π starting at state s until the subtask i terminates we can rewrite the above Bellman equation as

$$V^\pi(i, s) = V^\pi(\pi_i(s), s) + C^\pi(i, s, \pi_i(s)), \quad (2)$$

where a new term $C^\pi(i, s, \pi_i(s))$ is called the *completion function*, which is the expected reward received when continuing to follow the policy π until task i terminates. This term is given as

$$C^\pi(i, s, a) = \sum_{s',k} \gamma^k p_i^\pi(s', k|s, a) V^\pi(i, s') \quad (3)$$

where $p_i^\pi(s', k|s, a)$ is the joint probability distribution which defines dynamics of subtask i.

As a consequence, we can recursively decompose the value function at root node $V^\pi(0, s)$ until the value functions of primitive actions appear, at which $V(i, s) = R(i, s)$ for i are primitive actions. Existing methods for hierarchical planning in MDPs estimate the model of compound actions $p_i^\pi(s', k|s, a)$ as PolCA method (Pineau 2004). In hierarchical RL setting, the MAXQ algorithm was proposed by Dietterich (Dietterich 2000).

Partially Observable Semi-MDP

Partially observable semi-MDPs (POSMDPs) (White 1976) generalize POMDPs by integrating macro actions. A POSMDP is defined as a 6-tuple $\{\cdot, \cdot, \mathcal{A}, \mathcal{P}, \mathcal{R}\}$, where the state and observation spaces are as in a POMDP; $\mathcal{A}$ is an action space which could consist of either primitive actions or macro actions; $\mathcal{P}$ is a joint probability distribution over a next state, the number of time steps and the observation if in state s an action a is executed, $\mathcal{P}(s, a, s', \mathbf{o}, k) = Pr(s', k, \mathbf{o}|s, a)$, where $\mathbf{o} \in \mathcal{O}^k$ is a sequence of primtive observations o_t until time k; $\mathcal{R}$ is a reward function $\mathcal{R}(s, a)$. The belief update operator is $b' = \tau(b, a, \mathbf{o})$, where

$$b'(s') \propto \sum_{t=1}^{\infty} \gamma^{t-1} \sum_{s \in \mathcal{S}} \Pr(s', \mathbf{o}, t|s, a) b(s) . \quad (4)$$

Monte-Carlo Tree Search

Upper Confidence Bounds (UCB) applied to Trees (UCT) (Kocsis and Szepesvári 2006), a standard instance of MCTS algorithms, is a tree search algorithm for planning in MDPs which uses UCB1 (Auer, Cesa-Bianchi, and Fischer 2002) as the tree policy. Later, it was extended for planning in a POMDP, which is called *Partially Observable Monte-Carlo Planning* (POMCP) (Silver and Veness 2010). The POMCP method extends UCT in two important ways: 1) it builds a search tree of beliefs instead of states, e.g. using particle sets as belief representation, 2) action selection is based on counts and value functions of beliefs instead of states. Each node of the search tree consists of a tuple $\langle n(b), V(b) \rangle$, where $n(b)$ is the number of times that belief b has been visited; and $V(b)$ is the average return of simulations which have started from belief b. When running a simulation, if a node has all its actions selected at least once, its subsequent actions are selected by the UCB1 policy, named *tree policy*, in which it maximizes the sum of the average return and an exploration bonus

$$a = \text{argmax}_{a'} \left[Q(b, a') + c\sqrt{\log n(b)/n(b, a')} \right] \quad (5)$$

where $n(b, a')$ is the total number of times that action a' has been selected at belief b, and c is a bias constant.

Otherwise, the actions are selected by another policy, named *rollout policy*. A uniform action selection strategy is often used for the rollout policy. Domain knowledge can be

used to design a better rollout policy. When the simulation ends only one new node is added to the search tree. This node corresponds to the next belief of the last belief in the tree visited during that simulation.

