Subgoal-Based Temporal Abstraction in Monte-Carlo Tree Search

Thomas Gabor , **Jan Peter** , **Thomy Phan** , **Christian Meyer** and **Claudia Linnhoff-Popien**
LMU Munich
thomas.gabor@ifi.lmu.de

Abstract

We propose an approach to general subgoal-based temporal abstraction in MCTS. Our approach approximates a set of available macro-actions locally for each state only requiring a generative model and a subgoal predicate. For that, we modify the expansion step of MCTS to automatically discover and optimize macro-actions that lead to subgoals. We empirically evaluate the effectiveness, computational efficiency and robustness of our approach w.r.t. different parameter settings in two benchmark domains and compare the results to standard MCTS without temporal abstraction.

1 Introduction

Markov Decision Processes (MDPs) provide the formal foundation for many current approaches to planning and decision making [Russell and Norvig, 2010; Sutton and Barto, 2018]. In that context, making decisions at multiple levels of abstraction has been studied as a challenging problem [Barto and Mahadevan, 2003; Bai *et al.*, 2012; Vien and Toussaint, 2015]. One approach is to compress the search space in the temporal dimension by introducing macro-actions at a coarser resolution of time, which is motivated by human decision making, where reasoning about the problem takes place at different levels [Botvinick *et al.*, 2009]. This approach, which is known as *temporal abstraction* or *hierarchical planning* [Mausam and Kolobov, 2012; Sutton and Barto, 2018], can accelerate direct reinforcement learning and planning algorithms as it decomposes the domain into a hierarchy of *subtasks* or *subgoals* that can be addressed individually [Dietterich, 2000; He *et al.*, 2010; Vien and Toussaint, 2015]. Primitive action sequences to achieve such subgoals can be combined into *macro-actions* to enable efficient decision making at higher levels.

Many approaches rely on extensive domain knowledge by assuming all subgoals or macro-actions per state to be fully known beforehand [Parr and Russell, 1998; Sutton *et al.*, 1999; Dietterich, 2000]. However, a complete specification of subgoals or macro-actions is generally infeasible for domains with large state spaces and many subgoals.

For many problems, it is easy to determine whether a given state is desirable as a subgoal, but it is infeasible to specify all possible follow-up subgoals or macro-actions for each state in advance. Especially in online planning, subgoals and macro-actions should be determined locally for each state, and only when needed, to meet real-time requirements.

In this paper, we propose an approach to general subgoal-based temporal abstraction in Monte Carlo Tree Search (MCTS). We assume the availability of a subgoal predicate (in addition to a generative model, i.e., an MDP), which we integrate into the expansion step of MCTS to automatically discover and optimize macro-actions that lead to subgoals (cf. Section 4). Empirical evaluation shows that subgoal-based MCTS is more efficient than standard MCTS with little loss w.r.t effectiveness in the gridworld domain or even considerable gain in Tetris (cf. Section 5).

2 Background

2.1 Markov Decision Processes

An *MDP* is defined as a tuple $F = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a (finite) set of states, $\mathcal{A}$ is the (finite) set of primitive actions, $\mathcal{P}(s_{t+1}|s_t, a_t)$ is the transition probability, $\mathcal{R}(s_t, a_t) \in \mathbb{R}$ is the reward, and $\gamma \in [0, 1]$ is the discount factor [Puterman, 2014]. We always assume that $s_t, s_{t+1} \in \mathcal{S}$, $a_t \in \mathcal{A}$, and $r_t = \mathcal{R}(s_t, a_t)$, where s_{t+1} is reached after executing the action a_t in state s_t at time step t. Most importantly, we assume MDPs to be *discrete* and to have *deterministic* state transitions, i.e., $\mathcal{P}(s_{t+1}|s_t, a_t) \in \{0, 1\}$ for all s_{t+1}, s_t, a_t.

The goal is to find a *policy* $\pi : \mathcal{S} \to \mathcal{A}$ which maximizes the expectation of return G_t at state s_t for a horizon H:

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

A policy π can be evaluated with a *state value function* $V^\pi(s_t) = \mathbb{E}_\pi[G_t|s_t]$, i.e., the expected return at state s_t. $Q^\pi(s_t, a_t) = \mathbb{E}_\pi[G_t|s_t, a_t]$ is the *action-value function*, i.e., the expected return when executing action a_t in state s_t.

π is optimal iff $V^\pi(s_t) \geq V^{\pi'}(s_t)$ for all $s_t \in S$ and all policies π'. We denote the optimal policy by π^* and the value functions by $V^{\pi^*} = V^*$ and $Q^{\pi^*} = Q^*$ respectively.

