CS 747, Autumn 2022: Lecture 25

Shivaram Kalyanakrishnan

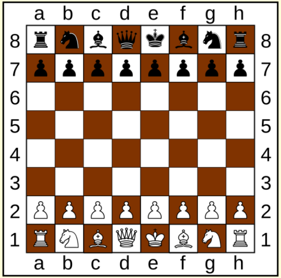
Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Autumn 2022

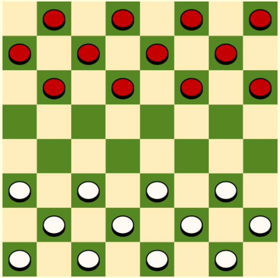
Search in Games, Decision-time Planning in MDPs

- Game trees and minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Summary

Search and Games



[1]

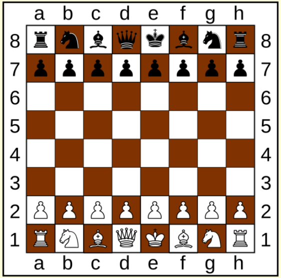


[2]

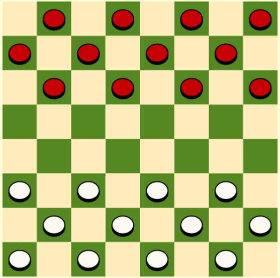
[1] https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

[2] https://commons.wikimedia.org/wiki/File:Draughts.svg.

Search and Games



Chess [1]

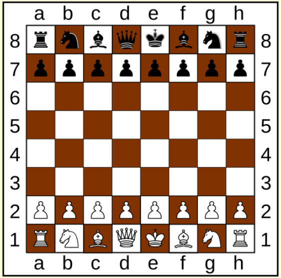


Checkers/Draughts [2]

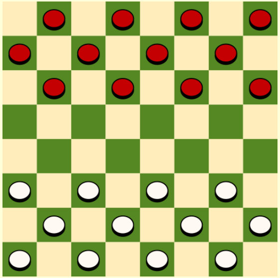
[1] https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg.

[2] https://commons.wikimedia.org/wiki/File:Draughts.svg.

Search and Games



Chess [1]



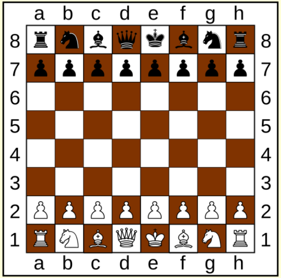
Checkers/Draughts [2]

- Winning at chess/checkers: a search problem?

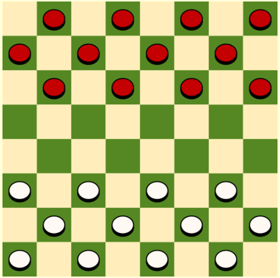
[1] https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

[2] https://commons.wikimedia.org/wiki/File:Draughts.svg.

Search and Games



Chess [1]



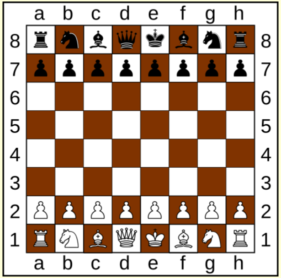
Checkers/Draughts [2]

- Winning at chess/checkers: a search problem?
- What's the main difference from previous examples?

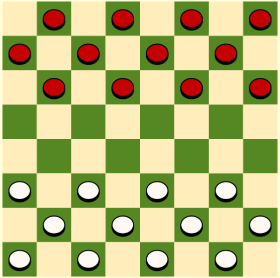
[1] https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

[2] https://commons.wikimedia.org/wiki/File:Draughts.svg.

Search and Games



Chess [1]



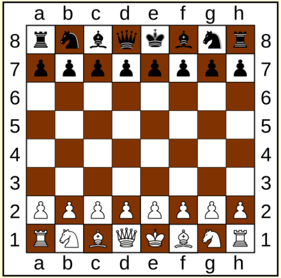
Checkers/Draughts [2]

- Winning at chess/checkers: a search problem?
- What's the main difference from previous examples? There's another player!

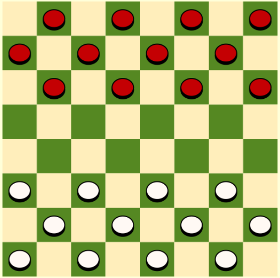
[1] `https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg`. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

[2] `https://commons.wikimedia.org/wiki/File:Draughts.svg`.