Hierarchical POMDPs as a Hierarchy of POSMDPs

We first discuss how a hierarchical POMDP implies a hierarchy of POSMDPs, similar to the approach of Pineau (Pineau 2004). The main differences are two-fold: 1) we decompose tasks until we meet primitive subtasks which cannot be further decomposed, which is similar to the original procedure in MAXQ, instead of using an approximation of subtasks' models or value functions; 2) these decomposed tasks are solved online by our proposed hierarchical MCTS algorithm.

Given an action hierarchy as previously described, the hierarchical policy $\pi = \{\pi_0, \cdots, \pi_n\}$ is defined as a tree of subtask policies. The execution of a subtask terminates with the observation of a terminal symbol or when exceeding a maximum execution time. The call of a task policy leads to calls of all its child policies. When a child node's execution terminates, it returns control to its immediate parent node in the hierarchy. To optimize the subtask policies we will first establish below that each subtask with macro actions as child nodes implies a POSMDP. We then continue to describe how to decompose the value functions for such a POSMDP, giving similar definitions for reward terms and completions as in the MAXQ framework.

Definition 1 *We define $V^{\pi}(i, b_t)$ as the expected value when following a policy π, starting at belief point b_t at time t until task i terminates.*

The reward term is also interpreted as the projected value function of the policy π at belief b_t on subtask i (Dietterich 2000).

Definition 2 *The* completion term *is the expected value of completing task i after executing the subtask $\pi_i(b_t)$ at b_t.*

$$C^{\pi}(i, b_t, \pi_i(b_t)) = \gamma \sum_{\mathbf{o},k} p\big(\mathbf{o}, k | b_t, \pi(b_t)\big) \times \gamma^k V^{\pi}\big(i, \tau(b_t, \pi_i(b_t), \mathbf{o}, k)\big)$$

where the observation $\mathbf{o} \in \mathcal{O}^k$ is a vector of subsumed observations received between time t and $t+k$; the belief update $\tau(b_t, \pi_i(b_t), \mathbf{o}, k)$ is computed by k consecutive primitive updates $\{\tau(b_{t+h}, a_{t+h}, o_{t+h})\}_{h=0}^{k}$. Hence, we can decompose $V^{\pi}(i, b_t)$ into two terms: a reward term which is received when taking the first action at b_t, and a completion term which is received when continuing following policy π until termination of task i.

$$V^{\pi}(i, b_t) = V^{\pi}\big(\pi_i(b_t), b_t\big) + \gamma \sum_{\mathbf{o},k} p\big(\mathbf{o}, k | b_t, \pi(b_t)\big) \times \gamma^k V^{\pi}\big(i, \tau(b_t, \pi_i(b_t), \mathbf{o}, k)\big)$$

where π_i is a macro action's policy for node i. The completion term is a mixture of projected value functions on subtask i at different next belief points. Therefore, this term continues being decomposed until we have a projected value function on a primitive actions which is immediately equal to $r(b_t, a) = \sum_s r(s, a) b_t(s)$.

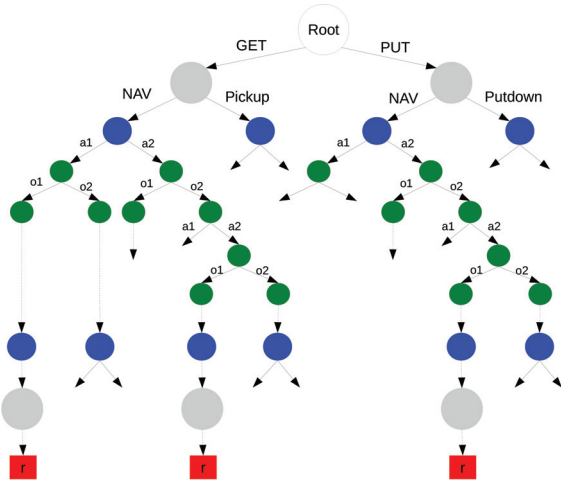

Figure 1: An example similar with the Taxi problem, with four actions $\{a_1, a_2, Pickup, Putdown\}$. The macro actions are GET, PUT, NAV. Dotted lines represent rollout policy calls, solid lines are tree policy calls. The green subtree of NAV consists of only primitive actions. The blue subtrees of GET and PUT consist of NAV and Pickup/Putdown. The gray subtree of Root consists of GET and PUT.