2.2 Monte Carlo Online Planning

Planning searches for a (near-)optimal policy, given a model $\hat{F}$ of the actual environment F. $\hat{F}$ usually provides $\mathcal{P}$ and

$\mathcal{R}$ of the underlying MDP. *Global planning* searches the whole state space $\mathcal{S}$ to find π^* [Bellman, 1957; Weinstein and Littman, 2013]. An example is *value iteration*, which computes the optimal value function according to the *Bellman equation* [Bellman, 1957]:

$$V^*(s_t) = \max_{a_t \in \mathcal{A}} \{ r_t + \gamma \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_{t+1}|s_t, a_t) V^*(s_{t+1}) \} \quad (2)$$

Local planning only regards the current state s_t and possible future states to find a policy π_t [Weinstein and Littman, 2013]. We focus on local *Monte Carlo planning*, where $\hat{F}$ is a *generative* or *forward model*, which can be used for simulation-based planning [Kocsis and Szepesvári, 2006; Weinstein and Littman, 2013] without reasoning about explicit probability distributions. Given s_t and a_t, $\hat{F}$ provides a sample $\langle s_{t+1}, r_t \rangle \sim \hat{F}(s_t, a_t)$. We also focus on *online planning*, where planning is performed at every time step t with a horizon H and a computation budget n_b. The recommended action a_t is executed in s_t, which thereby transitions to s_{t+1}. If s_{t+1} is no terminal state, this procedure is repeated.

Monte Carlo Tree Search (MCTS) is a popular approach to Monte Carlo Planning and has been applied to a wide range of challenging environments [Rubin and Watson, 2011; Silver *et al.*, 2017]. MCTS incrementally builds up a search tree, which stores the visit counts $N(s_t)$, N_{s_t,a_t}, and the values $V(s_t)$ and $Q(s_t, a_t)$ for each simulated state and action respectively. MCTS iteratively executes the following four steps until a computation budget n_b has run out:

Selection Starting from the root node as current state s_0, the search tree is traversed by selecting nodes with a *tree policy* π_{tree} until a leaf state node s_t is reached.

Expansion The leaf state node s_t is expanded by a new node representing the next state s_{t+1}, which is reached after simulating a random action a_t in $\hat{F}$.

Simulation A rollout using a *rollout policy* $\pi_{rollout}$ is performed from s_{t+1} until a maximum search depth H or a terminal state is reached.

Backup The observed rollout rewards are accumulated to G_t (Eq. 1) and used to update the value estimates and visit counts of every node in the simulated path.

Upper Confidence bounds applied to Trees (UCT) uses $\pi_{tree}(s_t) = \text{argmax}_{a_t \in \mathcal{A}}\, \mathit{UCB1}(s_t, a_t)$ with *UCB1* being defined by [Auer *et al.*, 2002; Kocsis and Szepesvári, 2006]:

$$\mathit{UCB1}(s_t, a_t) = Q(s_t, a_t) + c\sqrt{\frac{2log(N(s_t))}{N(s_t, a_t)}} \quad (3)$$

where c is an exploration constant. *UCB1* balances between *exploration* and *exploitation* of actions. Exploration is encouraged by the second term multiplied with c and tries out actions to estimate action-values more accurately. The more a_t is selected in s_t the smaller the exploration term becomes and the more exploitation is encouraged, which greedily uses the action with the currently highest action-value $Q(s_t, a_t)$. UCT has been shown to converge to the optimal best-first tree, given infinite computation [Kocsis and Szepesvári, 2006].

3 Related Work

Temporal abstraction methods summarize temporal sequences of primitive actions into *macro-actions* by dividing the original goal into *subgoals*, for which the planner can generate plans individually and independently [Kaelbling, 1993; Barto and Mahadevan, 2003; Solway *et al.*, 2014].

There exist different frameworks for decision making using temporal abstraction: [Sutton *et al.*, 1999] proposed *options*, where each option o has an internal policy π_o which can be selected from specific states. If option o is selected, then π_o is executed until a termination condition is met. [Dietterich, 2000] proposed *MAXQ Value Function Decomposition* by defining *subtasks*, with each subtask $m_i \in \mathcal{M}$ having a *pseudo-reward function*. A *subtask policy* has to be computed for each subtask. In both cases, there is a high-level policy π, which has to select a lower level policy according to the corresponding option or subtask. These frameworks assume detailed prior knowledge (e.g. internal policy, reward function) about each subgoal, which is infeasible for very large macro-action spaces and many subgoals.

[He *et al.*, 2010] proposed *Planning under Uncertainty with Macro-Actions (PUMA)*, which generates macro-actions from automatically discovered subgoals given a global subgoal distribution. Each macro-action $\overline{a}_t = \langle a_t, a_{t+1}, ..., a_{t+L-1} \rangle$ represents an open-loop sequence of primitive actions. Macro-actions are determined by iteratively finding open-loop plans of length L, which can reach a randomly sampled subgoal state from a given state s_t. The action-value $Q(s_t, \overline{a}_t)$ is estimated via Monte Carlo simulation to determine the action to be executed. Our approach is closely related to PUMA: Each macro-action leads to a subgoal and is generated locally for each state during planning. However, PUMA assumes a fixed length and a fixed number of macro-actions, which is highly domain-dependent. Instead, our approach only requires a subgoal predicate, which can determine whether a given state is a subgoal or not. It does not need a pre-defined global distribution of subgoals.