Search and Games



Chess [1]



Checkers/Draughts [2]

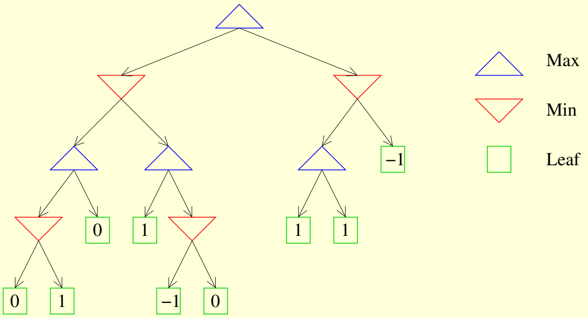
- Winning at chess/checkers: a search problem?
- What's the main difference from previous examples? There's another player!
- Instances of turn-based two player zero-sum games.

[1] https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg. CC image courtesy of ILA-boy on WikiMedia Commons licensed under CC-BY-SA-3.0.

[2] https://commons.wikimedia.org/wiki/File:Draughts.svg.

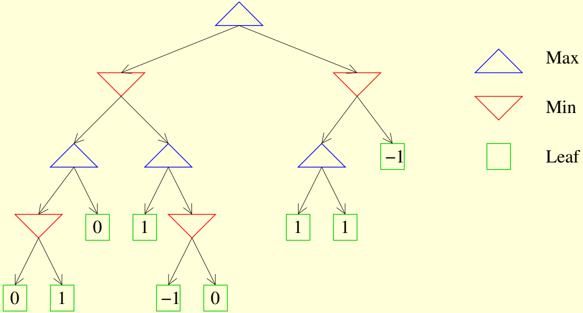
Game Tree

- Assume turn-taking zero sum game played by Max and Min.
- Action costs usually taken as 0, but leaves have value
  -1 (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.
  Value of Min node is minimum of values of children.



Game Tree

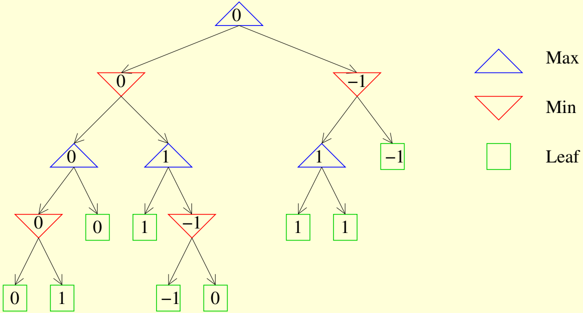
- Assume turn-taking zero sum game played by Max and Min.
- Action costs usually taken as 0, but leaves have value
  -1 (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.
  Value of Min node is minimum of values of children.
- What is the value of the root node?



Game Tree

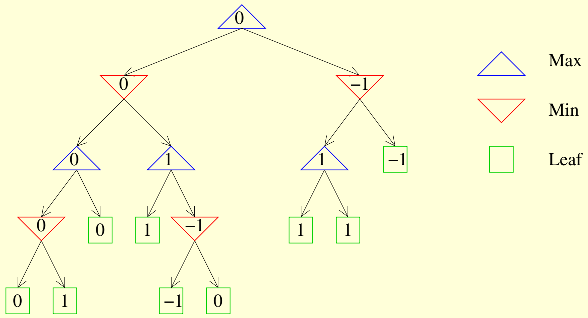
- Assume turn-taking zero sum game played by Max and Min.
- Action costs usually taken as 0, but leaves have value

  -1 (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.
  Value of Min node is minimum of values of children.
- What is the value of the root node?



Game Tree

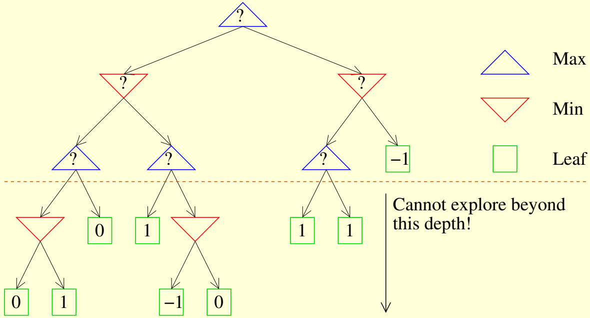
- Assume turn-taking zero sum game played by Max and Min.
- Action costs usually taken as 0, but leaves have value
  -1 (Max loses), 0 (draw), 1 (Max wins).
- Value of Max node is maximum of values of children.
  Value of Min node is minimum of values of children.
- What is the value of the root node?



