Monte Carlo Tree Search & Temporal-Difference

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Fall 2019

avereshc@buffalo.edu

September 12, 2019

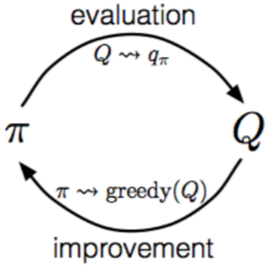
*Slides are based on Monte Carlo Tree Search, MIT 16.412J / 6.834J Cognitive Robotics
Deep Reinforcement Learning and Control, CMU 10703, Carnegie-Mellon University

Overview

Table of Contents

Recap: Monte-Carlo Control

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$



- MC policy iteration step: Policy evaluation using MC methods followed by policy improvement
- Policy improvement step: greedify with respect to value (or action-value) function

Recap: Monte-Carlo Algorithm

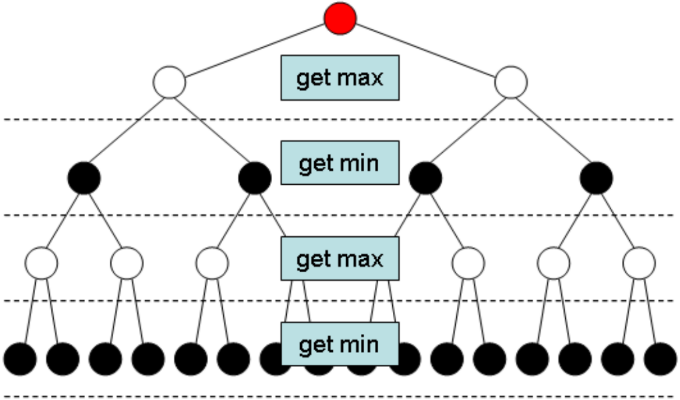
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
- $Q(s,a) \leftarrow$ arbitrary
- $\pi(s) \leftarrow$ arbitrary
- $Returns(s,a) \leftarrow$ empty list

Fixed point is optimal policy π^*

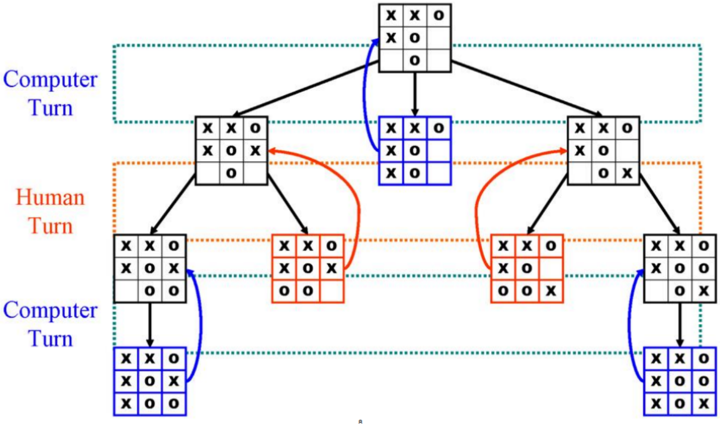
Repeat forever:
- Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0
- Generate an episode starting from S_0, A_0, following π
- For each pair s, a appearing in the episode:
  - $G \leftarrow$ return following the first occurrence of s, a
  - Append G to $Returns(s,a)$
  - $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
- For each s in the episode:
  - $\pi(s) \leftarrow \arg\max_a Q(s,a)$

Table of Contents

Minimize the maximum possible loss



Minimax



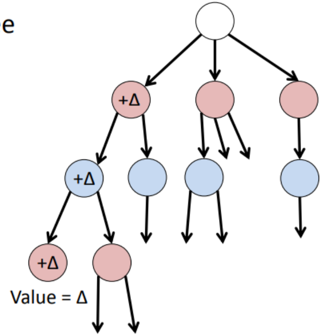
MCTS Outline

1. Descend through the tree
2. Create new node
3. Simulate
4. Update the tree

Repeat!

5. When you're out of time,

Return "best" child.

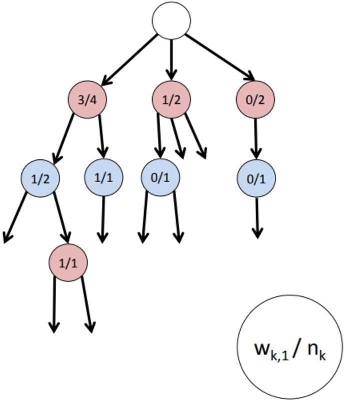


What do we store?