4 Subgoal-based Temporal Abstraction

4.1 Terminology

Given a deterministic and discrete MDP, we define a *macro-action* $m_t = \langle a_t, ..., a_{t+N-1} \rangle$ as an open-loop sequence of primitive actions $a_i \in \mathcal{A}$. The *macro-action space* $\mathcal{M} = \mathcal{A}^+$ is the set of all non-empty sequences over $\mathcal{A}$. The macro-level generative model $\overline{F} : \mathcal{S} \times \mathcal{M} \to \mathcal{S}$ determines the successor state $\overline{F}(s_t, m_t) = s_{t+|m_t|}$ after performing $m_t \in \mathcal{M}$ with length $|m_t|$ in state s_t. The reward function for macro-actions $m_t \in \mathcal{M}$ is defined by:

$$\overline{\mathcal{R}}(s_t, m_t) = \sum_{k=0}^{|m_t|-1} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k}) \quad (4)$$

We define a *macro-level policy* $\overline{\pi} : \mathcal{S} \to \mathcal{M}$ to select macro-actions, which can be evaluated with a value function $V^{\overline{\pi}}(s_t) = \overline{\mathcal{R}}(s_t, \overline{\pi}(s_t)) + \gamma^{|\overline{\pi}(s_t)|} V^{\overline{\pi}}(\overline{F}(s_t, \overline{\pi}(s_t)))$.

We define a set of subgoals $\mathcal{G} \subseteq \mathcal{S}$ with $\mathcal{G} = \{ s_t \in \mathcal{S} \mid g(s_t) = 1 \}$, where $g : \mathcal{S} \to \{0, 1\}$ is a *subgoal predicate* returning 1, if s_t is a subgoal, and 0 otherwise. $\mathcal{G}(s_t)$

defines the set of subgoals directly reachable from state s_t with any macro-action. $\mathcal{M}(s_t, g_t)$ contains all macro-actions that terminate in subgoal $g_t \in \mathcal{G}$ when performed in state s_t and $\mathcal{M}(s_t)$ represents all macro-actions which can directly reach any subgoal $g_t \in \mathcal{G}(s_t)$ from state s_t:

$$\mathcal{M}(s_t) = \bigcup_{g_t \in \mathcal{G}(s_t)} \mathcal{M}(s_t, g_t) \tag{5}$$

$\mathcal{M}^*(s_t) \subseteq \mathcal{M}(s_t)$ is the set of macro-actions m_t for state s_t which locally maximize the reward $\overline{\mathcal{R}}(s_t, m_t)$:

$$\mathcal{M}^*(s_t) = \{ \operatorname*{argmax}_{m_t \in \mathcal{M}(s_t, g_t)} \overline{\mathcal{R}}(s_t, m_t) \mid g_t \in \mathcal{G}(s_t) \} \tag{6}$$

By assuming $\gamma = 1$, we can show that the hierarchically optimal value function $V^{\overline{\pi}^*}$ is preserved when replacing $\mathcal{M}(s_t)$ with the locally optimized set $\mathcal{M}^*(s_t)$:

$$\begin{aligned} V^{\overline{\pi}^*}(s_t) &\overset{(2)}{=} \max_{m_t \in \mathcal{M}(s_t)} \{\overline{r}_t + V^{\overline{\pi}^*}(\overline{F}(s_t, m_t))\} \\ &\overset{(5)}{=} \max_{g_t \in \mathcal{G}(s_t)} \{\max_{m_t \in \mathcal{M}(s_t, g_t)} \{\overline{r}_t + V^{\overline{\pi}^*}(g_t)\}\} \\ &= \max_{g_t \in \mathcal{G}(s_t)} \{(\max_{m_t \in \mathcal{M}(s_t, g_t)} \{\overline{r}_t\}) + V^{\overline{\pi}^*}(g_t)\} \\ &\overset{(6)}{=} \max_{m_t \in \mathcal{M}^*(s_t)} \{\overline{r}_t + V^{\overline{\pi}^*}(\overline{F}(s_t, m_t))\} \end{aligned} \tag{7}$$

where $\overline{r}_t = \overline{\mathcal{R}}(s_t, m_t)$.

4.2 Generating Macro-Actions

We now describe an approach to approximate $\mathcal{M}^*(s_t)$ for any given state s_t. $\hat{\mathcal{M}}(s_t) \approx \mathcal{M}^*(s_t)$ is generated incrementally by randomly sampling a macro-action m_t from $\mathcal{M}(s_t)$. If m_t reaches a previously undiscovered subgoal, it is added to $\hat{\mathcal{M}}(s_t)$. If a subgoal is rediscovered, the existing macro-action m'_t is replaced with m_t, when $\overline{\mathcal{R}}(s_t, m_t) > \overline{\mathcal{R}}(s_t, m'_t)$.

The sampling is regarded as a Bernoulli trial, where p is the *action coverage* representing the fraction of already discovered macro-actions of $\mathcal{M}^*(s_t)$. Let $\boldsymbol{X} = \langle X_1, ..., X_n \rangle$ be a Bernoulli process according to the Bernoulli distribution $\mathcal{B}(p)$ with $X_i \sim \mathcal{B}(p)$. $X_i = 1$, if the sampled macro-action is already known, and $X_i = 0$ otherwise. The likelihood $L(p = p_0|\boldsymbol{X})$ that the unknown action coverage p equals p_0 given the observations $\boldsymbol{X}$ is defined by $L(p = p_0|\boldsymbol{X}) = \binom{n}{k} p^k (1-p)^{n-k}; k = \sum_i^n X_i$. If all trials were successful ($k = n$), then $L(p = p_0|\boldsymbol{X}) = p^n$. To test if $p \geq p_0$, we define a statistical test $\psi : \{0,1\}^n \rightarrow \{0,1\}$ with null hypothesis $H_0 : p < p_0$ and alternative hypothesis $H_1 : p \geq p_0$:

$$\psi(\boldsymbol{X}) = \mathbb{I}(L(p = p_0|\boldsymbol{X}) \leq \alpha) = \mathbb{I}(n > log_{p_0}\alpha) \tag{8}$$

where $\alpha \in [0,1]$ is the *tolerated error*.