- In 2007, a massive, long-running computation concluded that the value of the root node for Checkers is 0 (draw).

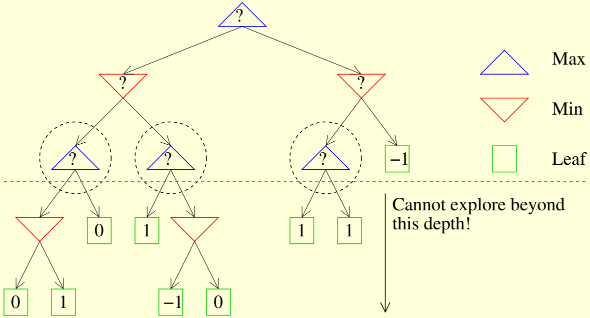
Evaluation Function

- The Checkers tree has $\approx 10^{40}$ nodes; Chess has $\approx 10^{120}$. Infeasible to solve!



Evaluation Function

- The Checkers tree has $\approx 10^{40}$ nodes; Chess has $\approx 10^{120}$. Infeasible to solve!



- At some depth d from current node, estimate node value using features.
- For example, in Chess, set evaluation as

$$w_1 \times \text{Material diff.} + w_2 \times \text{King safety} + w_3 \times \text{pawn strength} + \ldots.$$

- Weights $w_1, w_2, w_3, \ldots$ are tuned or learned.

Search in Games, Decision-time Planning in MDPs

- Game trees and minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Summary

Decision-time planning: Problem

- So far we have assumed that an agent's learning algorithm produces π or Q as output. While acting on-line, the agent just needs a "look up" or associative "forward pass" from any state s to obtain its action.

Decision-time planning: Problem

- So far we have assumed that an agent's learning algorithm produces π or Q as output. While acting on-line, the agent just needs a "look up" or associative "forward pass" from any state s to obtain its action.

- Sometimes π or Q might be difficult to learn in compact form, but a model $M = (T, R)$ (given or learned, exact or approximate) might be available.

Decision-time planning: Problem

- So far we have assumed that an agent's learning algorithm produces π or Q as output. While acting on-line, the agent just needs a "look up" or associative "forward pass" from any state s to obtain its action.

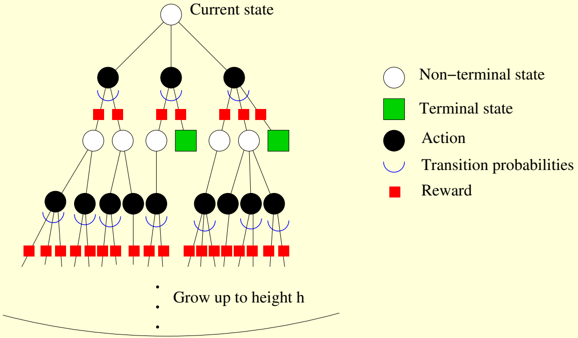
- Sometimes π or Q might be difficult to learn in compact form, but a model $M = (T, R)$ (given or learned, exact or approximate) might be available.

- In decision-time planning, at every time step, we "imagine" possible futures emanating from the current state by using M, and use the computation to decide which action to take.

Decision-time planning: Problem

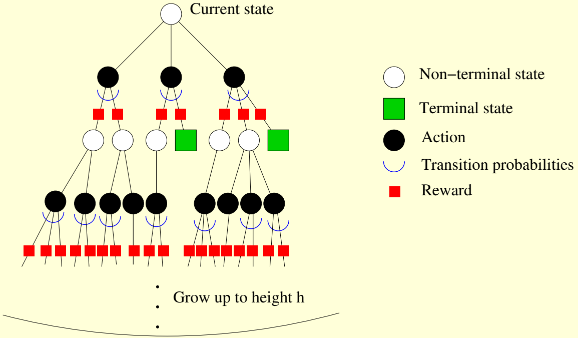
- So far we have assumed that an agent's learning algorithm produces π or Q as output. While acting on-line, the agent just needs a "look up" or associative "forward pass" from any state s to obtain its action.
- Sometimes π or Q might be difficult to learn in compact form, but a model $M = (T, R)$ (given or learned, exact or approximate) might be available.
- In decision-time planning, at every time step, we "imagine" possible futures emanating from the current state by using M, and use the computation to decide which action to take.
- How to rigorously do so?