From the above decomposition we can derive an online hierarchical MCTS planning algorithm for hierarchical POMDP problems. First, note that the reward term, which is an immediate reward if projected on its parent task and was previously represented by one node in the search tree in flat MCTS for MDP/POMDP as it is primitive, should now be represented by one sub-search tree in the parent's search tree. Therefore a search tree of many subtrees is used for planning in hierarchical POMDPs. When simulating the subtask $\pi_i(b)$, its corresponding sub-search tree is used to evaluate its reward term's rollout. The leaf nodes of a sub-search tree serve as the next nodes of its parent's search tree. As depicted in Figure 1, the NAV subtask is executed with a search tree labeled by green colors, its leaves are the next nodes of the GET search tree, etc.

In the next section we discuss in detail how to algorithmically integrate a task hierarchy in POMCP for online planning in hierarchical POMDPs.

Hierarchical Monte-Carlo Planning

In this section, we propose a generalization of both, UCT and POMCP, for planning in hierarchical MDPs and POMDPs. The extension needs to deal with 1) POSMDP planning and 2) online planning with a hierarchy of POSMDPs. We call the two extended methods *Hierarchical UCT* (H-UCT) and *Hierarchical POMCP* (H-POMCP), which both consists of multiple levels of hierarchical H-UCT/H-POMCP searches. To unify their description, in the following we just refer to H-UCT but include the (optional) belief set maintenance in the algorithms to handle the H-POMCP case.

We describe the generic approach first. Each H-UCT search is responsible for selecting the best action with respect to the subgoal of a particular macro action. The generic algorithm consists of two interleaved stages: exe-

Algorithm 1 MAIN and EXECUTE procedures

MAIN(action hierarchy $\mathcal{H}$)
From the ROOT node, call EXECUTE$(0, b_0, \mathcal{H})$.

EXECUTE(task i, belief b, action hierarchy $\mathcal{H}$)
1: $\mathbf{o} = []$
2: **if** i is primitive **then**
3: execute i, observe r, o.
4: **return** o.
5: **else**
6: **while** i not terminates **do**
7: j = H-UCT(i,b,$\mathcal{H}$) // Action selection
8: $\mathbf{obs}$ = EXECUTE(j,b)
9: $\mathbf{o} = [\mathbf{o}; \mathbf{obs}]$
10: $b = \tau(b, j, \mathbf{o})$. // belief update
11: **return** $\mathbf{o}$.

Algorithm 2 H-UCT

H-UCT(task i, belief b, hierarchy $\mathcal{H}$)
1: **Require:** hierarchy $\mathcal{H}$, belief b.
2: Initialize search tree T for task i.
3: **for** $n = 1 : N$ **do**
4: Sample $s \sim b$.
5: SIMULATE$(s, b, i, \mathcal{H}, T, 0)$
6: **return** $\arg\max_a Q(i, b, a)$

cution which means executing an action and observing an observation and reward as shown in Algorithm 1, and planning to select actions as shown in Algorithm 2.

Each macro action in the action hierarchy has a H-UCT subroutine for action selection as in line 7 of EXECUTE in Algorithm 1 (for each different i). The child task is executed until termination, then returns control to its parent subroutine. If the task i consists of only primitive actions, then its action selection subroutine is a POMCP. If the task i is a sub-hierarchy (the task GET is a sub-hierarchy, from node GET to NAVIGATE, pickup to primitive actions: {*move-left, move-right, move-down, move-up*}) then we resort to hierarchical UCT planning. The action selection algorithm is shown in Algorithm 2. At the current belief, we sample N states, then evaluate its value function by simulations. This evaluation calls SIMULATE, shown in Algorithm 4. The rollout policy is to hierarchically call the rollout policies of subtasks as shown in Algorithm 3.