If we consecutively sample n macro-actions, whose subgoals are already known, and $n > log_{p_0}\alpha$, we will assume that a coverage of at least p_0 is achieved. Increasing the desired action coverage p_0 leads to a higher percentage of discovered macro-actions of $\mathcal{M}^*(s_t)$ and to higher quality of each macro-action in $\hat{\mathcal{M}}(s_t)$ at the cost of more trials.

Algorithm 1 Expansion with Macro-Action Generation

procedure *ExpansionWithMA*$(\hat{F}, g, t, s_t, H, p_0, \alpha)$
  $n \leftarrow 0$
  repeat ▷ discover macro actions until confident
    $s_{t+1} \leftarrow s_t$
    $m_t \leftarrow \langle\rangle$
    repeat ▷ sample until macro state or horizon
      $a_t \sim \mathcal{A}$
      $m_t \leftarrow m_t +\!\!+ \langle a_t \rangle$
      $s_{t+1} \leftarrow \hat{F}(s_{t+1}, a_t)$
    until $(g(s_{t+1}) = 1) \vee (t + |m_t| \geq H)$
    if $\exists m'_t \in \hat{\mathcal{M}}(s_t) : \overline{F}(s_t, m'_t) = s_{t+1}$ **then**
      $n \leftarrow n + 1$ ▷ macro state rediscovered
      if $\overline{\mathcal{R}}(s_t, m_t) > \overline{\mathcal{R}}(s_t, m'_t)$ **then**
        $\hat{\mathcal{M}}(s_t) \leftarrow (\hat{\mathcal{M}}(s_t) \setminus \{m'_t\}) \cup \{m_t\}$
    else ▷ new macro state discovered to expand
      $\hat{\mathcal{M}}(s_t) \leftarrow \hat{\mathcal{M}}(s_t) \cup \{m_t\}$
      return m_t
  until $n > log_{p_0}\alpha$
  fullyExpanded$(s_t) \leftarrow$ *true*
  return *nil*

4.3 Integration with MCTS

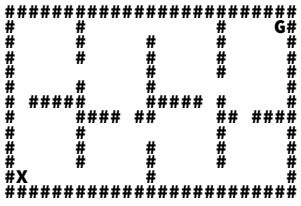
The approach described above can be easily integrated into MCTS, since actions are iteratively explored for each state. Instead of pre-computing the whole set of macro-actions, we only need to find one new macro-action in the expansion step. This saves computation time, since macro-actions are only generated on demand according to the selection policy.

The complete formulation of the modified expansion step is given in Algorithm 1, where $\hat{F}$ is the generative model of the MDP, g is the subgoal predicate, t is the current time step, s_t is the current state, H is the planning horizon, p_0 is the desired action coverage, and α is the tolerated error.

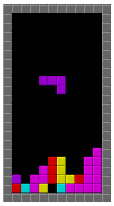
Our approach constructs a search tree for deterministic MDPs, where the root node represents the current state, all other nodes represent subgoal states, and links represent macro-actions. Our expansion step updates the macro-action set $\hat{\mathcal{M}}(s_t)$ for the current leaf node representing s_t. If the desired action coverage p_0 has been achieved, the node of s_t is considered as *fully expanded* and returns *nil*. The expansion step will not be invoked for fully expanded nodes.[1]

The advantage of this approach is that only subgoals *directly reachable* from a given state are regarded. The set of available subgoals per state is only computed *on demand*, when the state is actually visited during tree search, and is not required to be specified beforehand. Our approach enables planning with macro-actions m_t of *arbitrary length* ($1 \leq |m_t| \leq H - t$), thus being more flexible than PUMA [He *et al.*, 2010]. Note that the *Backup* step has to be adjusted to consider the discount of subgoal rewards $\overline{\mathcal{R}}(s_t, m_t)$ according to the length of each macro-action m_t (Eq. 4).

[1]For a complete description of that integration in pseudo-code, please refer to github.com/hugo-voodo/temporal-abstraction/blob/master/supplement.pdf.



(a) Gridworld



(b) Tetris

Figure 1: (a) *Gridworld* of [Bai *et al.*, 2016] with 'X' representing the agent's initial position, 'G' representing the goal and '#' representing obstacles. (b) The game *Tetris*.

