Reinforcement Learning

CSCI 4511/6511

Joe Goldfrank

Announcements

- Extra Credit HW: Due 4 Dec
- **Project Proposals**
- Final Exam: 4 Dec
- Project Deadline: 13 Dec

Monte Carlo Tree Search

Multi-Armed Bandits

- Slot machine with more than one arm
- Each pull has a cost
- Each pull has a payout
- Probability of payouts unknown
- Goal: maximize reward
    - Time horizon?

Solving Multi-Armed Bandits

Confidence Bounds

- Expected value of reward per arm
  - Confidence interval of reward per arm
- Select arm based on upper confidence bound

- How do we estimate rewards?
  - Explore vs. exploit

Bandit as MDP?

Bandit Strategies

- Gittins Index: $\lambda = \max_{T>0} \frac{E[\sum^{T-1} \gamma^t R_t]}{E[\sum^{T-1} \gamma^t]}$
- Upper Confidence Bound for arm M_i:
  - $UCB(M_i) = \mu_i + \frac{g(N)}{\sqrt{N_i}}$
  - $g(N)$ is the "regret"
- Thompson Sampling
  - *Sample* arm based on probability of being optimal

Tree Search

- Forget DFS, BFS, Dijkstra, A*
  - State space too large
  - Stochastic expansion
- Impossible to search entire tree
- Can *simulate* problem forward in time from starting state

Monte Carlo Tree Search

- Randomly simulate trajectories through tree
  - Complete trajectory
  - No heuristic needed[1]
  - *Need* a model
- Better than exhaustive search?

1. Heuristics can be used.

Selection Policy

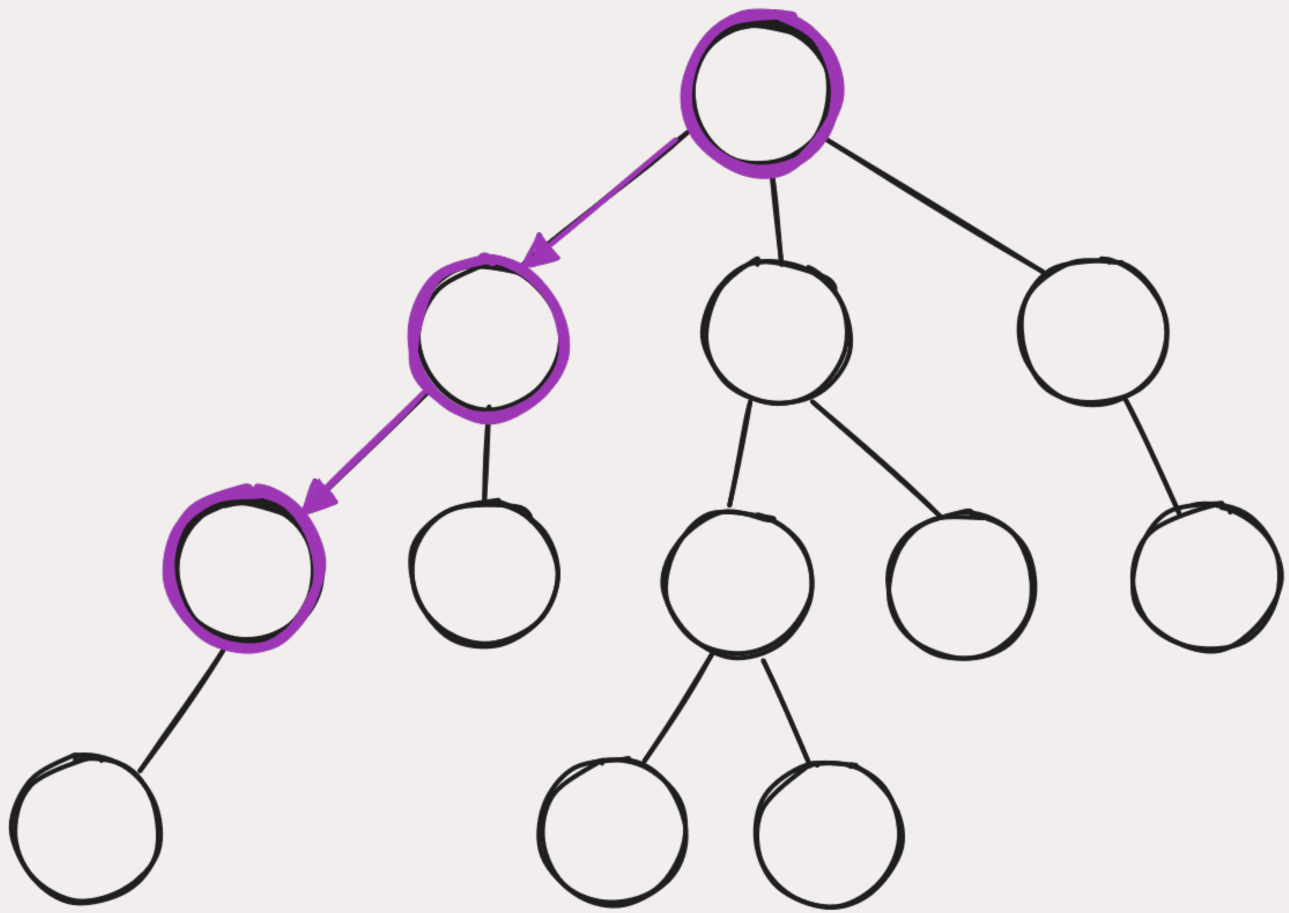
- Focus search on “important” parts of tree
  - Similar to alpha-beta pruning
- Explore vs. exploit
  - Simulation
  - Not actually exploiting the problem
  - Exploiting the *search*