We now explain how beliefs are maintained and the hierarchical tree is incrementally built. The search tree of H-UCT is different from that of flat UCT in three ways

First, each node in the tree contains similar information $\langle N(b), V(b)\rangle$. However its children action nodes a are represented by sub-search trees. Whenever a simulation executes a child action, it would run until the leaf nodes of that child's sub-search tree, then returns a corresponding sequence of simulated observations and the reward term for the respective macro action. Its observation branching depends on this observation sequence. The simulation inside the sub-search tree is directed by its corresponding tree policy (or rollout policy), which means that simulation is guided towards how to accomplish the subtask of this macro action. The counts and values of selecting these children macro actions are also maintained $\langle N(b, a), V(b, a)\rangle$.

Algorithm 3 ROLLOUT

ROLLOUT$(s, i, \mathcal{H}, depth)$
1: $steps = 0$
2: $\mathbf{o} = null$
3: **if** i is primitive **then**
4: $(s', o, r, 1) \sim$ Generative-Model(s, i)
5: **return** $s', o, r, 1$
6: **else**
7: **if** i terminates **or** $\gamma^{depth} < \epsilon$ **then**
8: **return** 0
9: $a \sim \pi_{rollout}(i, \cdot)$
10: $[s', \mathbf{obs}, r, k] =$ ROLLOUT$(s, a, \mathcal{H}, 0)$
11: $steps = steps + k$
12: $\mathbf{o} = [\mathbf{o}; \mathbf{obs}]$
13: $[\cdot, \mathbf{obs}, R, n] =$ ROLLOUT$(s', i, \mathcal{H}, depth + k)$
14: **return** $s', \mathbf{o}, r + \gamma^k \cdot R, steps$

Second, each sub-search tree may include other sub-search trees, if its children are still macro actions. The policy used during a simulation depends on which sub-search tree it is currently in.

Third, in the case of a macro action, which node of the search tree is expanded next depends on the macro action as usual, but also on the whole sequence of observations, as mentioned in the previous section.

Theoretical Results

In UCT, the main source of the bias of the estimated expected return at the root node is due to the limited number of simulations. In (Kocsis and Szepesvári 2006) (Theorem 7) it was proven that the bias reflects that the number of necessarily visited states is exponential in the horizon. In H-UCT, the bias comes from two main sources: the planning horizon—which is similar to UCT—and the propagated bias from subtrees as its nodes.

We first study the bias for a simplified H-UCT search tree that is built from only a two-level action hierarchy (for instance, the tree w.r.t. the GET task in the Taxi domain). In this case the search tree has multiple UCT tree searches as children nodes. We first provide a theorem similar to the Theorem 7 in (Kocsis and Szepesvári 2006), which gives the bias of the estimated expected payoff at the root.

Theorem 1 *Consider H-UCT running on a tree of UCTs of depth H, with branching factor of K. Each UCT is itself a tree of depth T and a branching factor K_0. Then the bias of the estimated expected payoff at the root is*

$$O\left([KHc^2 \ln(n) + K^H + H(TK_0 \ln(n) + K_0^T)]/n\right) \quad (6)$$

where c is defined in Eq.5. Please see the supplementary material for a proof. For a sketch of proof, computing a bias at a node should incorporate the bias from its children nodes and the bias from doing UCT planning from that node. Therefore, we can easily extend the computation of the bias to cases of general H-UCT.

In the case of H-POMCP, the proof of the derived belief SMDP from a POSMDP can be similarly done as Silver (Silver and Veness 2010) did, using the transformation from a POMDP to a belief MDP.