4.4 Possible Variants and Enhancements

Modes. Similarly to [Bai *et al.*, 2016], our approach can operate in different control modes. In *hierarchical control mode*, the MCTS returns the macro-action as sequence of primitive actions, which is successively applied to the real environment. In this case, planning is only performed at subgoal states, thus reducing the overall computation. In *polling control mode*, the MCTS returns only the first primitive action of the recommended macro-action. This mode can recover from locally poor decisions, since planning is performed at every time step t but requires more computation in total.

Tree Reuse. In MCTS, the computed subtree of the following state can be reused to avoid complete replanning by keeping the learned statistics from the previous planning step. If polling control mode is used, tree reuse cannot be trivially applied to our approach because the state nodes represent subgoal states, while planning can be performed on non-subgoal states in polling mode. Thus, we only regard tree reuse in combination with hierarchical control mode.

Parallelization. There exist many approaches to parallelization of MCTS ranging from *tree parallelization* with mutually exclusive locks to *root parallelization*, where multiple separate trees are generated and searched in parallel [Chaslot *et al.*, 2008; Browne *et al.*, 2012]. We only focus on root parallelization due to the minimal synchronization overhead. We also expect root parallelization to compensate for approximation errors caused by low action coverages, since multiple trees will generate different macro-action sets, thus converging to different trees. Since we focus on deterministic problems, it is most promising to make decisions based on the single best action found by an individual tree.

5 Experiments

5.1 Evaluation Environments

Gridworld is one of the must studied example domains in artificial intelligence [Sutton *et al.*, 1999; Russell and Norvig, 2010; Bai *et al.*, 2016; Sutton and Barto, 2018]. An agent has to navigate in a two-dimensional grid to reach a goal position. The agent is able to move north, east, south and west to adjacent grid cells, but it cannot pass obstacles in the grid. A reward of -0.01 is given at every time step. Reaching the goal gives a reward of 1 and terminates the episode. The setup used in this paper is shown in Fig. 1a. The grid is divided into eight rooms with connecting 'doors'.

The *Tetris* game is another popular research domain in artificial intelligence research [Thiery and Scherrer, 2009; Zhongjie *et al.*, 2011; Scherrer *et al.*, 2015; Jaskowski *et al.*, 2015]. An object called *tetrimono* has to be controlled in a $W_b \times H_b$ board (violet 'L' shape in Fig. 1b), while it is falling to the ground. At every time step, the tetrimono moves down to the next grid cell until it touches the ground (of stacked previous tetrimonos). The agent can rotate the tetrimono left or right, move it left or right, or do nothing. After the tetrimino has fallen down, a reward of $1-(0.5\frac{h_c^2}{H_b^2}+0.5\min\{1, \frac{x}{X}\})$ is given and a new tetrimono appears in the upper center of the board. h_c is the current height of the stack, H_b is the board height, W_b is the board width, x is the number of holes in the tetrimono stack, and $X = \frac{W_b}{4}(H_b - 2)$ is an arbitrary upper bound on stack holes. The game ends with a reward of -1, when the tetrimono stack height exceeds the board height. The goal is to minimize the height of the stacked tetrimonos in the long run. We always set $W_b = 10$ and $H_b = 10$.

5.2 Methods

Online Planning

We implemented different instances of our approach, which we call *Subgoal-MCTS (S-MCTS)*.[2] The polling control mode is used as default mode. S-MCTS-H uses hierarchical control mode and S-MCTS-H-R additionally enables tree reuse. We also implemented UCT of [Kocsis and Szepesvári, 2006], which we refer to as MCTS or as MCTS-R, if tree reuse is enabled. All MCTS implementations use *UCB1* as tree policy and random rollouts. We also implemented random rollout planning, referred to as MC.

For each planning algorithm, we experimented with different settings of tunable parameters.[3] For all experiments, we set $\gamma = 1$, $p_0 = 0.95$, and $\alpha = 0.001$. We implemented root parallelization for all MCTS-based approaches using multithreading to generate multiple trees.

Subgoal Heuristics

For Gridworld, we use a subgoal predicate g returning 1, if the agent position is at a 'door' position with obstacles on opposing sides next to it, and 0 otherwise.

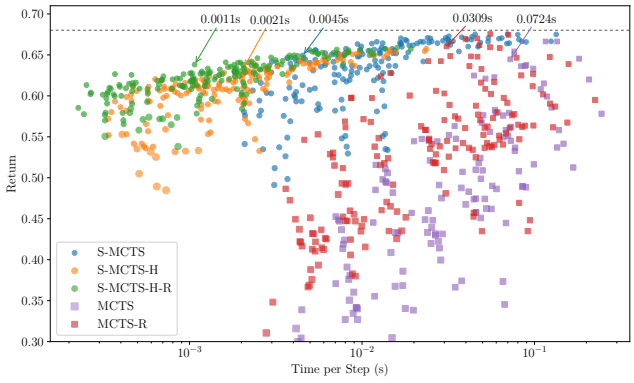
For Tetris, we use a subgoal predicate g, which returns 1, if the current tetrimono has fallen down, and 0 otherwise. Note that while it is easy to determine whether a given state is a subgoal or not (e.g., by checking the y-coordinate of the current tetrimono's position with our subgoal predicate g), it is infeasible to specify all possible subgoals for each tetrimono type beforehand, since there are too many possible combinations of position and rotation per tetrimono type.