Monte Carlo Tree Search

- Choose a node
    - Explore/exploit
    - Choose a successor
    - Continue to leaf of search tree
- Expand leaf node
- Simulate result until completion
- Back-propagate results to tree

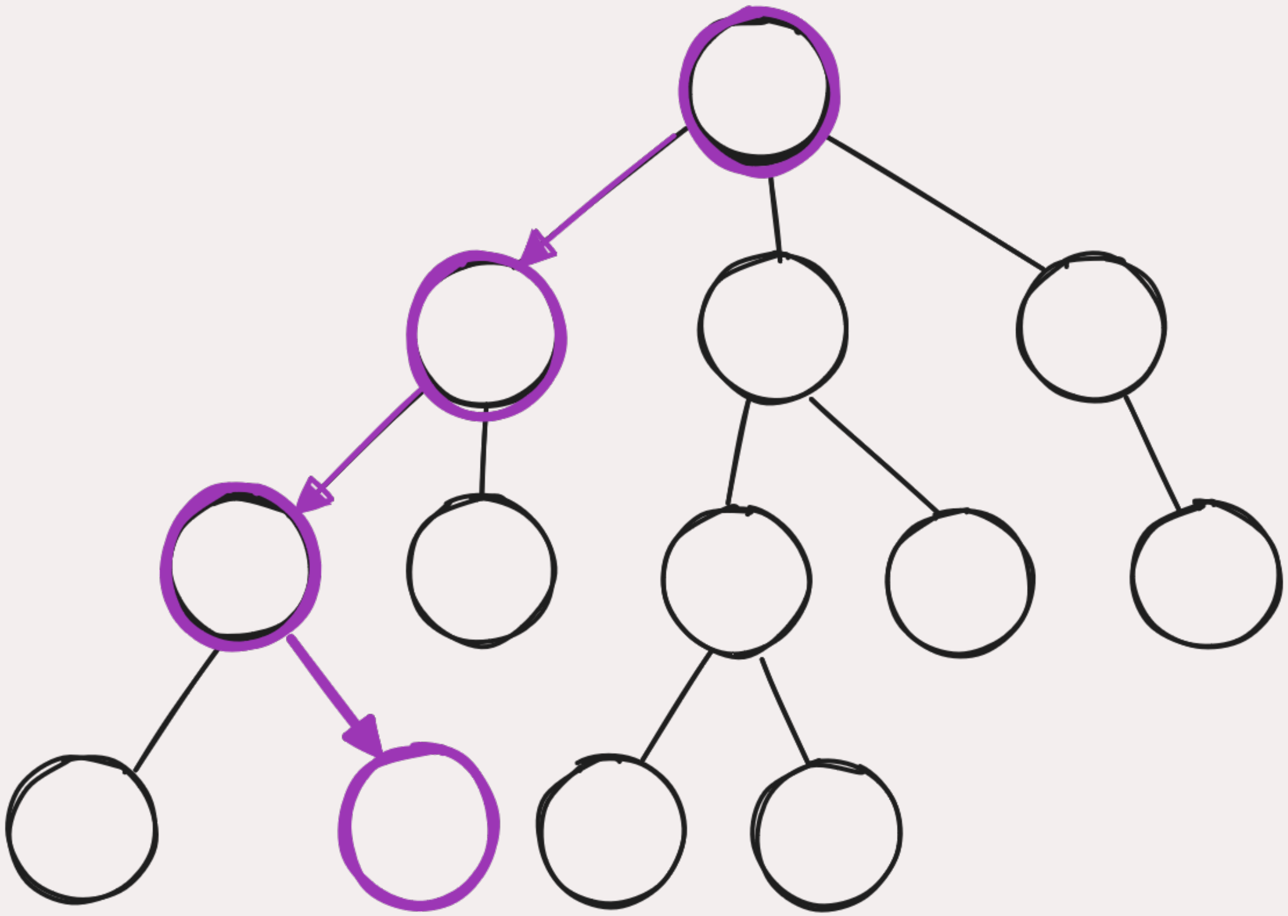
Monte Carlo Tree Search

Selection/Search



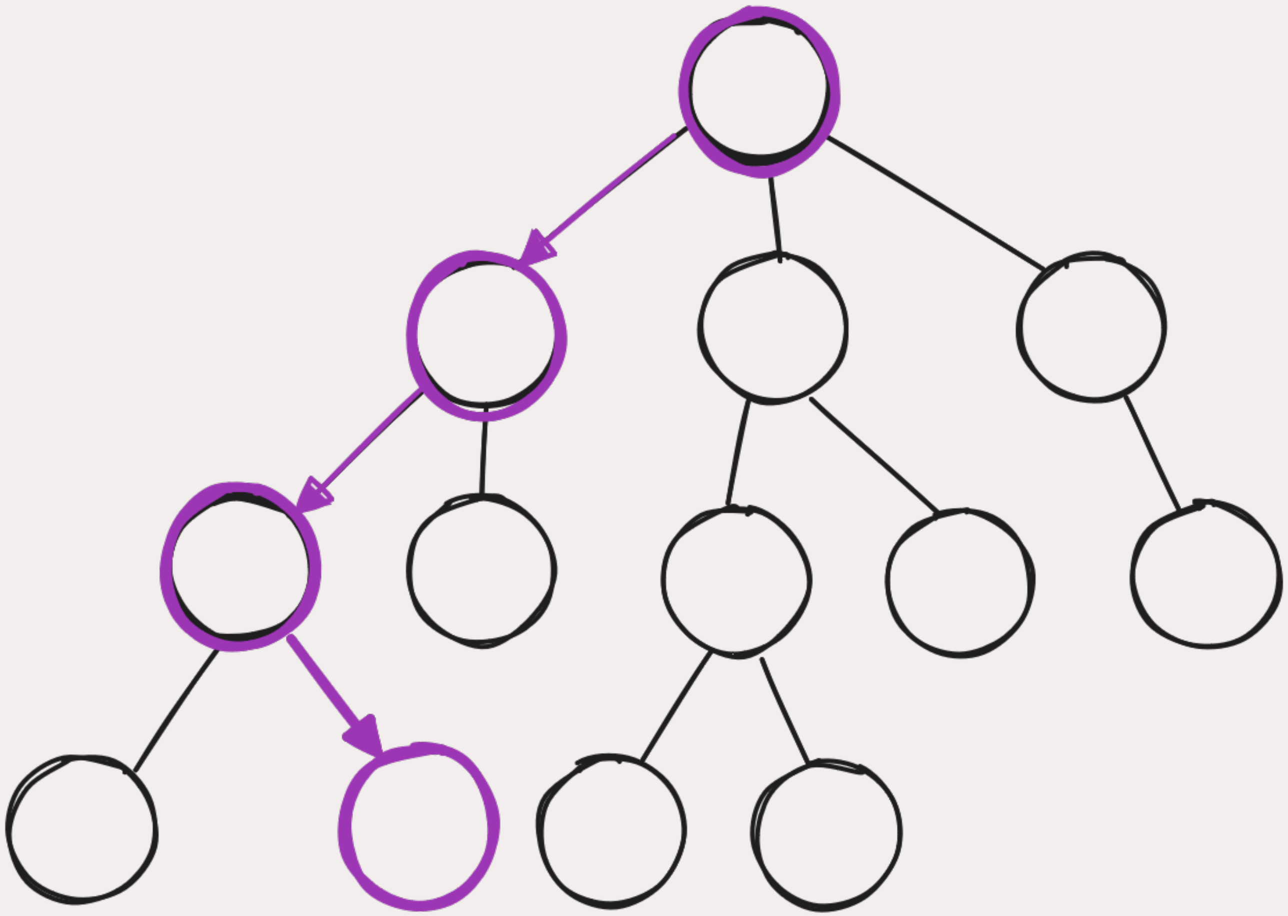
Monte Carlo Tree Search

Expansion



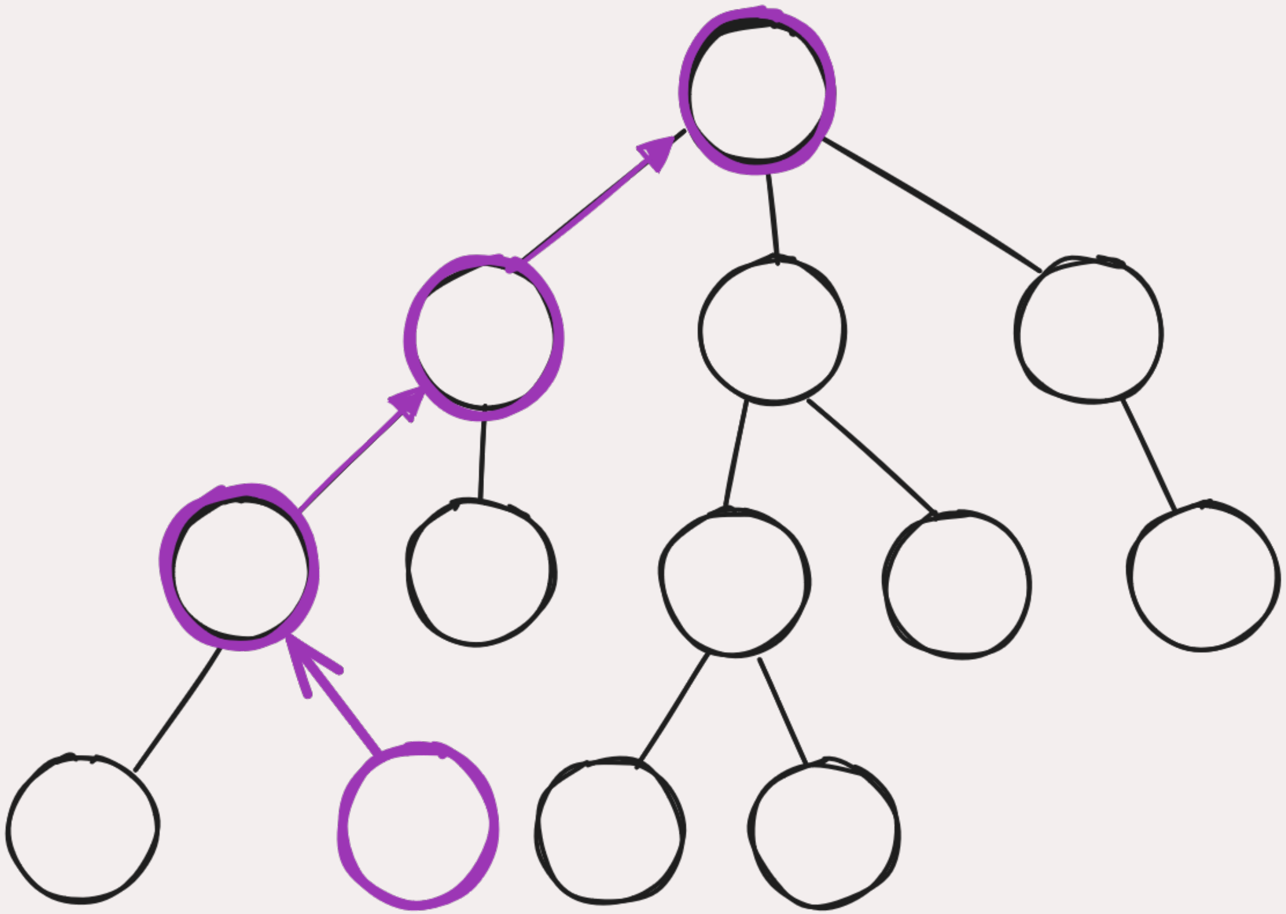
Monte Carlo Tree Search

Simulation/Rollout



Monte Carlo Tree Search

Back-Propagation



Upper Confidence Bounds for Trees (UCT)