For game state k:

n_k = # games played involving k

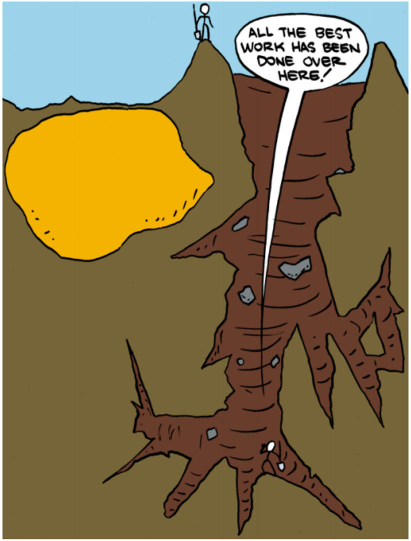
$w_{k,p}$ = # games won (by player p) that involved k



1. Descending

We want to **expand**,

but also to **explore**.



17

MCTS: Outline

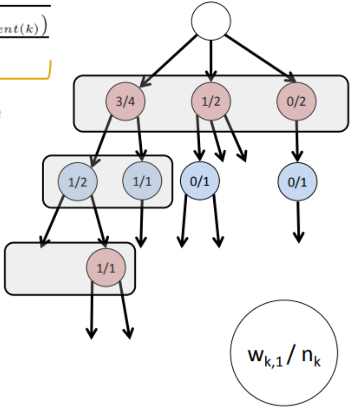
Solution: *Upper Confidence Bound*

$$UCB1(k,p) = \underbrace{E[win|k,p]}_{\text{expand}} + \underbrace{C\sqrt{\frac{2\ln(n_{parent(k)})}{n_k}}}_{\text{explore}}$$

$$\approx \frac{w_{k,p}}{n_k} + C\sqrt{\frac{2\ln(n_{parent(k)})}{n_k}}$$

At each step,

maximize UCB1(k, p)

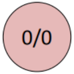


2. Expanding

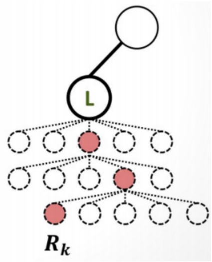
Not very complicated.

Make a new node!

Set $n_k = 0$, $w_k = 0$



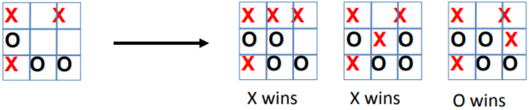
3. Simulating

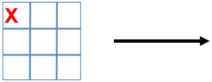


Simulating a real game is hard.

Let's just play the game out randomly!

If we win, $\Delta = +1$. If we lose or tie, $\Delta = 0$.





A lot of options...

4. Updating the Tree

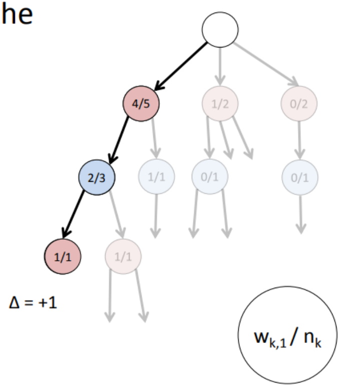
Propagate recursively up the parents.

Given simulation result Δ,

for each k:

$$n_{k\text{-new}} = n_{k\text{-old}} + 1$$

$$w_{k,1\text{-new}} = w_{k,1\text{-old}} + \Delta$$

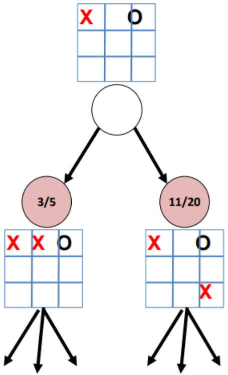


5. Terminating

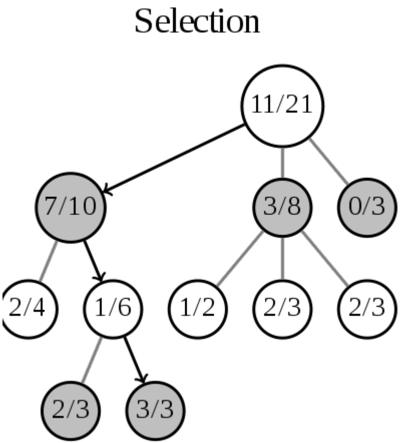
Return the best-ranked first ancestor!

What determines "best"?