5.3 Results

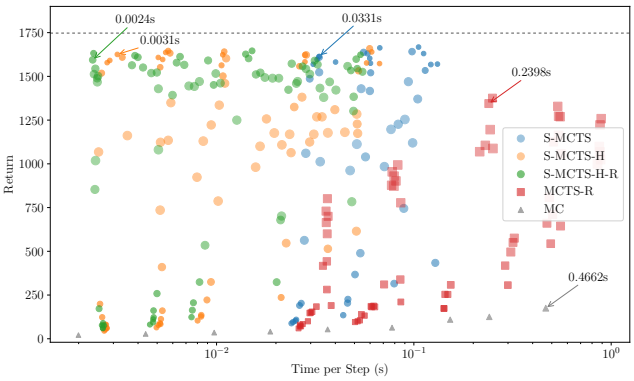
We ran each experiment with different parameter settings. Each setting was run 60 times for a maximum of 1000 time steps in Gridworld or 20000 time steps in Tetris.

[2]The code can be found at github.com/jnptr/subgoal-mcts.

[3]For all parameter configurations, see github.com/hugo-voodo/temporal-abstraction/blob/master/supplement.pdf.



(a) Gridworld



(b) Tetris

Figure 2: Average return per computation time per step of each planning algorithm using different parameter settings. The x-axis uses a logarithmic scale.

Performance-Computation Tradeoff

First, we evaluated the performance and computation time of each algorithm with different parameter settings (Section 5.2). The results are shown in Fig. 2. Each data point represents the average return for an average amount of computation time per step t given a specific parameter setting. The color indicates the algorithm as described in Section 5.2. The dotted horizontal line indicates the maximum possible return for the respective domain (which overestimates for Tetris).

In Gridworld (Fig. 2a), S-MCTS, S-MCTS-H, and S-MCTS-H-R achieve competitive performance compared to MCTS and slightly lower performance than MCTS-R with much less computation time. In Tetris (Fig. 2b), we omitted MCTS due to intractable run times and replaced it with MC. S-MCTS-H and S-MCTS-H-R quickly achieve near-maximum return with S-MCTS-H-R being less sensitive to the concrete parameter setting. S-MCTS is slower than S-MCTS-H and S-MCTS-H-R but achieves the best overall performance. MCTS-R requires much more computation time than Subgoal-MCTS approaches, while being unable to keep up in performance. MC slightly improves with more computation time but is clearly inferior to all other approaches.

In both domains, S-MCTS-H and S-MCTS-H-R are generally faster than S-MCTS, since computation only takes place at subgoal states. Still, both approaches are able to achieve competitive performance compared to S-MCTS. S-MCTS-H and S-MCTS-H-R also seem to be less sensitive to the parameter settings than the other approaches.

Action Coverage and Parallelization

We also evaluated different combinations of desired action coverages p_0 and tree counts in the Tetris domain. The results are shown in Fig. 3 for S-MCTS, S-MCTS-H, and S-MCTS-H-R. Each plot shows the average return of each algorithm with the best parameter setting (Section 5.2), when using a particular number of trees, which are generated in parallel. The dotted horizontal line indicates the maximum possible return. Increasing p_0 generally leads to increasing return and reduces the variance for all settings. Using a large number of trees can compensate for low desired action coverages p_0, generally leading to higher returns, while requiring less time per step. In case of S-MCTS and S-MCTS-H (Fig. 3a and 3b), all settings achieve similar performance, when the desired action coverage is sufficiently large ($p_0 = 0.95$). S-MCTS-H-R (Fig. 3c) has more variance in its returns, but requires much less time, when using many trees.

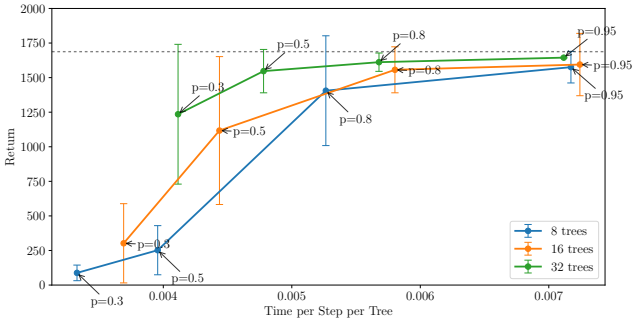
Robustness w.r.t. Subgoal Heuristics

Finally, we evaluated the robustness of our approach w.r.t. different subgoal heuristics for the Gridworld domain with different parameter settings (Section 5.2). The 'exact definition' identifies subgoals by checking, if there are obstacles on opposing sides next to the agent, indicating a 'door' in the grid. A coarser heuristic ($|\mathcal{A}(s_t)| \leq N_a$) checks, if the number of legal actions $|\mathcal{A}(s_t)|$ at the current state is N_a at most. If $N_a = 3$, e.g., then positions next to walls are regarded as subgoals as well. The results are shown in Fig. 4 for S-MCTS, S-MCTS-H, and S-MCTS-H-R. Each data point represents the average return for an average amount of computation time per step t given a specific parameter setting. The color indicates the used subgoal heuristic. The dotted horizontal line indicates the maximum return for the Gridworld domain. When using the heuristic $|\mathcal{A}(s_t)| \leq 2$, then all approaches perform slightly worse than the 'exact definition', while being similarly robust w.r.t. the parameter setting. However, when using the heuristic $|\mathcal{A}(s_t)| \leq 3$, then the performance is generally worse and all approaches are much more sensitive to the parameter setting, while requiring significantly more computation time. Hierarchical control mode and tree reuse seem to slightly improve the robustness of Subgoal-MCTS w.r.t. the parameter setting, while not having a general impact on the overall performance.

