Wrapup: AlphaZero + Warmup for UCB-VI

Lucas Janson and Sham Kakade

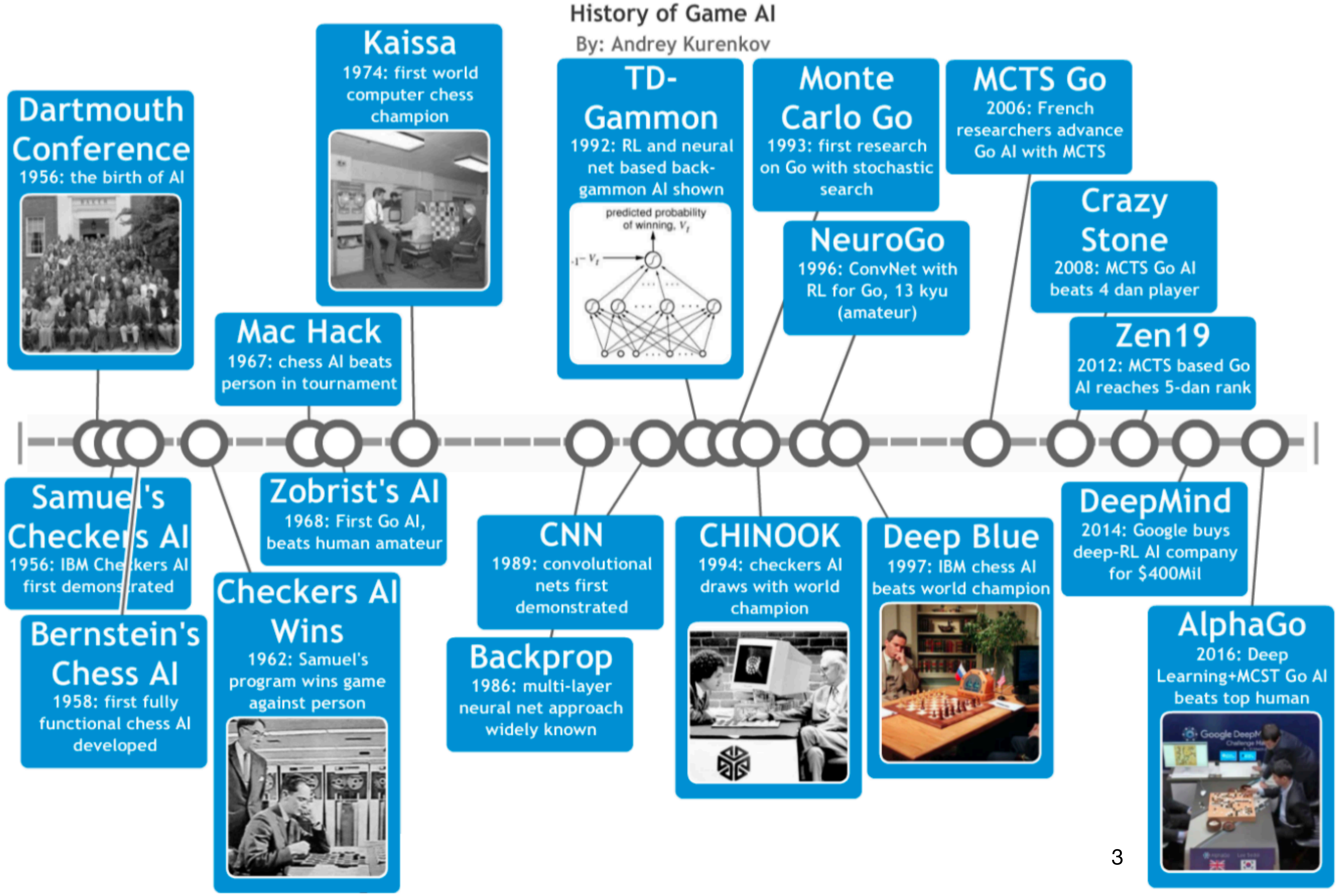
CS/Stat 184: Introduction to Reinforcement Learning