- MDP: Maximize $Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$
  - Q for state s and action a
- POMDP: Maximize $Q(h, a) + c\sqrt{\frac{\log N(h)}{N(h,a)}}$
  - Q for history h and action a
  - History: action/observation sequence
- c is exploration bonus

UCT Search - Algorithm

Algorithm 4.9 Monte Carlo tree search

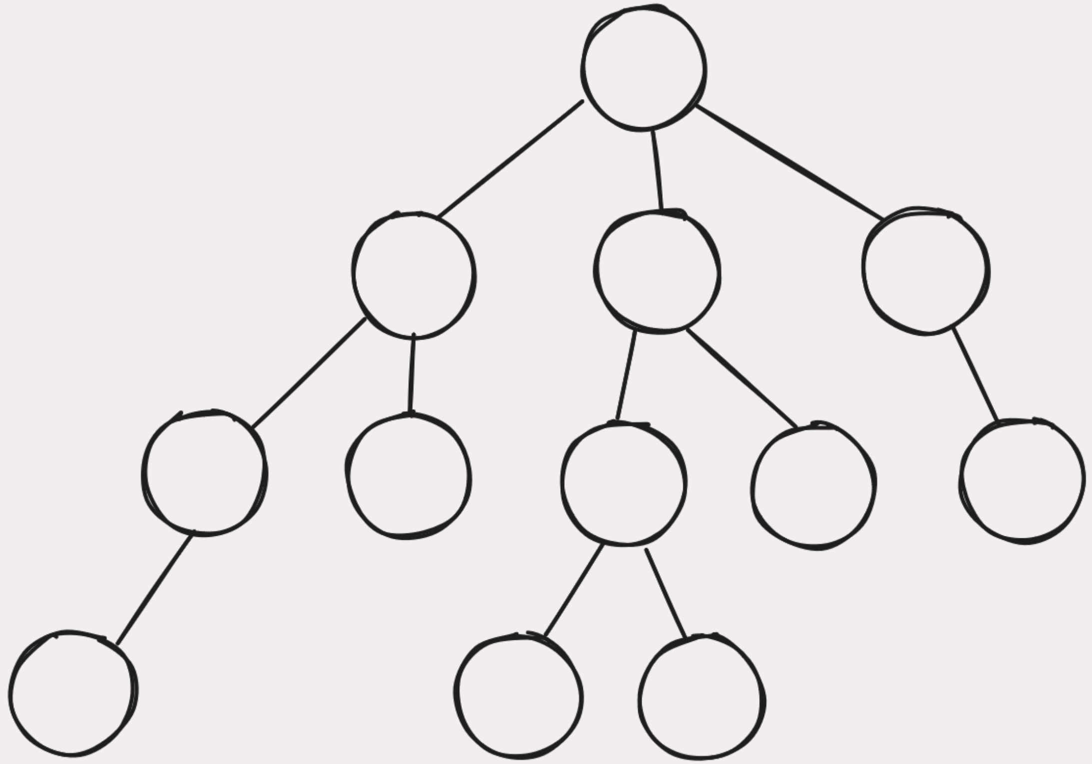
function SELECTACTION(s, d)
  loop
    SIMULATE(s, d, π_0)
  return $\arg\max_a Q(s, a)$
function SIMULATE(s, d, π_0)
  if $d = 0$
    return 0
  if $s \notin T$
    for $a \in A(s)$
      $(N(s,a), Q(s,a)) \leftarrow (N_0(s,a), Q_0(s,a))$
    $T = T \cup \{s\}$
    return ROLLOUT(s, d, π_0)
  $a \leftarrow \arg\max_{a \in A(s)} \left[Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \right]$
  $(s', r) \sim G(s, a)$
  $q \leftarrow r + \gamma \text{SIMULATE}(s', d-1, \pi_0)$
  $N(s,a) \leftarrow N(s,a) + 1$
  $Q(s,a) \leftarrow Q(s,a) + \frac{q - Q(s,a)}{N(s,a)}$
  return q

Algorithm 4.10 Rollout evaluation

function ROLLOUT(s, d, π_0)
  if $d = 0$
    return 0
  $a \sim \pi_0(s)$
  $(s', r) \sim G(s, a)$
  return $r + \gamma \text{ROLLOUT}(s', d-1, \pi_0)$