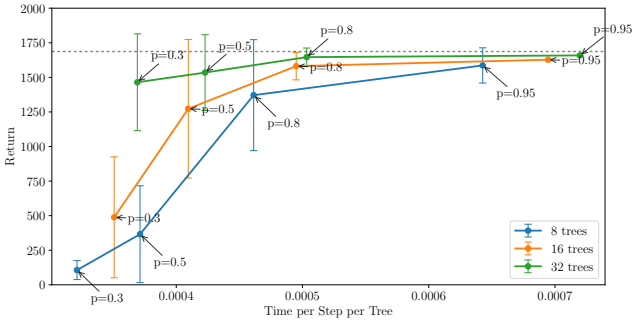
6 Discussion

We proposed an approach to general subgoal-based temporal abstraction in MCTS. Our approach approximates a set of macro-actions locally for each state only requiring a generative model and a subgoal predicate.

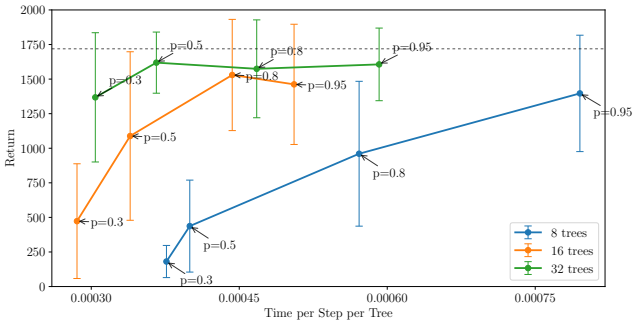
Our experiments show that S-MCTS and its variants are competitive against standard MCTS in terms of performance-computation tradeoff. While all variants of S-MCTS perform slightly worse in the Gridworld domain, they are able to outperform MCTS-R in Tetris, while generally requiring much less computation time than MCTS in all domains.



(a) Subgoal-MCTS in polling control mode (S-MCTS)

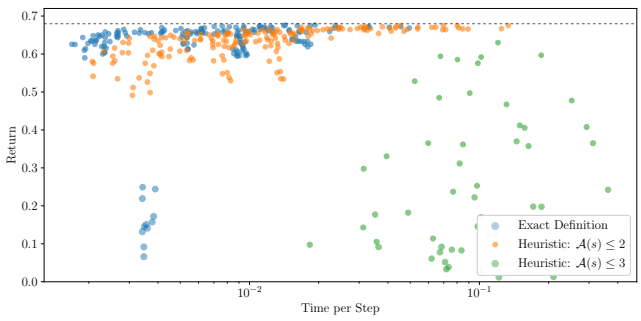


(b) Subgoal-MCTS in hierarchical control mode (S-MCTS-H)

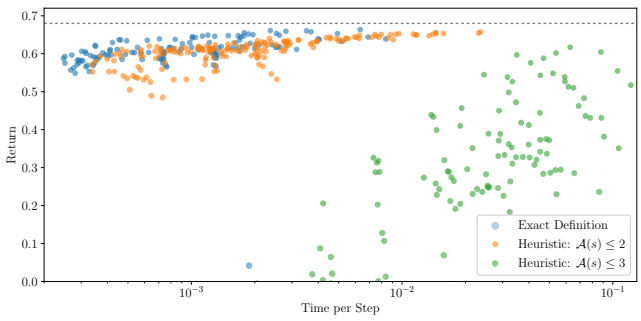


(c) Subgoal-MCTS in hierarchical control mode with tree reuse (S-MCTS-H-R)

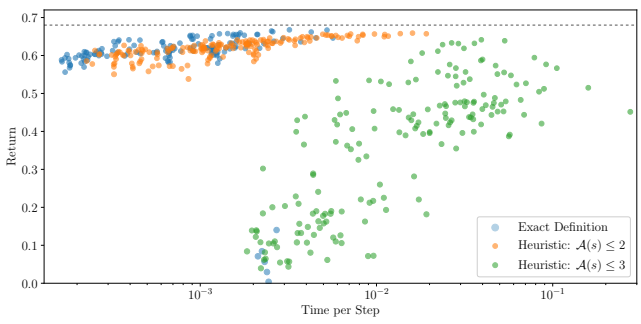
Figure 3: Average return per computation time per step (normalized by the number of trees) of S-MCTS, S-MCTS-H, and S-MCTS-H-R for different desired action coverages p_0 and tree counts in the Tetris domain. The x-axis uses a logarithmic scale.