Tree Search on MDPs



- Expectimax calculation. Set $Q^h \leftarrow \mathbf{0}$ //Leaves.
  For $d = h-1, h-2, \dots, 0$://Bottom-up calculation.
  $V^d(s) \leftarrow \max_{a \in A} Q^{d+1}(s, a)$;
  $Q^d(s, a) \leftarrow \sum_{s' \in S} T(s, a, s')\{R(s, a, s') + \gamma V^d(s')\}$.

Tree Search on MDPs



- Need $h = \Theta(\frac{1}{1-\gamma})$ for sufficient accuracy.
- With branching factor b, tree size is $\Theta(b^h)$. Expensive!
- Often M is only a sampling model (not distribution model).
- Can we avoid expanding (clearly) inferior branches?

Search in Games, Decision-time Planning in MDPs

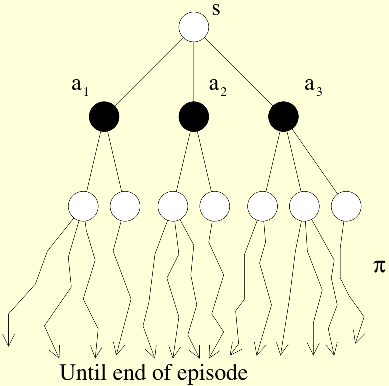
- Game trees and minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Summary

Rollout Policies

- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \text{argmax}_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.

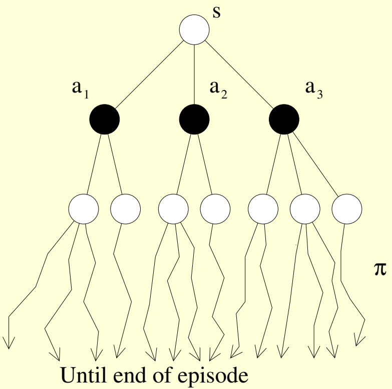
Rollout Policies

- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \text{argmax}_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.
- We implement π' using Monte Carlo rollouts (through M).



Rollout Policies

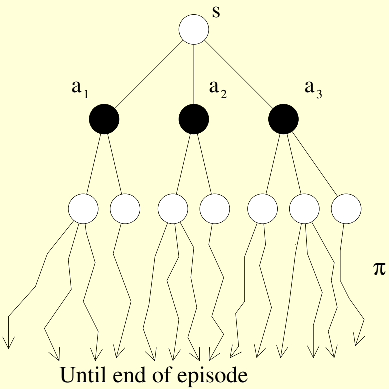
- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \text{argmax}_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.
- We implement π' using Monte Carlo rollouts (through M).



- From current state s, for each action $a \in A$, generate N trajectories by taking a from s and thereafter following π.

Rollout Policies

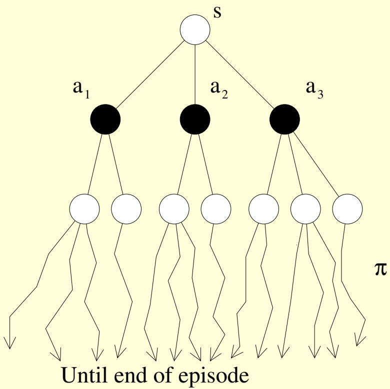
- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \text{argmax}_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.
- We implement π' using Monte Carlo rollouts (through M).



- From current state s, for each action $a \in A$, generate N trajectories by taking a from s and thereafter following π.
- Set $\hat{Q}^{\pi}(s, a)$ as average of episodic returns.

Rollout Policies

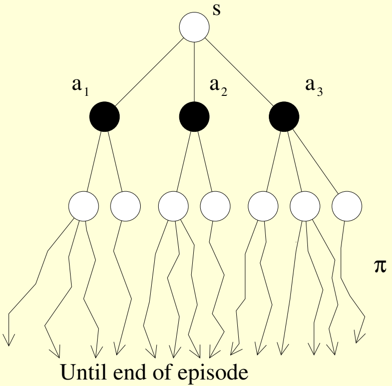
- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \text{argmax}_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.
- We implement π' using Monte Carlo rollouts (through M).