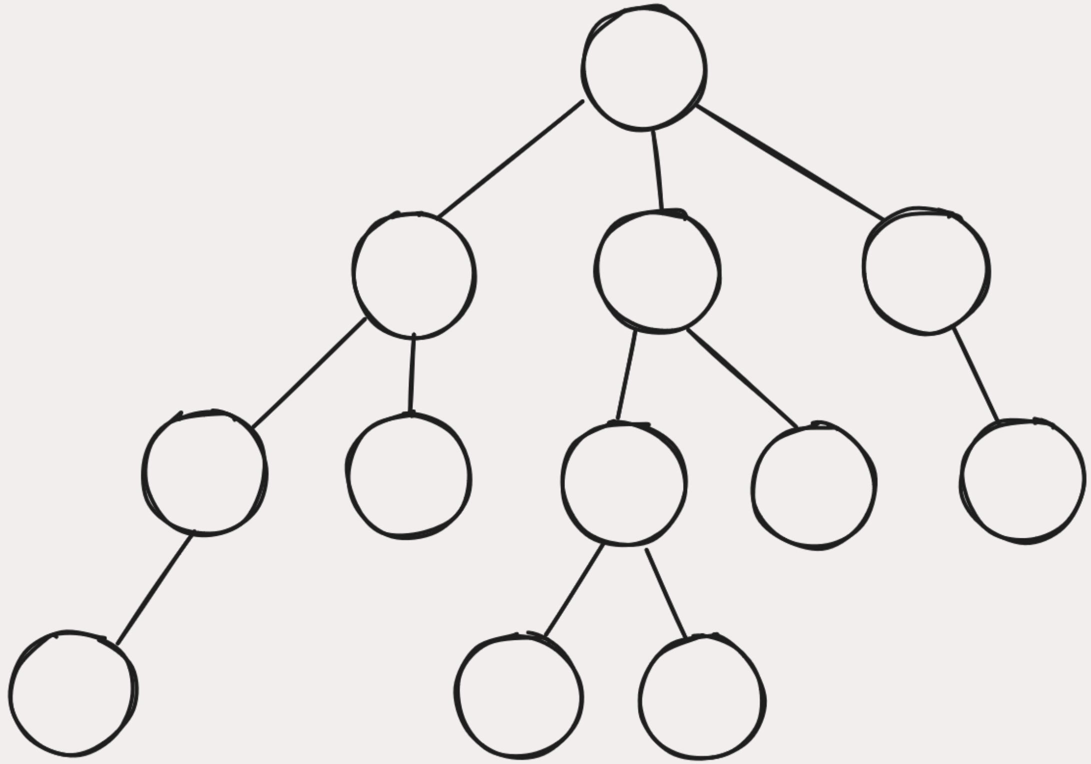
Mykal Kochenderfer. *Decision Making Under Uncertainty*, MIT Press 2015

Monte Carlo Tree Search - Search



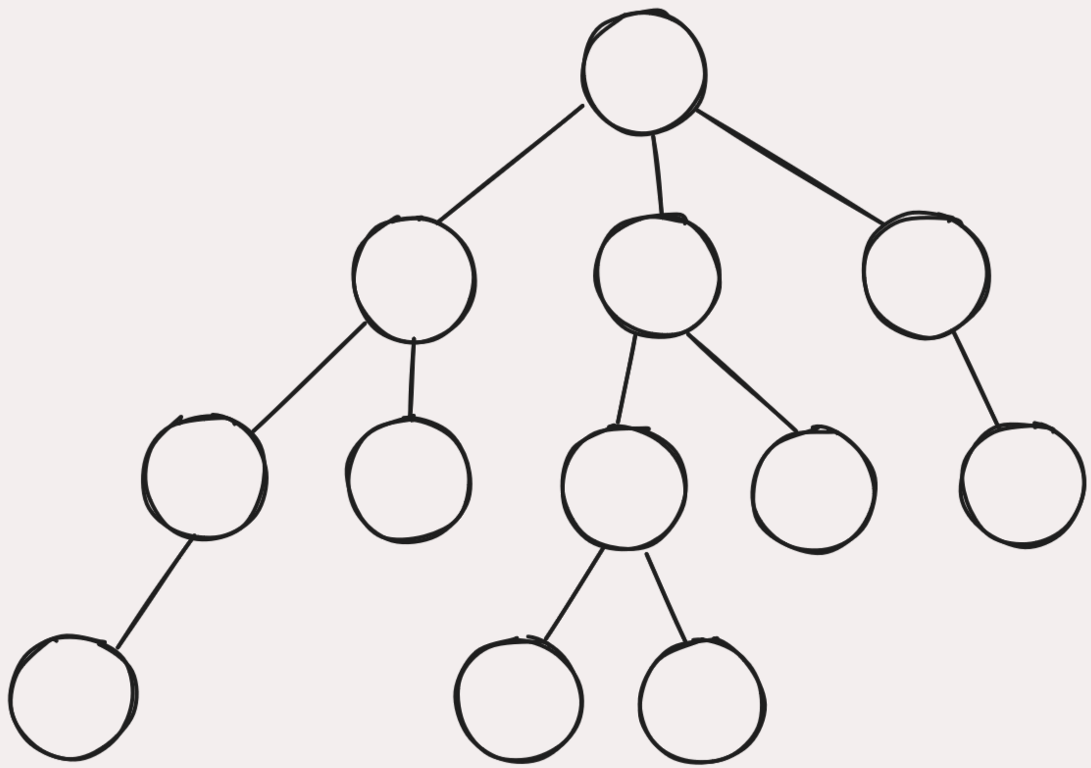
- If current state $\in T$ (tree states):
  - Maximize:
    $Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$
  - Update $Q(s,a)$ during search

Monte Carlo Tree Search - Expansion



- State $\notin T$
  - Initialize $N(s, a)$ and $Q(s, a)$
  - Add state to T

Monte Carlo Tree Search - Rollout



- Policy π_0 is "rollout" policy
  - Usually stochastic
  - States *not* tracked

Model Uncertainty

Erstwhile

- States
- Actions
- *Transition model* between states, based on actions
- *Known* rewards

Model Uncertainty

- No model of transition dynamics
- No initial knowledge of rewards

😣

We can *learn* these things!