Algorithm 4 SIMULATE(state s, belief b, task i, hierarchy $\mathcal{H}$, tree T, depth h)

$steps = 0$ *//return the number of steps executed by task i*
$\mathbf{o} = null$
if i is primitive **then**
  $(s', o, r, 1) \sim$ Generative-Model(s, i) *//sampling*
  $steps = 1$
  return $(s', o, r, steps)$
else
  if (i terminates) **or** ($\gamma^h < \epsilon$) **then**
    return $(s^i_{terminal}, obs^i_{terminal}, 0, 0)$
  if $b \notin T$ **then**
    for $\forall a \in A$ **do**
      $T(ba) \leftarrow (N_{init}(i, b, a), Q_{init}(i, b, a), \emptyset)$
    return ROLLOUT$(s, i, \mathcal{H}, h)$
  $a = \arg\max_{a'} Q(i, b, a') + c\sqrt{\frac{\log N(i,b)}{N(i,b,a')}}$
  $[s', \mathbf{obs}, r, k] =$ SIMULATE$(s, b, a, \mathcal{H}, T, 0)$ *//compute a reward term*
  $steps = steps + k$
  $\mathbf{o} = [\mathbf{o}; \mathbf{obs}]$
  $[\cdot, \cdot, R', n] =$ SIMULATE$(s', \tau(b, a, \mathbf{o}), i, \mathcal{H}, T, h + k)$ *//compute completion term*
  $R = r + \gamma^k \cdot R'$ *//one-step look-ahead of task i*
  $steps = steps + n$
  $N(i, b) = N(i, b) + 1$
  $N(i, b, a) = N(i, b, a) + 1$
  $Q(i, b, a) = Q(i, b, a) + \frac{R - Q(i,b,a)}{N(i,b,a)}$
  return $[s_{next}, \mathbf{o}, R, steps]$ *// return state, subsumed observations, reward, and taken steps after taking action i*

Theorem 2 *Given a POSMDP $\{\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{P}, \mathcal{R}\}$, and a belief SMDP $\{\mathcal{B}, \mathcal{A}, \tilde{\mathcal{P}}, \tilde{\mathcal{R}}\}$ ($\mathcal{B}$ is the belief space) defined as $\tilde{\mathcal{P}}(b, a, \tau(b, a, \mathbf{o}, k), k) = \sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} b(s)\mathcal{P}(s, a, s', \mathbf{o}, k)$ and reward function $\tilde{\mathcal{R}}(b, a) = \sum_{s \in \mathcal{S}} R(s, a)$. Then, for all policies π, the value function of the belief SMDP is equal to the value function of the POSMDP: $\tilde{V}^\pi(b) = V^\pi(b), \forall b \in \mathcal{B}$.*

Please see the supplementary material for a proof.

Experiments

For experiments, we extensively evaluate H-UCT on various settings: MDP, POMDP, Bayesian RL which are all typical examples successfully using UCT.

Taxi Domain: Hierarchical Planning in MDP

We first evaluate H-UCT on a simple MDP domain, called 5×5 *Taxi* (625 states) which was first introduced by Dietterich (Dietterich 2000). We also created a more challengingly large 10×10 Taxi domain (2500 states) in a way that: Each cell in the 5×5 domain is divided into 4 smaller cells. This problem still has four similarly positioned landmarks. The reward functions are -1, -20, and 40 corresponding respectively to movement costs, wrong pickup or putdown, and correct putdown. This new task uses the same action hierarchy as in the original Taxi domain.

In Figure 2 we report the means and first deviations for both domains over 10 runs of the averaged cumulative return of 50 episodes varying on the number of samples. This measure can also be interpreted as an average of the total regret in each episode if the agent does not act optimally. The discount factor γ is set to 0.99. The number of samples is set to a wide range from 100 to 1000, which somehow resembles limited computation time given for each online planning step. The results show H-UCT consistently outperforms flat UCT. This is because that H-UCT search guided by an action hierarchy apparently have higher possibility of visiting high reward states in the very far future.