- From current state s, for each action $a \in A$, generate N trajectories by taking a from s and thereafter following π.
- Set $\hat{Q}^{\pi}(s, a)$ as average of episodic returns.
- Take action $\pi'(s) = \text{argmax}_{a \in A} \hat{Q}^{\pi}(s, a)$.

Rollout Policies

- Suppose we have a (look-up) policy π.
- Let policy π' satisfy $\pi'(s) = \text{argmax}_{a \in A} Q^{\pi}(s, a)$ for $s \in S$.
- By the policy improvement theorem, we know $\pi' \succeq \pi$.
- We implement π' using Monte Carlo rollouts (through M).



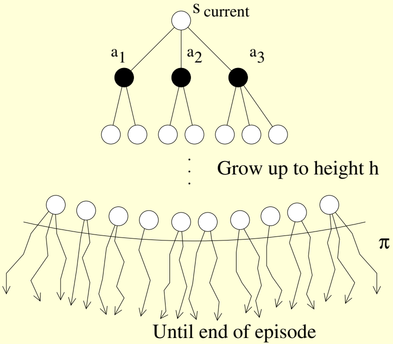
- From current state s, for each action $a \in A$, generate N trajectories by taking a from s and thereafter following π.
- Set $\hat{Q}^{\pi}(s, a)$ as average of episodic returns.
- Take action $\pi'(s) = \text{argmax}_{a \in A} \hat{Q}^{\pi}(s, a)$.
- Repeat same process from next state s'.

Search in Games, Decision-time Planning in MDPs

- Game trees and minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Summary

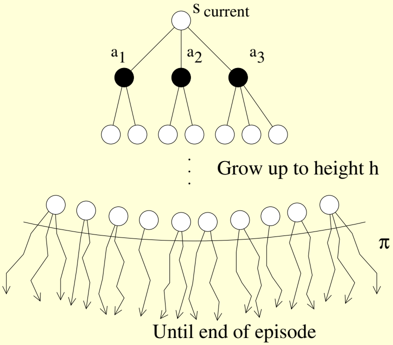
Monte Carlo Tree Search (UCT Algorithm)

- Build out a tree up to height h (say 5–10) from current state s_{current}. "Data" for the tree are samples returned by M.
- For (s, a) pairs reachable from s_{current} in $\leq h$ steps, maintain
  - $Q(s, a)$: average of returns of rollouts passing through (s, a).
  - $ucb(s, a) = Q(s, a) + C_p \sqrt{\frac{\ln(t)}{\text{visits}(s,a)}}$.



Monte Carlo Tree Search (UCT Algorithm)

- Build out a tree up to height h (say 5–10) from current state s_{current}.
  "Data" for the tree are samples returned by M.
- For (s, a) pairs reachable from s_{current} in $\leq h$ steps, maintain
  - $Q(s, a)$: average of returns of rollouts passing through (s, a).
  - $ucb(s, a) = Q(s, a) + C_p \sqrt{\frac{\ln(t)}{\text{visits}(s,a)}}$.

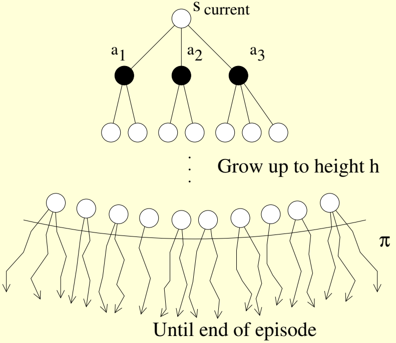


Repeat N times from s_{current}:

1. Generate trajectory by calling M. From stored state s, "take" action $\text{argmax}_{a \in A}\, ucb(s, a)$; from leaf follow rollout policy π until end of episode.

Monte Carlo Tree Search (UCT Algorithm)

- Build out a tree up to height h (say 5–10) from current state s_{current}. "Data" for the tree are samples returned by M.
- For (s, a) pairs reachable from s_{current} in $\leq h$ steps, maintain
  - $Q(s, a)$: average of returns of rollouts passing through (s, a).
  - $ucb(s, a) = Q(s, a) + C_p \sqrt{\frac{\ln(t)}{\text{visits}(s,a)}}$.



