Lecture 2a: Generative Models - Monte Carlo Tree Search

Qi Zhang

Last Updated: September 2025

1 Online Planning with a Generative Model

Planning algorithms like value iteration (VI) and policy iteration (PI) solve Bellman (optimality) equations with the full knowledge of the MDP (e.g., the exact distribution over next states given any state-action pair). We here consider a more relaxed setting, where transition and reward functions (P, R) of the underlying (infinite-horizon) MDP $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ are unknown, but we can query them for any state-action pair to get samples, $s' \sim P(s, a), r \sim R(s, a)$, or $s', r \sim M(s, a)$ as a shorthand. In this setting, we often say we have a *simulator* or a *generative model*. In this note, we restrictively consider MDPs where every trajectory ends in a terminal state after a finite number of transitions so we can safely choose $\gamma = 1$, examples including board games like chess and many others.

Moreover, VI and PI compute an optimal policy purely offline for all possible states and then look up the policy to select actions to take. While ensuring optimality, such *offline planning* methods are not feasible for large state spaces. In this note, we are instead interested in the *online planning* paradigm where planning and action execution is interleaved: at the current state, a certain computation budget is allocated to try to figure out a good action to take; this process is repeated after taking that action and transiting to a next state. This is much closer to how humans play games like chess.

2 Monte Carlo Tree Search - The Algorithmic Template

Monte Carlo Tree Search (MCTS) is a algorithmic technique to figure out a good action to take in current state by progressively building up a search tree where a parent-child connection represents a transition and nodes in the tree accumulates statistics from multiple sampled episodes. More specifically, initializing the tree with the only node being the current state as the root, MCTS repeatedly performs the following procedures as depicted in Figure 1 and outlined in 1 and 2:

1. Selection: From the root, navigate down the tree with transitions sampled from the generative model and some action-selection rule until a new transition occurs from a leaf node.
2. Expansion: Create a new leaf by attaching that new transition to the tree.
3. Simulation: Starting from the new leaf's state, simulate a trajectory by the generative model and some action-selection rule until reaching a terminal state.
4. Backpropagation: Using the rewards from the root to the terminal state, modify the value estimates along the path.

Each node τ tracks N_τ, the total number of sampled paths that visited this node's state, and G_τ^{total} (i.e., τ.total_reward in Algorithms 1 and 2), the total reward accumulated from that node to the terminal states over all N paths. To select a good action from the root node, we hope the empirical mean of $G_{\tau,a}^{\text{total}}/N_{\tau,a}$ well approximates Q-value of a reasonably good policy, where $G_{\tau,a}^{\text{total}}$ is the total

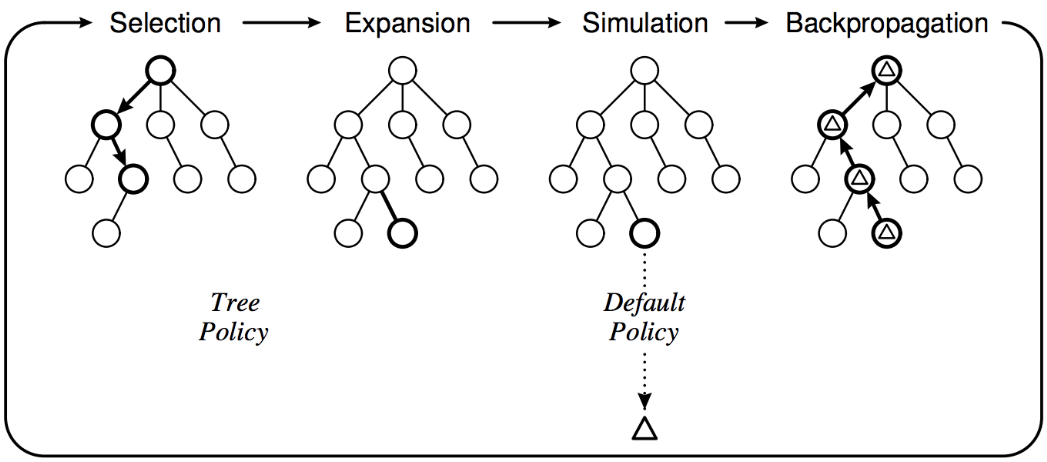

Figure 1: Illustration of an MCTS iteration (credit to [1]).

reward for all children of the node connected by a branch for action a (respectively for $N_{\tau,a}$). In the typical case where the state space is too huge to be fully explored, Selection (line 4 in Algorithm 2) should strike a balance between exploration of unknown paths and exploitation of historically promising paths. Expansion grows the tree, often by one leaf per iteration. We hope that each node on average gets a good number of sampled episodes to accumulate its statistics, so it is important not to grow the tree too fast. Selection and Expansion together are sometimes referred to as Tree Policy. Simulation aims to choose actions quickly, usually by some heuristics or just random action selection. Various algorithmic choices for these procedures give rise to specific instantiations. Sections **??** and 3 respectively discuss two well-known instantiations from this template, UCT and AlphaZero.