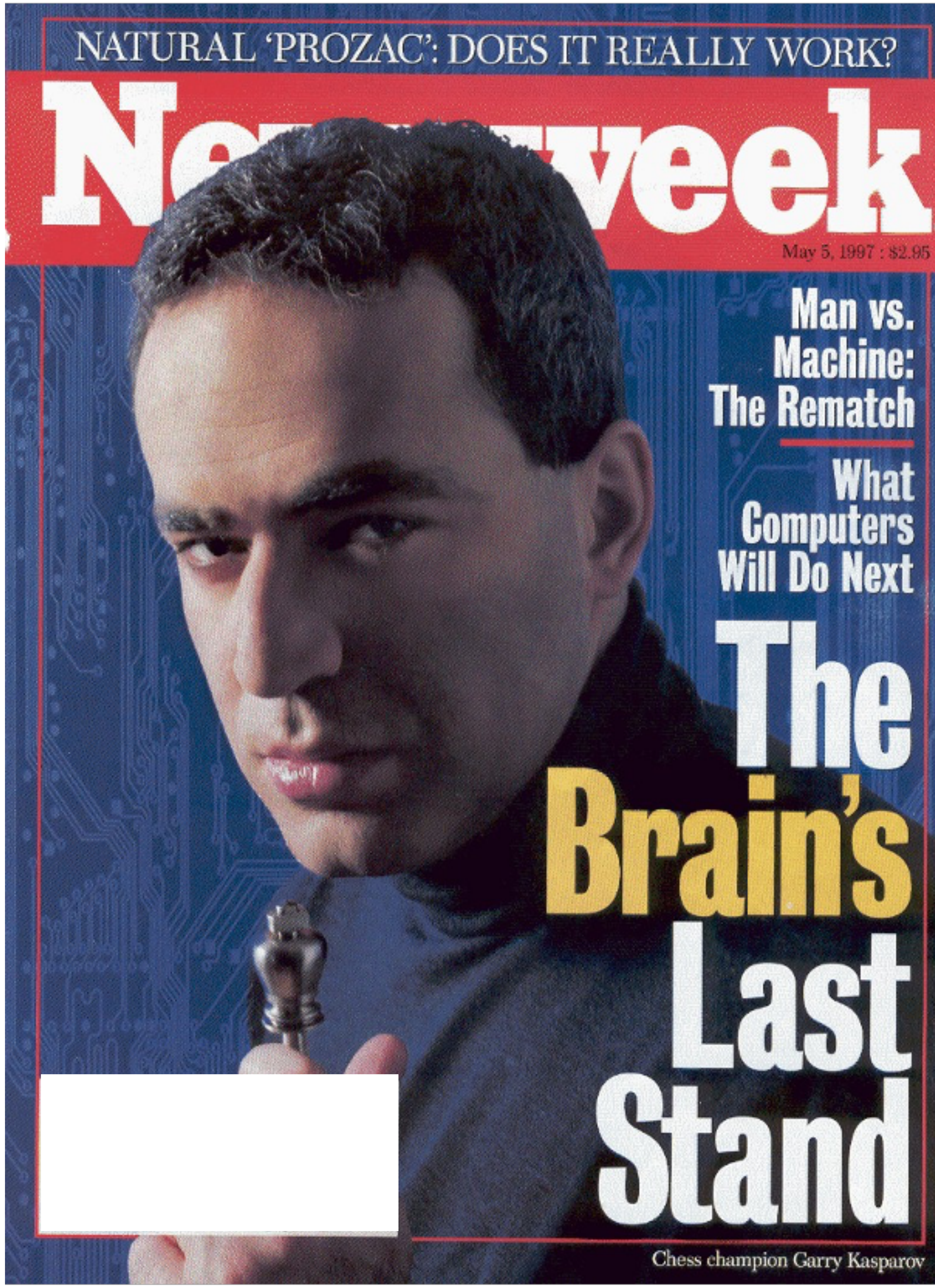
Fall 2023

Recap++

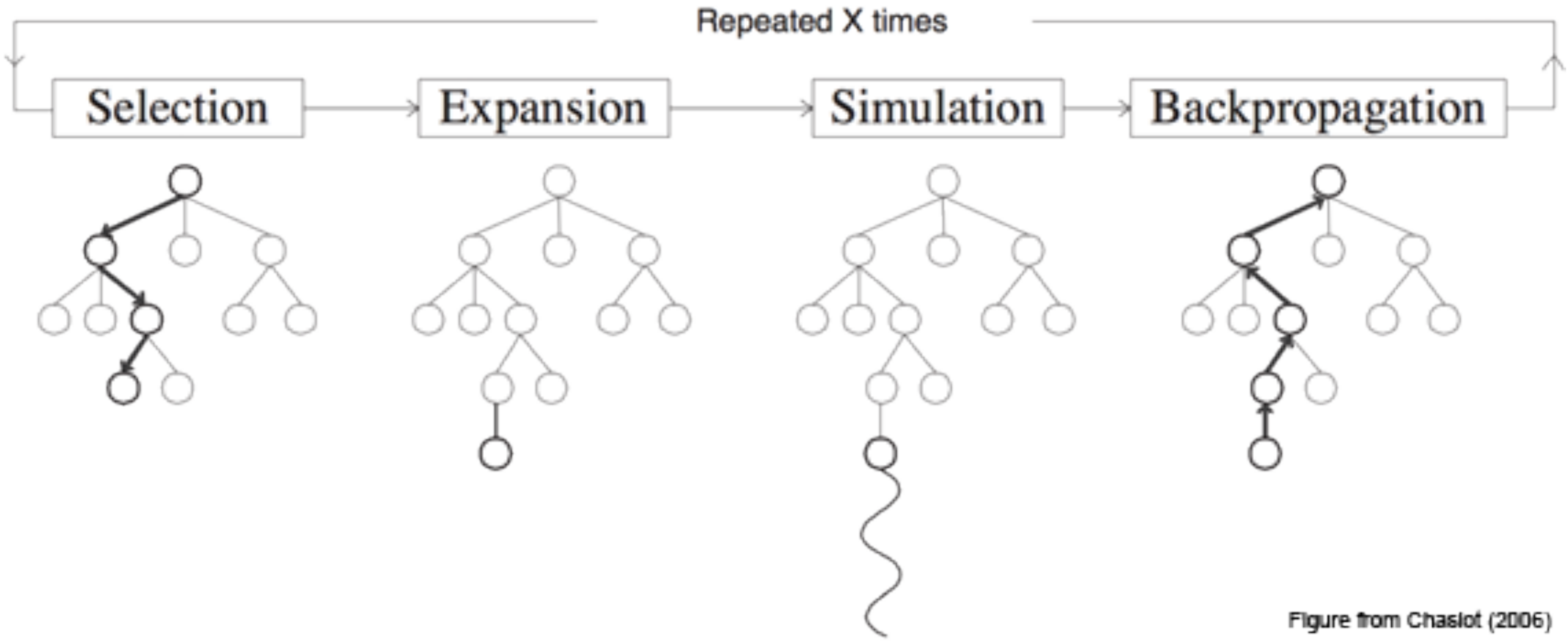
Fascination with AI and Games…

- DeepBlue v. Kasparov (1997)
  - winning in chess wasn't a good indicator of "progress in AI"





MCTS:
Monte Carlo Tree Search



- AlphaBeta pessimistic approach may not lead to effective heuristics.
- **MCTS: to decide on an action, we build a lookahead tree.** (and repeat)
  Input: game state/node "R"; Output: single action to take at R
  - For two player games
  - When building the lookahead tree, we use a heuristic to estimate the "value" of taking action "a" at any node "s"
    (no minmax values estimated).
- **Applicable to "small" games.**