Model Uncertainty

Action-value function:

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s' | s, a) U(s')$$

we don't know T:

$$U^\pi(s) = E_\pi \left[r_t + \gamma r_{t+1} + \gamma r_{t+2} + \gamma r_{t+3} + \ldots | s \right]$$

$$Q(s, a) = E_\pi \left[r_t + \gamma r_{t+1} + \gamma r_{t+2} + \gamma r_{t+3} + \ldots | s, a \right]$$

Temporal Difference (TD) Learning

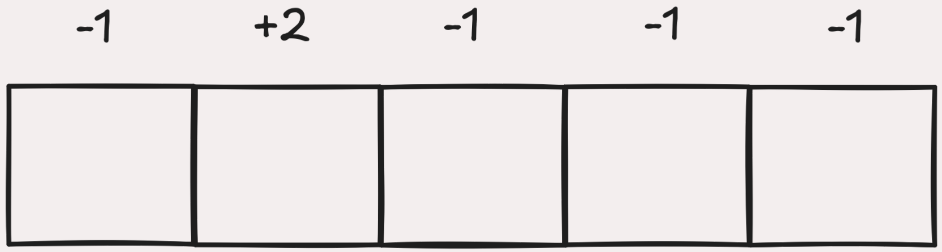
- Take action from state, observe new state, reward

$$U(s) \leftarrow U(s) + \alpha\left[r + \gamma U(s') - U(s)\right]$$

- Update immediately given (s, a, r, s')

- TD Error: $\left[r + \gamma U(s') - U(s)\right]$
  - Measurement: $r + \gamma U(s')$
  - Old Estimate: $U(s)$

TD Learning - Example



Q-Learning

- U^π gives us utility
- Solving for U^π allows us to pick a new policy

State-action value function: $Q(s, a)$

- $\max_a Q(s, a)$ provides optimal policy
- Goal: *Learn* $Q(s, a)$

Q-Learning

Iteratively update Q:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s', a') - Q(s,a) \right]$$

- Current state s and action a
- Next state s', next action(s) a'
- Reward R
- Discount rate γ
- Learning rate α

Q-Learning Algorithm

Algorithm 5.3 Q-learning

---

function QLEARNING

    $t \leftarrow 0$

    $s_0 \leftarrow$ initial state

    Initialize Q

    loop

        Choose action a_t based on Q and some exploration strategy

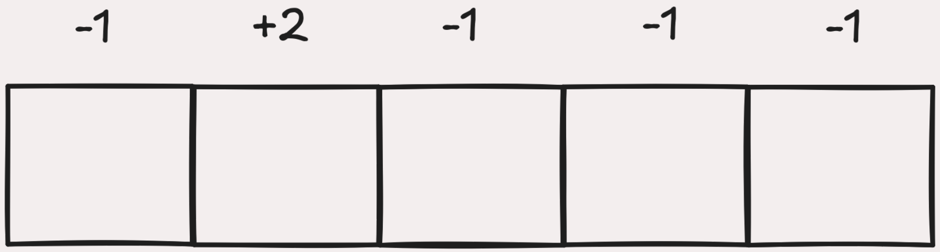
        Observe new state s_{t+1} and reward r_t

        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$

        $t \leftarrow t + 1$

Mykal Kochenderfer. *Decision Making Under Uncertainty*, MIT Press 2015

Q-Learning Example



Sarsa

Q-Learning:

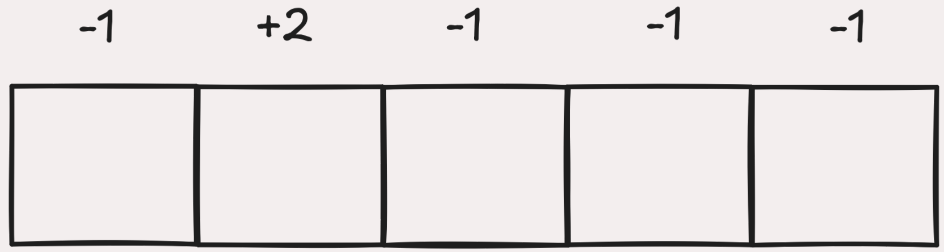
$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Sarsa:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma Q(s',a') - Q(s,a)\right]$$

Differences?

Sarsa Example



Q-Learning vs. Sarsa

- Sarsa is “on-policy”
  - Evaluates state-action pairs *taken*
  - Updates policy every step
- Q-learning is “off-policy”
  - Evaluates “optimal” actions for future states
  - Updates policy every step