Implementation details

As above, the algorithmic description of MCTS is often based on the data structure of *state node*: a node is associated with a state s along paths in the tree that visit this node, with its children indexed by (a, s') for a transition of (s, a, s'). In hw2.ipynb of our Homework 2, we adopt an alterative implementation that explicitly distinguishes between the state node, referred to as the *V-node*, and the action node that is referred to as the *Q-node*. In a path, V-nodes are still associated with the states therein, while Q-node are associated with the actions. Therefore,

- The two types of nodes are interleaved depth-wise: a V-node has only Q-nodes (but not V-nodes) as its children and a Q-node has only V-nodes (but not Q-nodes) as its children;
- The root node is always a V-node of current state.
- The expansion happens upon an unseen transition (s, a, s') starting from a leaf V-node of state s and grows the tree with a new leaf V-node of state s' and a new Q-node of action a as its parent if a has not been tried, so the leafs are always V-nodes.
- Statistics such as $N_{\tau,a}$, $G^{\text{total}}_{\tau,a}$, and step reward are accumulated at the corresponding Q-nodes, and V-node can thus access its statistics by aggregating its Q-node children. For example, $N_\tau = \sum_a N_{\tau,a}$ where N_τ is the number of visits for a V-node τ, which equals to the summation over its Q-node children.

Algorithm 1 Monte Carlo Tree Search

```
procedure MonteCarloTreeSearch(τ₀)                      ▷ Root node τ₀
    repeat
        τ ← TreePolicy(τ₀)          ▷ New leaf τ, total reward G_i cumulated from root τ₀
        s ← τ.state
        G ← DefaultPolicy(s)                  ▷ G is total reward cumulated from state s
        Backup(τ, G)                            ▷ Update node statistics along the path
    until Timeout()
    return BestAction(τ₀)           ▷ Pick the approximately optimal root-level action
end procedure

procedure InitNode(s)
    τ.parent ← null
    τ.state ← s
    τ.count ← 0
    τ.total_reward ← 0                          ▷ Cumulated over timesteps and episodes
    τ.children[a][s′] ← null, for all a, s′
    return τ
end procedure
```

Algorithm 2 Monte Carlo Tree Search - Procedures

```
procedure TreePolicy($\tau_0$)
    $\tau, s \leftarrow \tau_0, \tau_0$.state
    while Nonterminal($s$) do
        $a \leftarrow$ SelectAction($\tau$)                ▷ Heuristically select an action
        $s', r \sim M(s, a)$                ▷ Sample next state and reward from generative model
        if $\tau$.children[$a$][$s'$] = null then                ▷ Is this the first observation of $s \xrightarrow{a} s'$?
            $\tau' \leftarrow$InitNode($s'$)                ▷ Initialize leaf node for state $s'$
            $\tau'$.parent $\leftarrow \tau$.children[$a$][$s'$]                ▷ Attach leaf node to the tree
            $\tau$.children[$a$].reward $\leftarrow r$                ▷ Record $r = R(s, a)$
            return $\tau$.children[$a$][$s'$]                ▷ Move on to simulation phase
        end if
        $\tau$.children[$a$].reward $\leftarrow r$                ▷ Record $r = R(s, a)$
        $\tau \leftarrow \tau$.children[$a$][$s'$]
        $s \leftarrow s'$
    end while
    return $\tau$
end procedure

procedure DefaultPolicy($s$)                ▷ Decision policy for simulation
    $G = 0$
    while Nonterminal($s$) do
        $a \sim$ DefaultActionSelection($s$)                ▷ e.g., randomly select an action from $\mathcal{A}$
        $s', r \sim M(s, a)$                ▷ Sample next state and reward from generative model
        $G \leftarrow G + r$
    end while
    return $G$
end procedure

procedure Backup($\tau, G$)                ▷ $G$ is the total reward cumulated from node $\tau$
    repeat
        $\tau$.total_reward $\leftarrow \tau$.total_reward $+ G$
        $\tau$.count $\leftarrow \tau$.count $+ 1$
        $\tau, a \leftarrow \tau$.parent                ▷ $a$ is the action connecting the two nodes
        if $\tau$ is not null then
            $G \leftarrow G + \tau$.children[$a$].reward                ▷ Add step reward along the path
        end if
    until $\tau$ is null
    return
end procedure

procedure BestAction($\tau$)                ▷ Pick the empirically best action at node $\tau$
    return $\arg\max_{a \in \mathcal{A}} \frac{\sum_{\tau' \in \tau.\text{children}[a][\cdot]} \tau'.\text{total\_reward}}{\sum_{\tau' \in \tau.\text{children}[a][\cdot]} \tau'.\text{count}}$
end procedure
```

3 UCT

UCT [2], Upper Confidence Bounds (UCB) applied to Trees, applies the UCB algorithm [3] to the Selection procedure of MCTS to balance the exploration-exploitation tradeoff. Specifically, UCT's Selection selects the action at each (V-)node τ (line 4 in Algorithm 2) that maximizes the upper confidence bound on the estimated total reward from that node:

$$\arg\max_{a \in \mathcal{A}} \quad \text{UCB}(\tau, a) := \frac{G^{\text{total}}_{\tau,a}}{N_{\tau,a}} + c\sqrt{\frac{\ln(N_s)}{N_{\tau,a}}}$$

where $c > 0$ is a hyperparameter controlling how much exploration is favored over exploitation.

References

[1] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[2] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.

[3] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, May 2002.