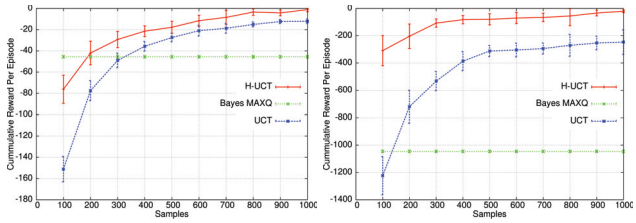

Figure 2: Results on MDP Taxi: (left) 5×5; (right) 10×10.

Figure 2 also includes a comparison to the result of Bayes-MAXQ (Cao and Ray 2012) when the subroutine MAXQ is allowed to make use of the true transition model (for more detail of this algorithm we refer reader to (Cao and Ray 2012)). For this we ran it long enough (at least with the same amount of time given to UCT and H-UCT) to make sure all value functions converged, then computed the average cumulative return of 50 subsequent episodes.

Taxi Domain: Bayesian Hierarchical RL Setting

We evaluate H-UCT in a Bayesian hierarchical RL (BHRL) setting on two Taxi domains. This problem was recently studied by (Cao and Ray 2012) who proposed an algorithm, named Bayesian MAXQ, combining the MAXQ framework and Bayesian RL. BHRL is a BRL problem with the assumption of being given an action hierarchy, so that we can vastly reduce computational cost and speed up learning in BRL. In the BHRL setting the agent works in an unknown environment; for learning it maintains a belief over the dynamics of the environment. With the additional assumption of an action hierarchy the RL agent is expected to find an optimal policy much more quickly than a flat RL agent.

As BRL naturally leads to a POMDP formulation we can apply our methods also to BHRL. The details for this are given in our previous work (Vien, Ngo, and Ertel 2014; Vien et al. 2014).

Table 1: Total time in 5×5 Taxi.

Algorithms	Time (s)
H-UCT	579.72 ± 11.565
UCT	635.551 ± 21.418
Bayesian MAXQ	46.697 ± 5.668
MAXQ	0.166 ± 0.002

We compare H-UCT to the method of (Guez, Silver, and Dayan 2012; Vien and Ertel 2012), which are flat UCT solvers for BRL, and to the Bayesian MAXQ method (Cao and Ray 2012), which is a Bayesian model-based method for hierarchical RL. We represent beliefs using Dirichlet distributions as in previous work (Cao and Ray 2012; Guez, Silver, and Dayan 2012). For UCT and H-UCT we use 1000 samples (5×5 map) and 2000 episodes (10×10 map), a discount factor of 0.995, and report a mean and its first deviation of the average cumulative rewards of 100 episodes (5×5 map) and 200 episodes (10×10 map) from

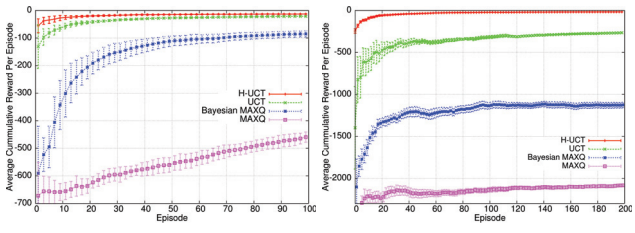

Figure 3: Results on BHRL Taxi: (left) 5×5; (right) 10×10.

10 runs. For Bayesian MAXQ and MAXQ we used the same setting as in (Cao and Ray 2012).