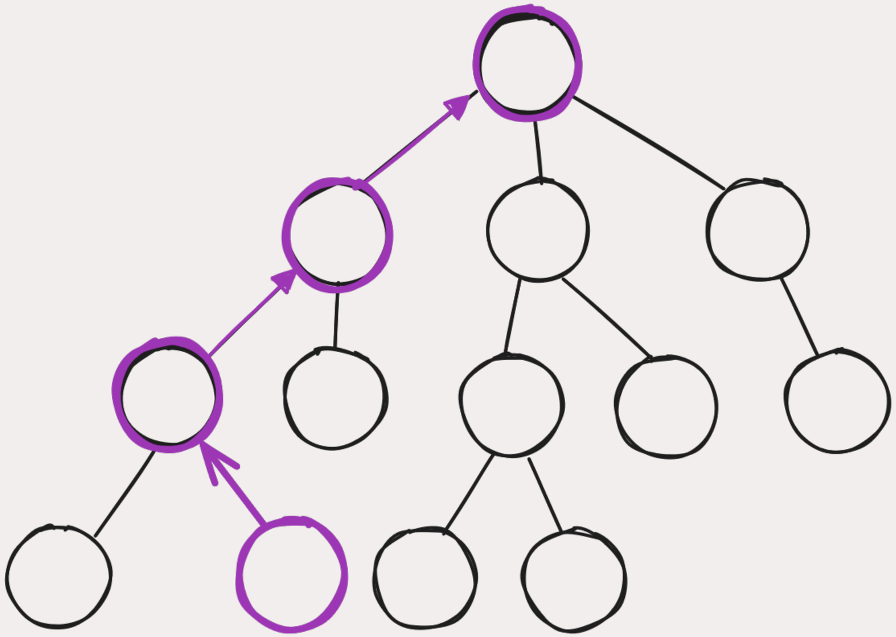
Exploration vs. Exploitation

- Consider only the goal of learning the optimal policy
  - Always picking "optimal" policy does not search
  - Picking randomly does not check "best" actions
- ϵ-greedy:
  - With probability ϵ, choose random action
  - With probability $1 - \epsilon$, choose 'best' action
  - ϵ need not be fixed

Eligibility Traces

- Q-learning and Sarsa both propagate Q-values slowly
  - Only updates individual state
- Recall MCTS:
  - (Also recall that MCTS needs a generative model)

Recall MCTS



Algorithm 4.9 Monte Carlo tree search

function $\textsc{SelectAction}(s, d)$
    loop
        $\textsc{Simulate}(s, d, \pi_0)$
    return $\arg\max_a Q(s, a)$
function $\textsc{Simulate}(s, d, \pi_0)$
    if $d = 0$
        return 0
    if $s \notin T$
        for $a \in A(s)$
            $(N(s, a), Q(s, a)) \leftarrow (N_0(s, a), Q_0(s, a))$
        $T = T \cup \{s\}$
        return $\textsc{Rollout}(s, d, \pi_0)$
    $a \leftarrow \arg\max_{a \in A(s)} \left[Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \right]$
    $(s', r) \sim G(s, a)$
    $q \leftarrow r + \gamma \textsc{Simulate}(s', d - 1, \pi_0)$
    $N(s, a) \leftarrow N(s, a) + 1$
    $Q(s, a) \leftarrow Q(s, a) + \frac{q - Q(s,a)}{N(s,a)}$
    return q

Eligibility Traces

- Keep track of what state-action pairs agent has seen
- Include future rewards in past Q-values
- *Very* useful for sparse rewards
  - Can be more efficient for non-sparse rewards

Eligibility Traces

- Keep $N(s, a)$: "number of times visited"
- Take action a_t from state s_t:
    - $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$
- Every time step:[1]
    - $\delta = R + \gamma Q(s', a') - Q(s, a)$
    - $Q(s, a) \leftarrow \alpha \delta N(s, a)$
    - $N(s, a) \leftarrow \gamma \lambda N(s, a)$
        - Discount factor γ
        - Time decay λ

1. Sarsa algorithm

Sarsa-λ

Sarsa:

- $Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma Q(s',a') - Q(s,a)\right]$

Sarsa-λ:

- $\delta = R + \gamma Q(s',a') - Q(s,a)$
- $Q(s,a) \leftarrow \alpha\delta N(s,a)$

Sarsa-λ

Algorithm 5.4 Sarsa(λ)-learning

function SarsaLambdaLearning(λ)
    Initialize Q and N
    $t \leftarrow 0$
    $s_0, a_0 \leftarrow$ initial state and action
    loop
        Observe reward r_t and new state s_{t+1}
        Choose action a_{t+1} based on some exploration strategy
        $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$
        $\delta \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$
        for $s \in S$
            for $a \in A$
                $Q(s, a) \leftarrow Q(s, a) + \alpha \delta N(s, a)$
                $N(s, a) \leftarrow \gamma \lambda N(s, a)$
        $t \leftarrow t + 1$

Sarsa-λ Example

Q-λ ?

Q-Learning:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Sarsa:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma Q(s',a') - Q(s,a) \right]$$

Sarsa-λ:

$$\delta = R + \gamma Q(s',a') - Q(s,a)$$
$$Q(s,a) \leftarrow \alpha \delta N(s,a)$$

Watkins Q-λ

Idea: only keep states in N(s,a) that policy would have visited

- Some actions are greedy: $\max_{a'} Q(s, a')$
- Some are random
- On random action, reset $N(s, a)$
- Why the difference from Sarsa?