ActionSelectionSubroutine

Input: game state ("root node" R), # playouts N

For rollouts $t = 1 : N$

1. Obtain the t-th roll-out: While CurrentNode $\notin$ {win, lose}
   a. For player $X \in \{A, B\}$, at current state s, define $s' = NextState(s, a)$ and define:

      $$\text{UCB score}_t(s, a) = \frac{\text{\#wins for player X at } s'}{\text{\#visits to } s'} + C \times \sqrt{\frac{\log(\text{total visits to } s)}{\text{\#visits to } s'}}$$

   b. Choose and "take" action:

      $$\hat{a} = \arg\max_a \text{UCB score}(s, a)$$

2. Update stats: For all visited states s in this "roll-out",
   c. update visit counts:

      $$[\text{\#visits to } s'] = [\text{\#visits to } s'] + 1$$

   d. for winner X and if s was visited by X:

      $$[\text{\#wins for X at } s'] = [\text{\#wins for X at } s'] + 1$$

   (data structure: only need to keep track of stats at visited states)

Output: return the action $\hat{a} = \arg\max_a \text{UCB score}_N(\text{Root Node } R, a)$

Example Diagram:

- **Obtaining the t-th rollout** (steps called **Selection/Expansion/Simulation):**
  Start from "root R" and select successive child nodes until a the game ends.
  - At state s (for player X), choose action a leading to $s' = NextState(s, a)$ which maximizes:

$$\text{UCB score}_t(s, a) = \frac{\text{\#wins for player X at } s'}{\text{\#visits to } s'} + C \times \sqrt{\frac{\log(\text{total visits to } s)}{\text{\#visits to } s'}}$$

Example Diagram:

- **The update step for the t-th rollout ("backpropagation"):**
  Use the result of the rollout to update information in the nodes on the visited path:
  [#visits to s'] $=$ [#visits to s'] $+ 1$
  [#wins for X at s'] $=$ [#wins for X at s'] $+ 1$

Example Diagram:

- **Repeat all steps N times**, (so we do N roll-outs)
- **select the "best" action at the root node R** (the game state):

$$\hat{a} = \arg\max_a \text{UCB score}_N(\text{Root Node } R, a)$$

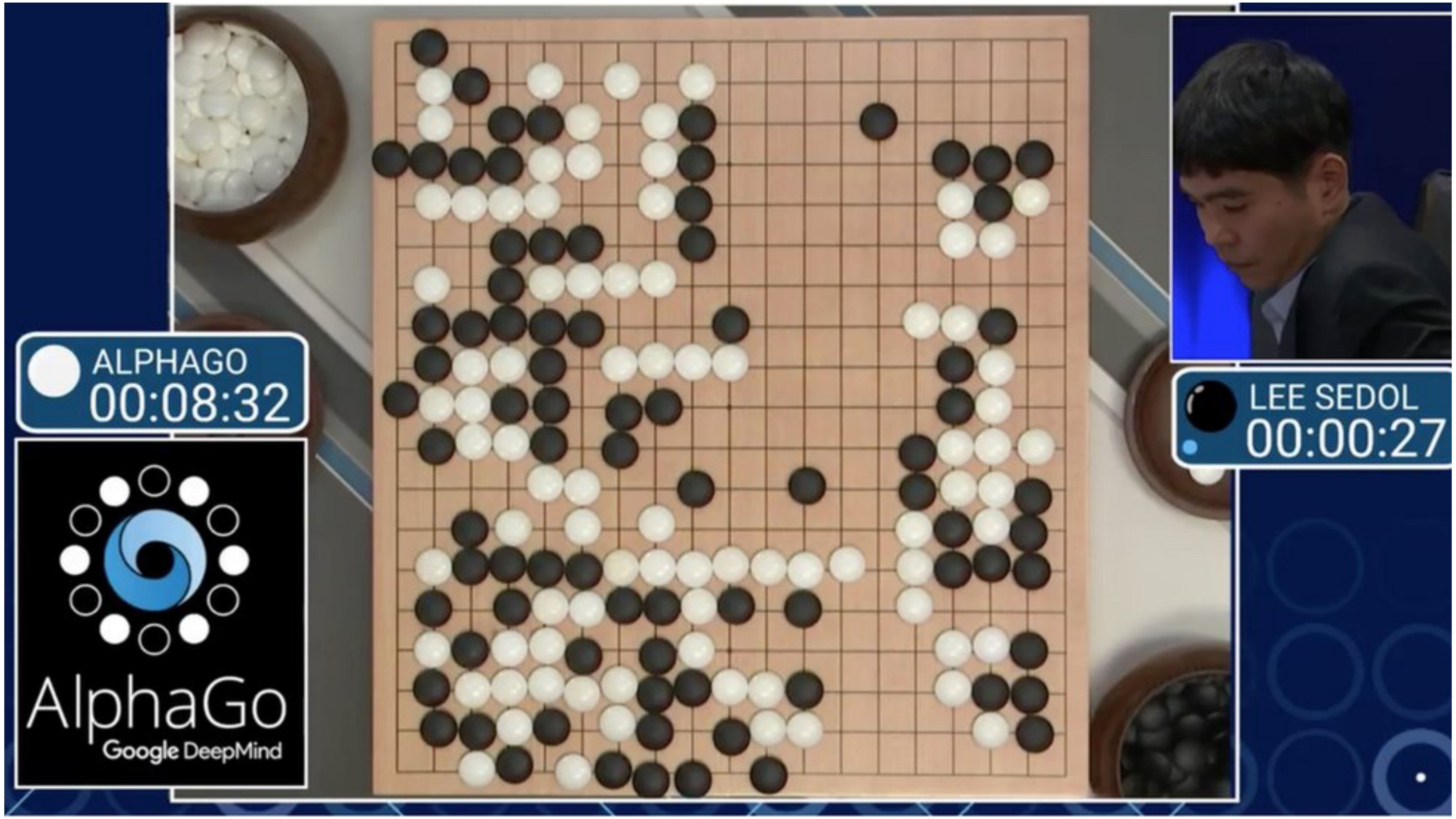
Today

- Recap
- Game Playing: AlphaBeta Search/Rule Based Systems
- MCTS
- AlphaZero and Self-Play