As shown in Figure 3, Bayesian MAXQ gave much worse average cumulative rewards compared to UCT and H-UCT, because it used the ϵ-greedy strategy to select actions, which have been known having infinite sample complexity. Therefore, during initial episodes it suffered from many wrong actions of high costs, i.e. wrong Pickup and Putdown actions. Moreover, the results also show that Bayesian MAXQ does not thoroughly exploit planning when maintaining posterior models. The way it samples from the posterior model can be considered as doing planning, but the value functions learnt incrementally from sampled models via the MAXQ subroutine are not tuned after observing a real transition. UCT and H-UCT prune the tree and do search from the next updated belief after observing a real transition. Again, the results show that H-UCT significantly outperforms UCT given limited computation resources. When given more resource, UCT asymptotically performs closer to the level of H-UCT. The performance improvement in the large 10×10 Taxi domain is much more dramatic, which additionally reinforces efficiency of integrating action decomposition.

Additionally, we report the total time (in seconds) needed to run 100 episodes on the 5×5 domain for each algorithm in Table 1. The reported numbers are the mean and its first deviation over 10 runs. As Bayesian MAXQ did not fully exploit planning, it used moderate computational time. Though we have increased planning time to the same given amount of H-UCT, Bayesian MAXQ failed in getting performance improvement due to the above mentioned non-tuning issue. H-UCT and UCT algorithms work similarly by simulating many rollouts. However H-UCT has slightly less planning time, as its rollouts sometimes terminate early. This is because of macro actions helping the rollouts to reach more promising state regions. In very large domains with long horizon reward this effect should become even more pronounced.

Hierarchical POMDP Domain

Finally we evaluate our methods on the partially observable Pac-Man domain, named *pocman*, which was first introduced in (Silver and Veness 2010). The pocman agent lives in a 17×19 gridworld and his task is to find and eat all food pellets and special power pills randomly scattered. The agent can choose four navigation actions. There are four ghosts also roaming the environment with particular strategies. The pocman agent receives a reward: -1.0 for each move, $+10$ for eating a pellet, $+25$ for eating a ghost, and -100 when dying (eaten by a ghost). The agent death terminates an episode. Observations are encoded by ten bits indicating whether there are walls, adjacent food, and ghosts.

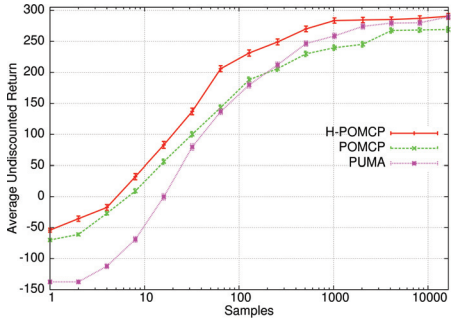

Figure 4: Results on POMDP Pac-Man domain.

This is a very large POMDP problem which has 2^{10} observations, and about 10^{56} states (Silver and Veness 2010). Therefore, planning has large complexity due to the large observation branching factor. To allow the comparison between H-UCT and UCT we designed a basic set of macro actions which merely depends on instant observations. The macro action set consists of moves in one direction of *go-left,go-right,go-down,go-up* until observing free ways perpendicular to the current direction. That means the agent can keep moving forward until there are turning possibilities. The termination of these macro actions are naturally defined depending on the four observation bits corresponding to walls, in which there is bit 1 (no wall) in the perpendicular direction. We use particles to represent beliefs and use the similar particle invigoration method as in (Silver and Veness 2010). We compare H-POMCP against flat POMCP and PUMA (He, Brunskill, and Roy 2010). For a fair comparison, we implemented PUMA to use the same set of macro actions and particles without using its macro action generation function, as it is not straightforward to generalize such a function for large state space problems. H-POMCP and PUMA use similar execution fashion, which executes one primitive action then re-planning. The reported numbers in Figure 4 are the mean and the mean's standard deviation of average undiscounted returns per episode over 500 runs. H-POMCP achieves a return of approximately 291, while flat POMCP can achieve 269 at best, PUMA got a best return of 288. The gap is more dramatic when limited planning resource is given. Remarkably, H-POMCP with at least 512 samples can easily outperform the flat POMCP with even 2^{14} samples.