Approximation Methods

- Large problems:
  - Continuous state spaces
  - Very large discrete state spaces
  - Learning algorithms can't visit all states
- Assumption: "close" states $\rightarrow$ similar state-action values

Local Approximation

- Store $Q(s, a)$ for a limited number of states: $\theta(s, a)$
- Weighting function β
  - Maps true states to states in θ

$$Q(s, a) = \theta^T \beta(s, a)$$

Update step:

$$\theta \leftarrow \theta + \alpha \left[R + \gamma \theta^T \beta(s', a') - \theta^T \beta(s, a) \right] \beta(s, a)$$

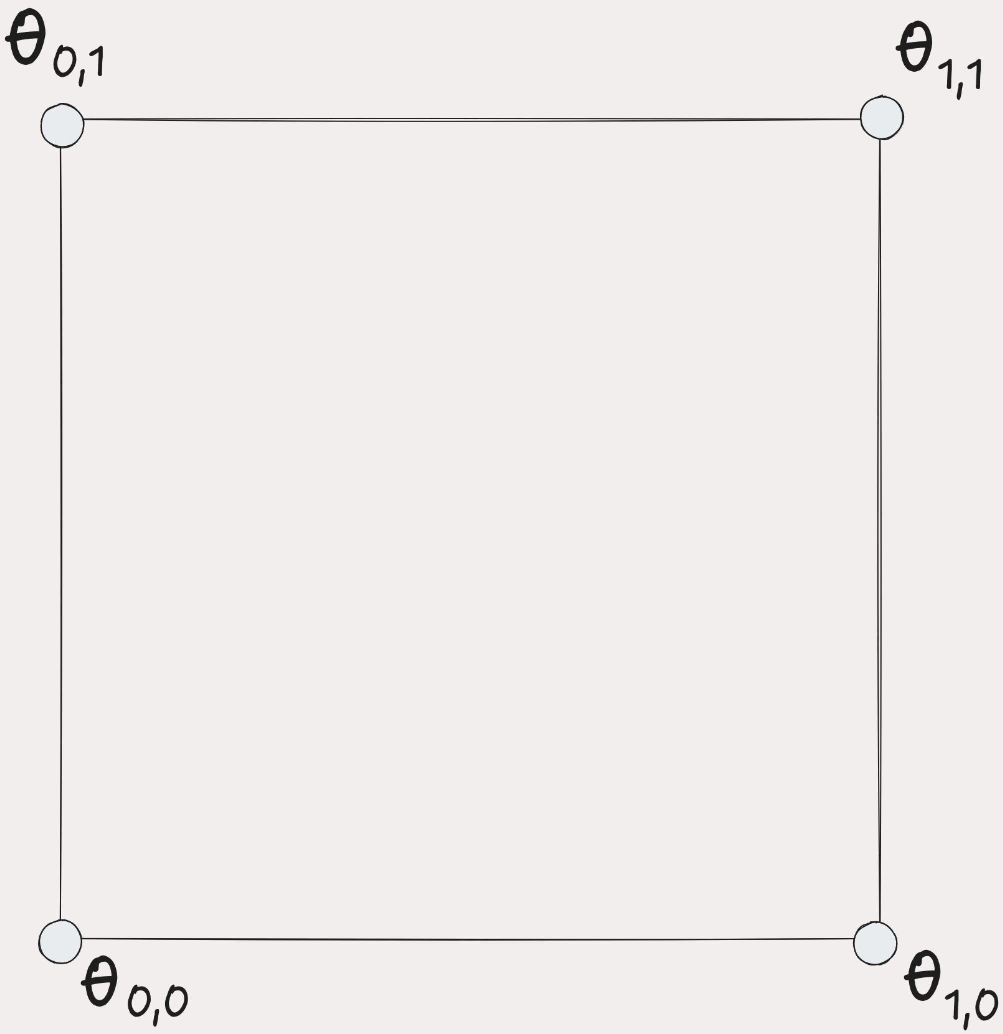
Linear Approximation Q-Learning

Algorithm 5.5 Linear approximation Q-learning

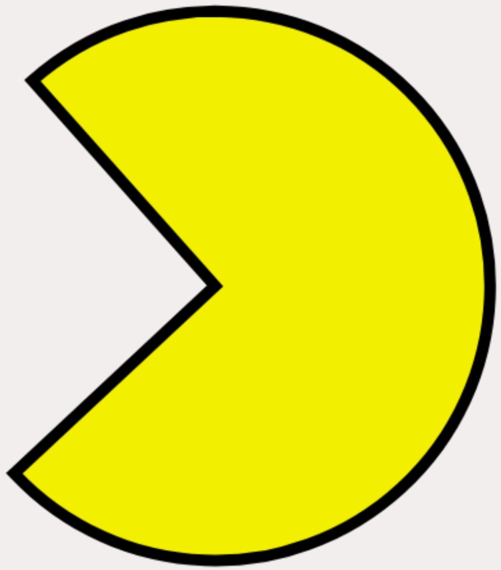
function LinearApproximationQLearning
  $t \leftarrow 0$
  $s_0 \leftarrow$ initial state
  Initialize θ
  loop
    Choose action a_t based on $\theta_a^\top \beta(s_t)$ and some exploration strategy
    Observe new state s_{t+1} and reward r_t
    $\theta \leftarrow \theta + \alpha(r_t + \gamma \max_a \theta^\top \beta(s_{t+1}, a) - \theta^\top \beta(s_t, a_t))\beta(s_t, a_t)$
    $t \leftarrow t + 1$

Mykal Kochenderfer. *Decision Making Under Uncertainty*, MIT Press 2015

Example: Grid Interpolations



End.



References

- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* 2nd Edition, 2018.
- Mykal Kochenderfer, Tim Wheeler, and Kyle Wray. *Algorithms for Decision Making.* 1st Edition, 2022.

David Silver and Joel Veness, Monte-Carlo Planning in Large POMDPs, *Advances in Neural Information Processing Systems 23 (NIPS 2010)*

- Stanford CS234 (Emma Brunskill)
- Stanford CS228 (Mykal Kochenderfer)