AlphaGo

AlphaGo versus Lee Sedol
4–1

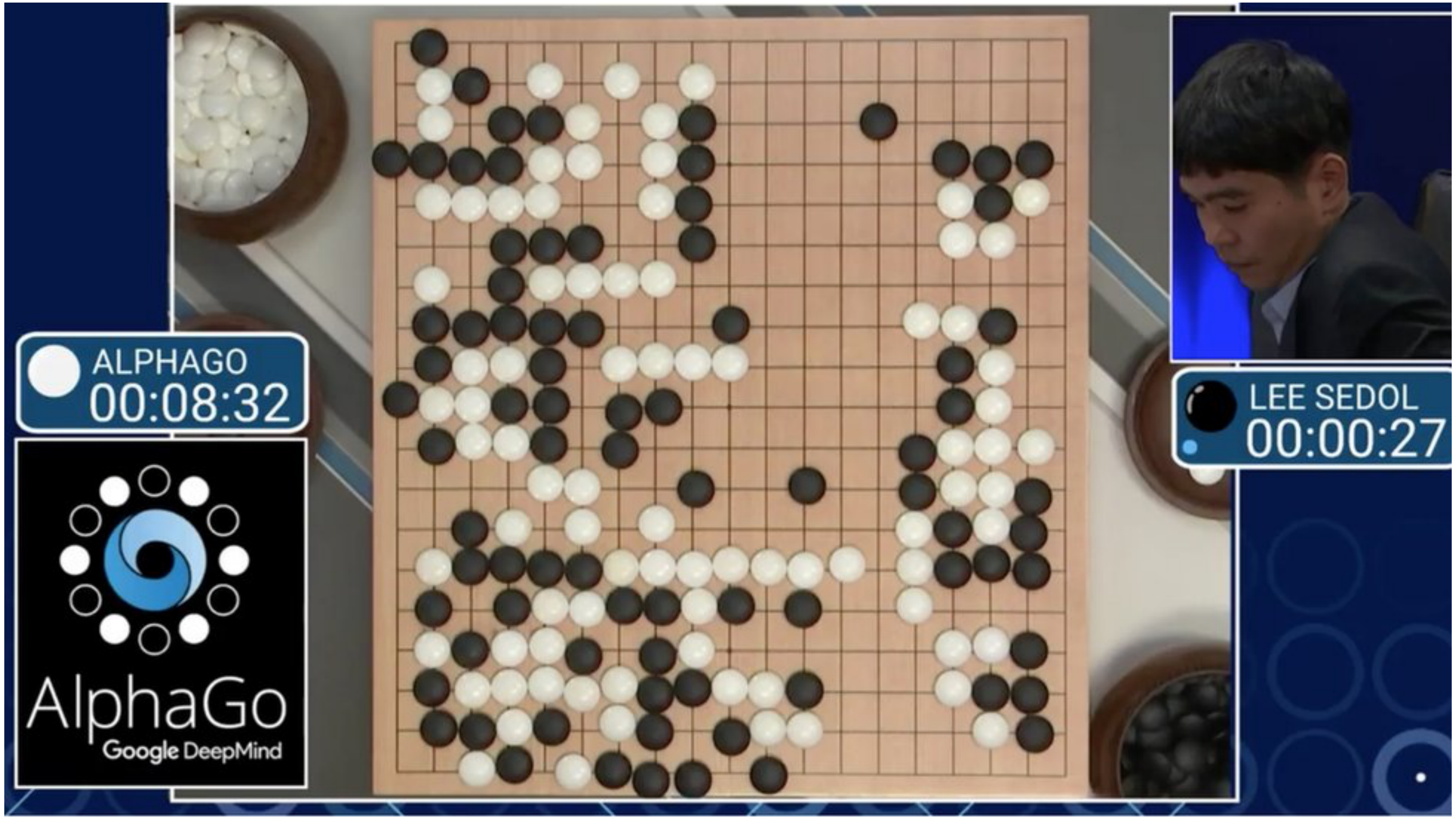
Seoul, South Korea, 9–15 March 2016	
Game one	AlphaGo W+R
Game two	AlphaGo B+R
Game three	AlphaGo W+R
Game four	Lee Sedol W+R
Game five	AlphaGo W+R



AlphaGo

AlphaGo versus Lee Sedol
4–1

Seoul, South Korea, 9–15 March 2016	
Game one	AlphaGo W+R
Game two	AlphaGo B+R
Game three	AlphaGo W+R
Game four	Lee Sedol W+R
Game five	AlphaGo W+R



- Lots of moving parts:
  - **Imitation Learning:** first, the algo estimates the values from historical games.
  - It then uses an **MCTS-stye lookahead** with **learned value functions**.
- **AlphaZero** (2017) is a simpler more successful approach.