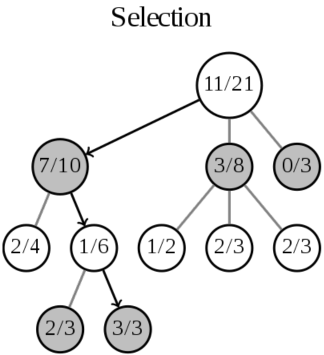
- Highest E[win|k]
- Highest E[win|k] AND most visited

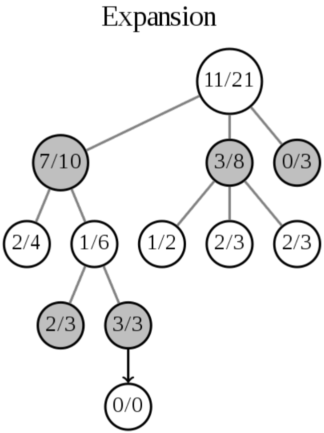


MCTS: Selection

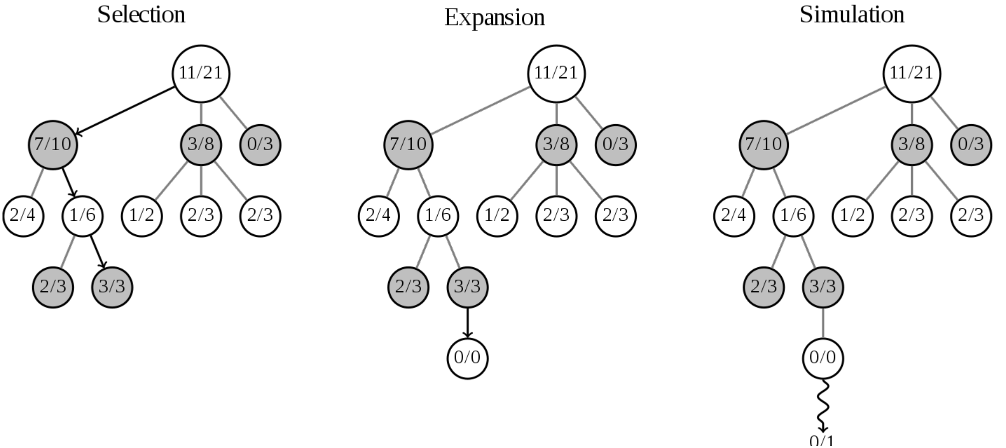


MCTS: Expansion

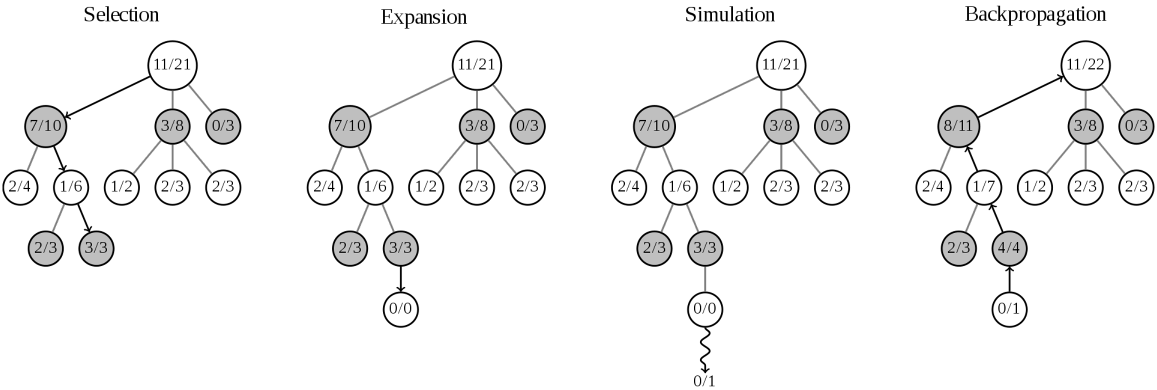




MCTS: Simulation



MCTS: Back-propagation



MCTS

Advantages:

1. Grows tree asymmetrically, balancing expansion and exploration
2. Depends only on the rules
3. Easy to adapt to new games
4. Heuristics not required, but can also be integrated
5. Complete: guaranteed to find a solution given time

Disadvantages:

1.

Table of Contents

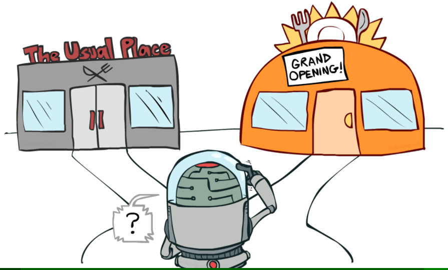
Exploration vs Exploitation

Online decision-making involves a fundamental choice:

- **Exploitation:** Make the best decision given current information (greedy)
- **Exploration:** Gather more information

The greedy algorithm selects action with highest value:

$$a_t^* = \arg\max_a Q_t(s, a)$$



Exploration vs Exploitation

$\epsilon -$ *greedy* algorithm:

- With probability ϵ choose a random action a
- With probability $1 - \epsilon$ choose "greedy" action a with the highest Q-value.

Exploration vs Exploitation

In ϵ-greedy action selection, for the case of two actions $[a_1, a_2]$ and $\epsilon = 0.5$, what is the probability that the greedy action is selected?