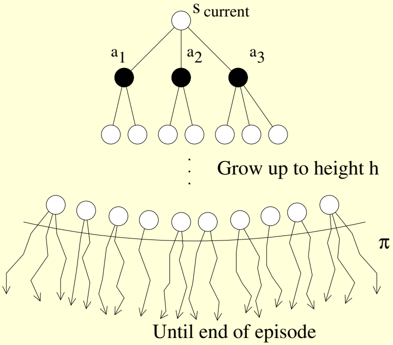
Repeat N times from s_{current}:

1. Generate trajectory by calling M. From stored state s, "take" action $\text{argmax}_{a \in A}\, ucb(s, a)$; from leaf follow rollout policy π until end of episode.

2. Update Q, ucb for (s, a) pairs visited in trajectory.

Monte Carlo Tree Search (UCT Algorithm)

- Build out a tree up to height h (say 5–10) from current state s_{current}. "Data" for the tree are samples returned by M.
- For (s, a) pairs reachable from s_{current} in $\leq h$ steps, maintain
  - $Q(s, a)$: average of returns of rollouts passing through (s, a).
  - $ucb(s, a) = Q(s, a) + C_p \sqrt{\frac{\ln(t)}{\text{visits}(s,a)}}$.



Repeat N times from s_{current}:

1. Generate trajectory by calling M. From stored state s, "take" action $\text{argmax}_{a \in A}\, ucb(s, a)$; from leaf follow rollout policy π until end of episode.

2. Update Q, ucb for (s, a) pairs visited in trajectory.

Take action $\text{argmax}_{a \in A}\, ucb(s_{\text{current}}, a)$.

Monte Carlo Tree Search (UCT Algorithm)

- Main parameters of UCT: rollout policy π, search tree height h, number of rollouts N.
- π typically an associative/look-up policy, often even a random policy.
- Better guarantees as h is increased (if $N = \infty$).
- In practice N limited by available "think" time.

Monte Carlo Tree Search (UCT Algorithm)

- Main parameters of UCT: rollout policy π, search tree height h, number of rollouts N.
- π typically an associative/look-up policy, often even a random policy.
- Better guarantees as h is increased (if $N = \infty$).
- In practice N limited by available "think" time.
- C_p in the UCB formula needs to be large to deal with nonstationarity (from changes downstream).

Monte Carlo Tree Search (UCT Algorithm)

- Main parameters of UCT: rollout policy π, search tree height h, number of rollouts N.
- π typically an associative/look-up policy, often even a random policy.
- Better guarantees as h is increased (if $N = \infty$).
- In practice N limited by available "think" time.
- C_p in the UCB formula needs to be large to deal with nonstationarity (from changes downstream).
- In general there could be multiple paths to any particular stored (s, a) pair starting from s_{current}.

Monte Carlo Tree Search (UCT Algorithm)

- Main parameters of UCT: rollout policy π, search tree height h, number of rollouts N.
- π typically an associative/look-up policy, often even a random policy.
- Better guarantees as h is increased (if $N = \infty$).
- In practice N limited by available "think" time.
- C_p in the UCB formula needs to be large to deal with nonstationarity (from changes downstream).
- In general there could be multiple paths to any particular stored (s, a) pair starting from s_{current}.
- UCT focuses attention on rewarding regions of state space.
- Rollouts can easily be parallelised.
- Extremely successful algorithm in practice.

Search in Games, Decision-time Planning in MDPs

- Game trees and minimax search
- Decision-time planning in MDPs
  - Problem
  - Rollout policies
  - Monte Carlo tree search
- Summary

Search in On-line Decision Making

- Key requirement: simulator (model).
- More computationally expensive than lookup of π or Q.
- MCTS with rollout policies an effective approach to handle stochasticity as well as large state spaces.
- Learning (say an evaluation function) can also help solution quality of search in practice.
- Proof of all these claims: AlphaGo!

Search in On-line Decision Making

- Key requirement: simulator (model).
- More computationally expensive than lookup of π or Q.
- MCTS with rollout policies an effective approach to handle stochasticity as well as large state spaces.
- Learning (say an evaluation function) can also help solution quality of search in practice.
- Proof of all these claims: AlphaGo! Coming up in next class.