AlphaZero

- AlphaZero: MCTS + DeepLearning
  - There is a value network and policy network:
    - a **value network** estimating for the state of the board $v_\theta(s)$
    - A **policy network** $p_\theta(a \mid s)$ that is a probability vector over all possible actions. (think $p_\theta(a \mid s)$ of as an estimate of which actions the "subroutine" selects)
  - There is a **termination condition** for each rollout, e.g. each rollout is no longer than K steps

AlphaZero: ActionSelectionSubroutine

AlphaZero: ActionSelectionSubroutine

Input: game state ("root node" R), # playouts N, **value network** $v_\theta(s)$, **policy network** $p_\theta(a \mid s)$

AlphaZero: ActionSelectionSubroutine

Input: game state ("root node" R), # playouts N, **value network** $v_\theta(s)$, **policy network** $p_\theta(a \mid s)$

For rollouts $t = 1 : N$

AlphaZero: ActionSelectionSubroutine

Input: game state ("root node" R), # playouts N, **value network** $v_\theta(s)$, **policy network** $p_\theta(a \mid s)$

For rollouts $t = 1 : N$

1. Obtain the t-th roll-out: While CurrentNode $\notin$ {termination condition}
   a. At current state s, define $s' = NextState(s, a)$ and define:

      $$\text{UCB score}_t(s, a) = \text{AvValue}(s') + C \cdot p_\theta(a \mid s) \cdot \sqrt{\frac{\log(\text{total visits to s})}{\text{\#visits to } s'}}$$

   b. Choose and "take" action:

      $$\hat{a} = \arg\max_a \text{UCB score}_t(s, a)$$

AlphaZero: ActionSelectionSubroutine

Input: game state ("root node" R), # playouts N, **value network** $v_\theta(s)$, **policy network** $p_\theta(a \mid s)$

For rollouts $t = 1 : N$

1. Obtain the t-th roll-out: While CurrentNode $\notin$ {termination condition}
   a. At current state s, define $s' = NextState(s, a)$ and define:

      $$\text{UCB score}_t(s, a) = \text{AvValue}(s') + C \cdot p_\theta(a \mid s) \cdot \sqrt{\frac{\log(\text{total visits to s})}{\text{\#visits to } s'}}$$

   b. Choose and "take" action:

      $$\hat{a} = \arg\max_a \text{UCB score}_t(s, a)$$

2. Update stats: For all visited states s in this "roll-out",
   c. Let C be the terminal node in this rollout.
   d. Update counts: $N(s) \leftarrow N(s) + 1$
   e. If state s was for player A: $AvValue(s) \leftarrow \frac{N(s)}{N(s)+1} AvValue(s) + \frac{1}{N(s)+1} v_\theta(C)$
   f. If state s was for player B: same update but with $-v_\theta(C)$

AlphaZero: ActionSelectionSubroutine

Input: game state ("root node" R), # playouts N, **value network** $v_\theta(s)$, **policy network** $p_\theta(a \mid s)$

For rollouts $t = 1 : N$

1. Obtain the t-th roll-out: While CurrentNode $\notin$ {termination condition}
   a. At current state s, define $s' = NextState(s, a)$ and define:

   $$\text{UCB score}_t(s, a) = \text{AvValue}(s') + C \cdot p_\theta(a \mid s) \cdot \sqrt{\frac{\log(\text{total visits to s})}{\text{\#visits to } s'}}$$

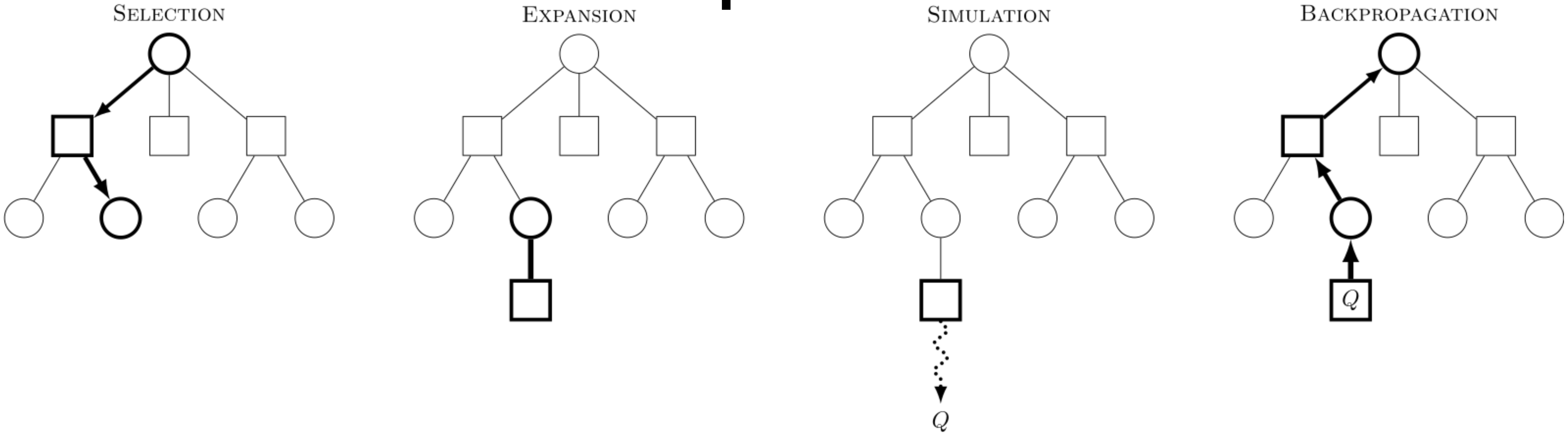
   b. Choose and "take" action:

   $$\hat{a} = \arg\max_a \text{UCB score}_t(s, a)$$

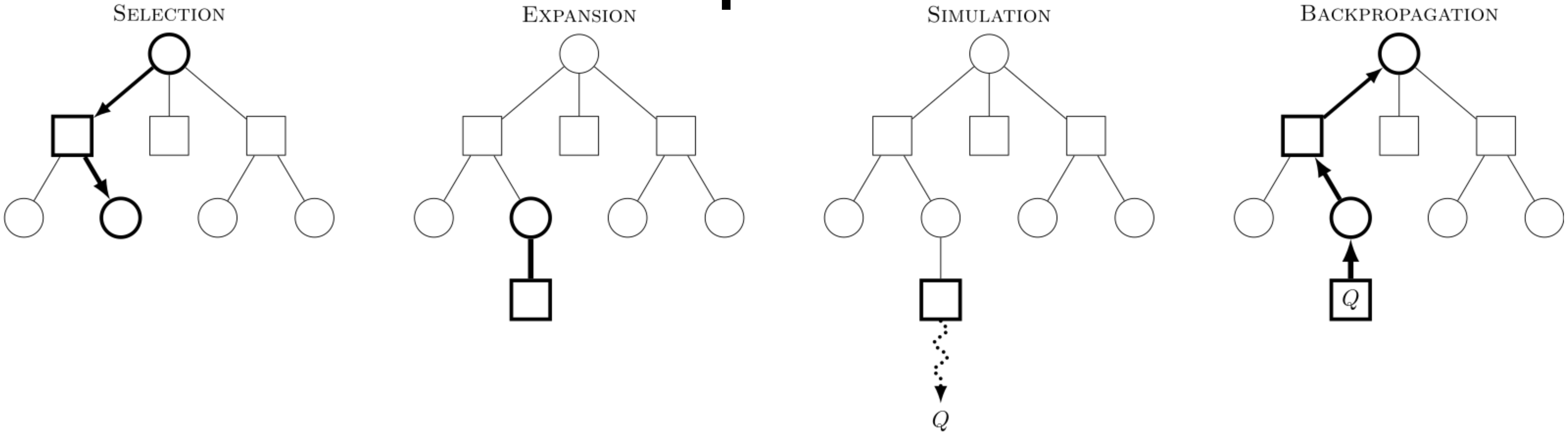
2. Update stats: For all visited states s in this "roll-out",
   c. Let C be the terminal node in this rollout.
   d. Update counts: $N(s) \leftarrow N(s) + 1$
   e. If state s was for player A: $AvValue(s) \leftarrow \frac{N(s)}{N(s)+1} AvValue(s) + \frac{1}{N(s)+1} v_\theta(C)$
   f. If state s was for player B: same update but with $-v_\theta(C)$

Output: return the action $\hat{a} = \arg\max_a \text{UCB score}_N(\text{Root Node } R, a)$

AlphaZero

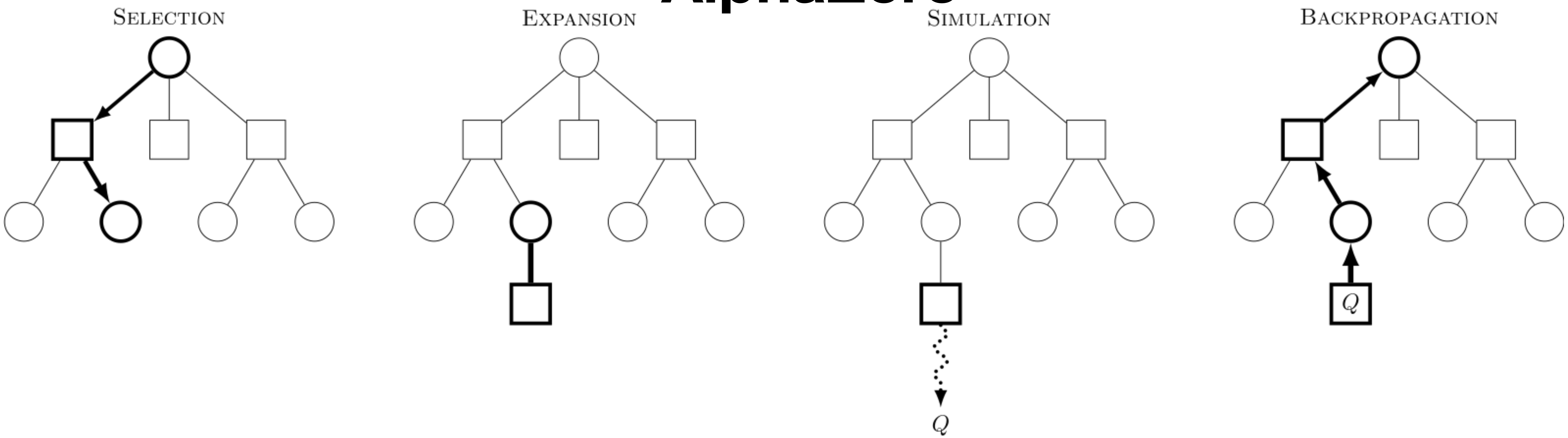


AlphaZero



- **Obtaining the t-th rollout** (steps called **Selection/Expansion/Simulation):**