(a) Subgoal-MCTS in polling control mode (S-MCTS)



(b) Subgoal-MCTS in hierarchical control mode (S-MCTS-H)



(c) Subgoal-MCTS in hierarchical control mode with tree reuse (S-MCTS-H-R)

Figure 4: Average return per computation time per step of S-MCTS, S-MCTS-H, and S-MCTS-H-R for different subgoal definitions and parameter settings in the Gridworld domain. The x-axis uses a logarithmic scale.

Exploration in S-MCTS depends on the number of subgoals. This was shown in the last experiment, where all variants of S-MCTS displayed worse performance, when using a very coarse subgoal heuristic.When using S-MCTS in hierarchical control mode, computational efficiency can be significantly increased. If a suboptimal choice has been made, S-MCTS-H is unable to recover from the performed actions until the next subgoal is reached, while S-MCTS in polling control mode can locally compensate for suboptimal choices at each time step. Enabling tree reuse slightly improves performance as shown in the first experiment (Fig. 2). Since tree reuse avoids complete replanning, S-MCTS can explore the search space more thoroughly to find better macro-actions. In addition, S-MCTS is shown to benefit from root parallelization. When generating multiple search trees in parallel to search for macro-actions, the performance of all S-MCTS variants can be further improved, while requiring less time. This encourages to exploit multiple cores in real-time applications to make high-quality decisions at certain time frames. When combining tree reuse with a high degree of parallelization, the search time can be drastically reduced by reusing the sets of discovered macro-actions from previous planning steps, while compensating for the approximation errors caused by each individual tree.

Overall, the question of how to adequately define subgoal predicates remains. Future work may further extend flexibility on subgoal predicates, for instance allowing to respect a history of states instead of just a single state.

References

[Auer *et al.*, 2002] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine learning*, 2002.

[Bai *et al.*, 2012] Aijun Bai, Feng Wu, and Xiaoping Chen. Online planning for large MDPs with MAXQ decomposition. In *AAMAS*. IFAAMAS, 2012.

[Bai *et al.*, 2016] Aijun Bai, Siddharth Srivastava, and Stuart J. Russell. Markovian State and Action Abstractions for MDPs via Hierarchical MCTS. In *IJCAI*. IJCAI/AAAI, 2016.

[Barto and Mahadevan, 2003] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1), Jan 2003.

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

[Botvinick *et al.*, 2009] Matthew M. Botvinick, Yael Niv, and Andrew C. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3), 2009.

[Browne *et al.*, 2012] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1), 2012.

[Chaslot *et al.*, 2008] Guillaume M. J. B. Chaslot, Mark H. M. Winands, and H. Jaap van den Herik. Parallel monte-carlo tree search. In *Computers and Games*. Springer, 2008.

[Dietterich, 2000] Thomas G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13, 2000.

[He *et al.*, 2010] Ruijie He, Emma Brunskill, and Nicholas Roy. PUMA: Planning Under Uncertainty with Macro-Actions. In *AAAI*. AAAI Press, 2010.

[Jaskowski *et al.*, 2015] Wojciech Jaskowski, Marcin Grzegorz Szubert, Pawel Liskowski, and Krzysztof Krawiec. High-Dimensional Function Approximation for Knowledge-Free Reinforcement Learning: a Case Study in SZ-Tetris. In *GECCO*. ACM, 2015.

[Kaelbling, 1993] Leslie Pack Kaelbling. Hierarchical Learning in Stochastic Domains: Preliminary Results. In *ICML*. Morgan Kaufmann, 1993.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo Planning. In *Eur. Conf. Machine Learning*. Springer, 2006.

[Mausam and Kolobov, 2012] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lect. on AI and Machine Learning. Morgan & Claypool Publishers, 2012.

[Parr and Russell, 1998] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *1997 Conf. on Advances in Neural Information Processing Systems*, NIPS '97. MIT Press, 1998.

[Puterman, 2014] Martin L Puterman. *Markov Decision Processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[Rubin and Watson, 2011] Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial Intelligence*, 175(5), 2011.

[Russell and Norvig, 2010] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[Scherrer *et al.*, 2015] Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist. Approximate modified policy iteration and its application to the game of Tetris. *Journal of Machine Learning Research*, 16, 2015.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550, October 2017.

[Solway *et al.*, 2014] Alec Solway, Carlos Diuk, Natalia Córdova, Debbie Yee, Andrew G. Barto, Yael Niv, and Matthew M. Botvinick. Optimal behavioral hierarchy. *PLOS Computational Biology*, 10(8), August 2014.

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

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.*, 112(1-2), 1999.

[Thiery and Scherrer, 2009] Christophe Thiery and Bruno Scherrer. Building Controllers for Tetris. *ICGA Journal*, 32(1), 2009.

[Vien and Toussaint, 2015] Ngo Anh Vien and Marc Toussaint. Hierarchical Monte-carlo Planning. In *Proc. of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15. AAAI Press, 2015.

[Weinstein and Littman, 2013] Ari Weinstein and Michael L Littman. Open-Loop Planning in Large-Scale Stochastic Domains. In *27th AAAI Conference on Artificial Intelligence*, 2013.

[Zhongjie *et al.*, 2011] Cai Zhongjie, Dapeng Zhang, and Bernhard Nebel. Playing Tetris Using Bandit-Based Monte-Carlo Planning. In *AISB 2011: AI and Games*, January 2011.