Table of Contents

Monte Carlo (MC) and Temporal Difference (TD) Learning

- Goal: learn $v_\pi(s)$ from episodes of experience under policy π
- Incremental every-visit Monte-Carlo:
  - Update value $V(S_t)$ toward actual return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$

- Simplest Temporal-Difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward estimated returns $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

- $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error.

DP vs. MC vs TD Learning

- Remember:

MC: sample average return approximates expectation

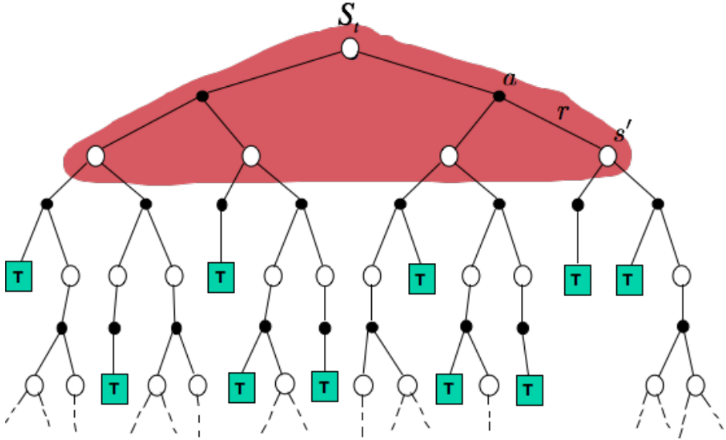
$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] .
\end{aligned}
$$

TD: combine both: Sample expected values and use a current estimate $V(S_{t+1})$ of the true $v_\pi(S_{t+1})$

DP: the expected values are provided by a model. But we use a current estimate $V(S_{t+1})$ of the true $v_\pi(S_{t+1})$

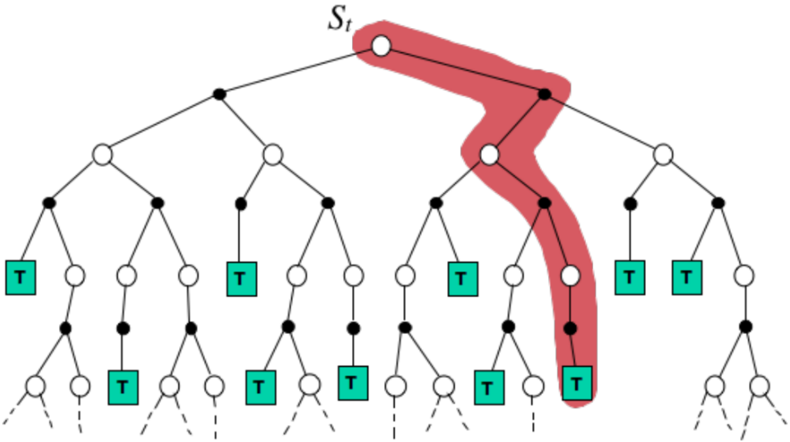
Dynamic Programming

$$V(S_t) \leftarrow E_\pi\left[R_{t+1} + \gamma V(S_{t+1})\right] = \sum_a \pi(a|S_t) \sum_{s',r} p(s',r|S_t,a)[r + \gamma V(s')]$$



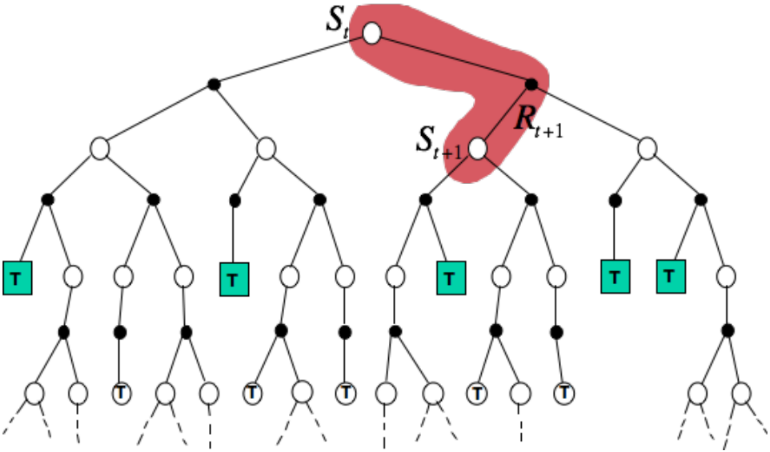
Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$



TD(0) Method

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$



TD Methods Bootstrap and Sample

- **Bootstrapping:** update involves an estimate
  - MC does not bootstrap
  - DP bootstrap
  - TD bootstrap
- **Sampling:** update does not involve an expected value
  - MC samples
  - DP does not sample
  - TD samples