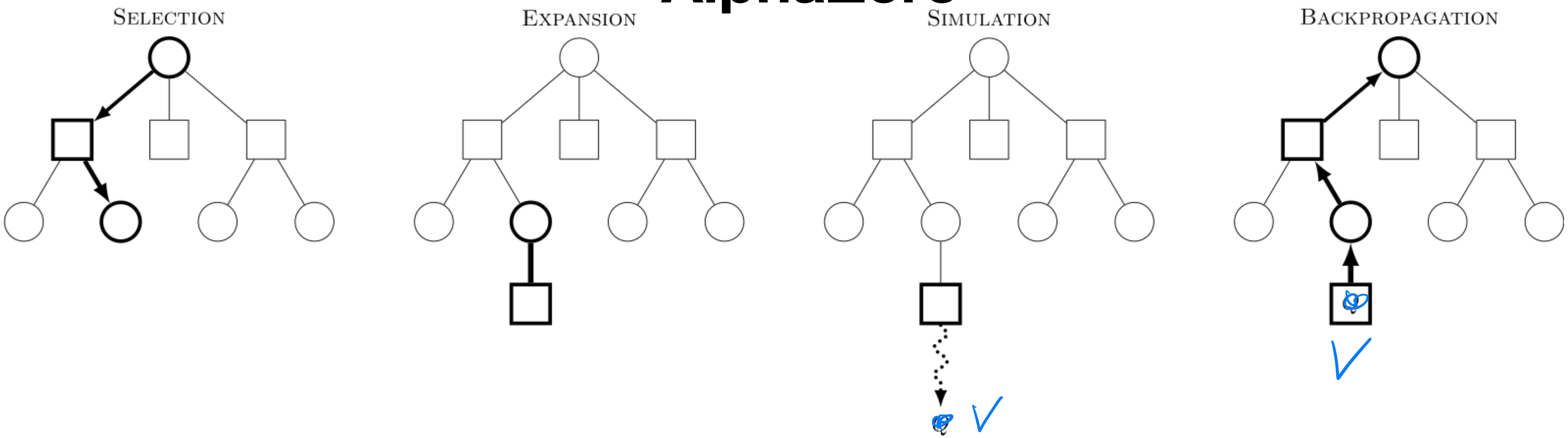
AlphaZero



- **Obtaining the t-th rollout** (steps called **Selection/Expansion/Simulation):**
  - Start from “root R” (current game) and do a rollout of no more than K steps.
  - At state s, choose action a leading to $s' = NextState(s, a)$ which maximizes:

$$\text{UCB score}(a) = \text{AvValue}(s') + C \cdot p_\theta(a \mid s) \cdot \sqrt{\frac{\log(\text{total visits to s})}{\text{\#visits to s'}}}$$

AlphaZero

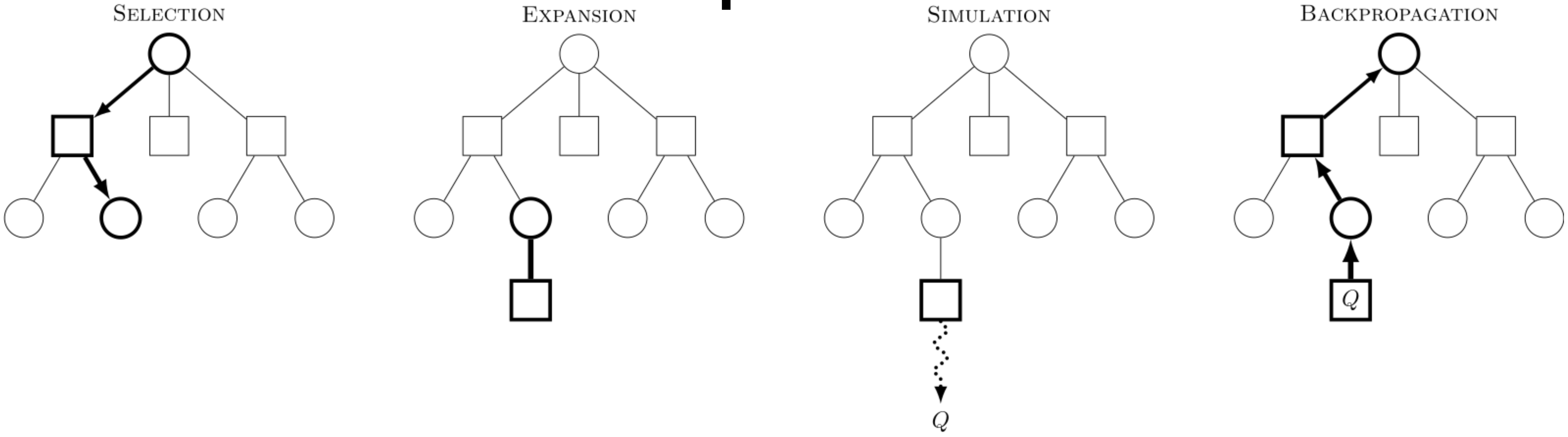


- **Obtaining the t-th rollout** (steps called **Selection/Expansion/Simulation):**
  - Start from "root R" (current game) and do a rollout of no more than K steps.
  - At state s, choose action a leading to $s' = NextState(s, a)$ which maximizes:

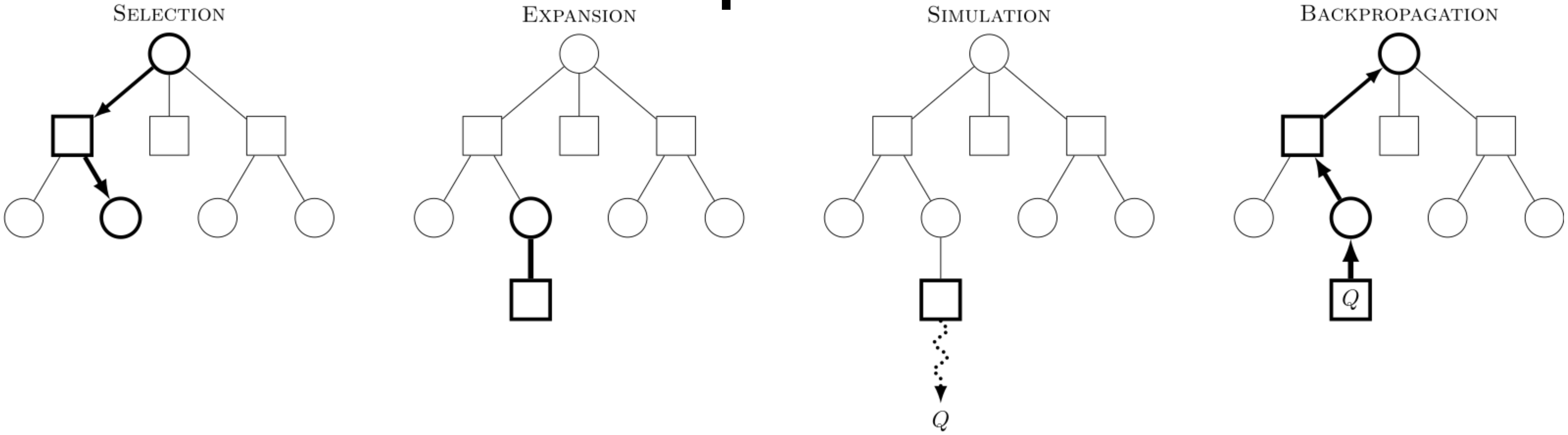
$$\text{UCB score}(a) = \text{AvValue}(s') + C \cdot p_\theta(a \,|\, s) \cdot \sqrt{\frac{\log(\text{total visits to s})}{\text{\#visits to s'}}}$$