Discussion

We have proposed a principled method to integrate a task hierarchy into MCTS algorithms and specifically extended UCT and POMCP, named H-UCT and H-POMCP. The new framework for hierarchical planning aims to mitigate two challenging problems in planning: *curse of dimensionality* and *curse of history*. In all benchmarks we considered for UCT and POMCP we demonstrated the efficiency of the new framework. Though H-UCT looks for only recursively optimal policies, H-UCT can be adjusted to look for a hierarchically optimal policy if formulating parametrized pseudo rewards of subtasks into partially observable space. In general, the technique used to extend UCT to H-UCT can be similarly be exploited to extend other MCTS methods to become hierarchical.

Acknowledgment

We would like to thank Csaba Szepesvari for constructive comments over the very first draft. This work was supported by the EU-ICT Project 3rdHand 610878.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2-3):235–256.

Bai, A.; Wu, F.; and Chen, X. 2012. Online planning for large MDPs with MAXQ decomposition. In *AAMAS*, 1215–1216.

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

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

Cao, F., and Ray, S. 2012. Bayesian hierarchical reinforcement learning. In Bartlett, P.; Pereira, F.; Burges, C.; Bottou, L.; and Weinberger, K., eds., *NIPS*, 73–81.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games*, 72–83.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res. (JAIR)* 13:227–303.

Finnsson, H., and Björnsson, Y. 2008. Simulation-based approach to general game playing. In *AAAI*, 259–264.

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

Guez, A.; Silver, D.; and Dayan, P. 2012. Efficient Bayes-adaptive reinforcement learning using sample-based search. In *NIPS*, 1034–1042.

Hansen, E. A., and Zhou, R. 2003. Synthesis of hierarchical finite-state controllers for POMDPs. In *ICAPS*, 113–122.

He, R.; Brunskill, E.; and Roy, N. 2010. PUMA: Planning under uncertainty with macro-actions. In *AAAI*.

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

Lim, Z. W.; Hsu, D.; and Sun, L. W. 2011. Monte Carlo value iteration with macro-actions. In *NIPS*, 1287–1295.

Lorentz, R. J. 2008. Amazons discover Monte-Carlo. In *Computers and Games*, 13–24.

Ma, H., and Pineau, J. 2014. Information gathering and reward exploitation of subgoals for POMDPs. In *ICAPS*.

Pineau, J. 2004. *Tractable Planning Under Uncertainty: Exploiting Structure*. Ph.D. Dissertation, Robotics Institute, Carnegie Mellon University.

Powley, E. J.; Whitehouse, D.; and Cowling, P. I. 2013. Monte Carlo tree search with macro-actions and heuristic route planning for the multiobjective physical travelling salesman problem. In *CIG*, 1–8.

Silver, D., and Veness, J. 2010. Monte-Carlo planning in large POMDPs. In *NIPS*, 2164–2172.

Theocharous, G., and Kaelbling, L. P. 2003. Approximate planning in POMDPs with macro-actions. In *NIPS*.

Toussaint, M.; Charlin, L.; and Poupart, P. 2008. Hierarchical POMDP controller optimization by likelihood maximization. In *UAI*, 562–570.

Vien, N. A., and Ertel, W. 2012. Monte Carlo tree search for Bayesian reinforcement learning. In *11th International Conference on Machine Learning and Applications, ICMLA, Boca Raton, FL, USA, December 12-15, 2012. Volume 1*, 138–143.

Vien, N. A.; Ngo, H. Q.; Lee, S.; and Chung, T. 2014. Approximate planning for Bayesian hierarchical reinforcement learning. *Applied Intelligence* 41(3):808–819.

Vien, N. A.; Ngo, H.; and Ertel, W. 2014. Monte Carlo Bayesian hierarchical reinforcement learning. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, 1551–1552.

White, C. C. 1976. Procedures for the solution of a finite-horizon, partially observed, semi-Markov optimization problem. *Operations Research* 24(2):348–358.