  - We'll specify AverageValue(s') soon.
    - in MCTS, this average was $\frac{\text{\#wins at s'}}{\text{\#visits to s'}}$

AlphaZero

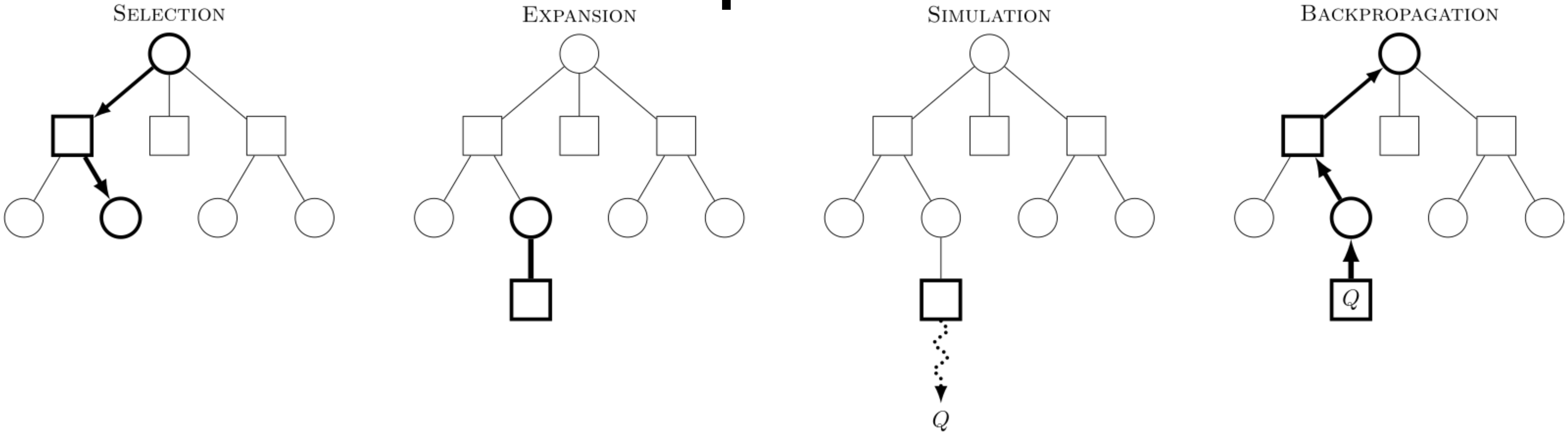


AlphaZero



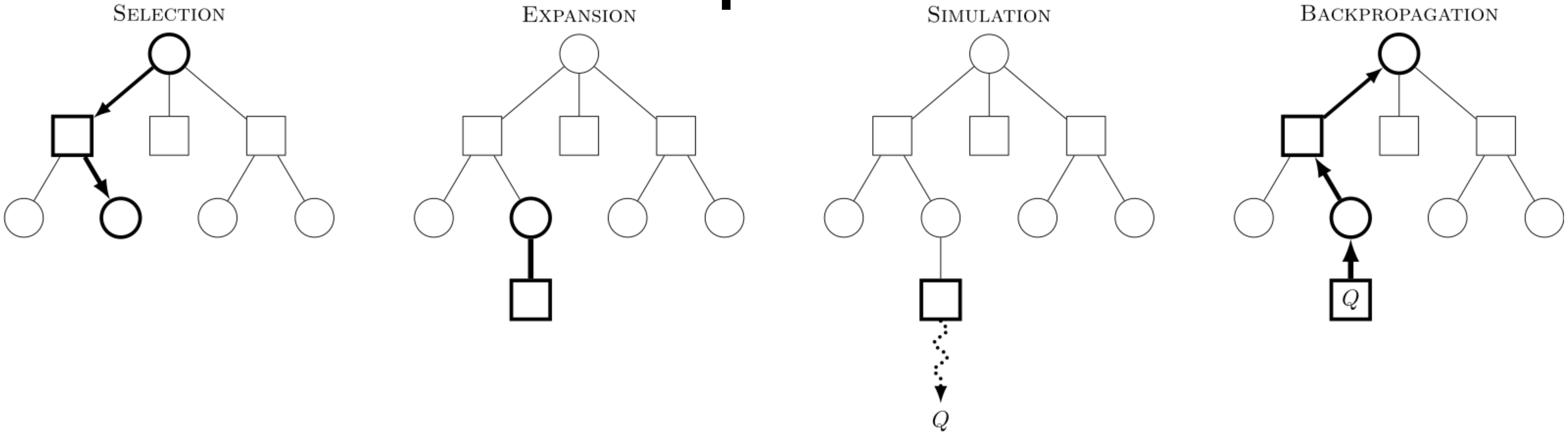
- **The update step for the t-th rollout (“backpropagation”):**

AlphaZero



- **The update step for the t-th rollout ("backpropagation"):**
  - Suppose the Simulation ends at node C after K steps.

AlphaZero



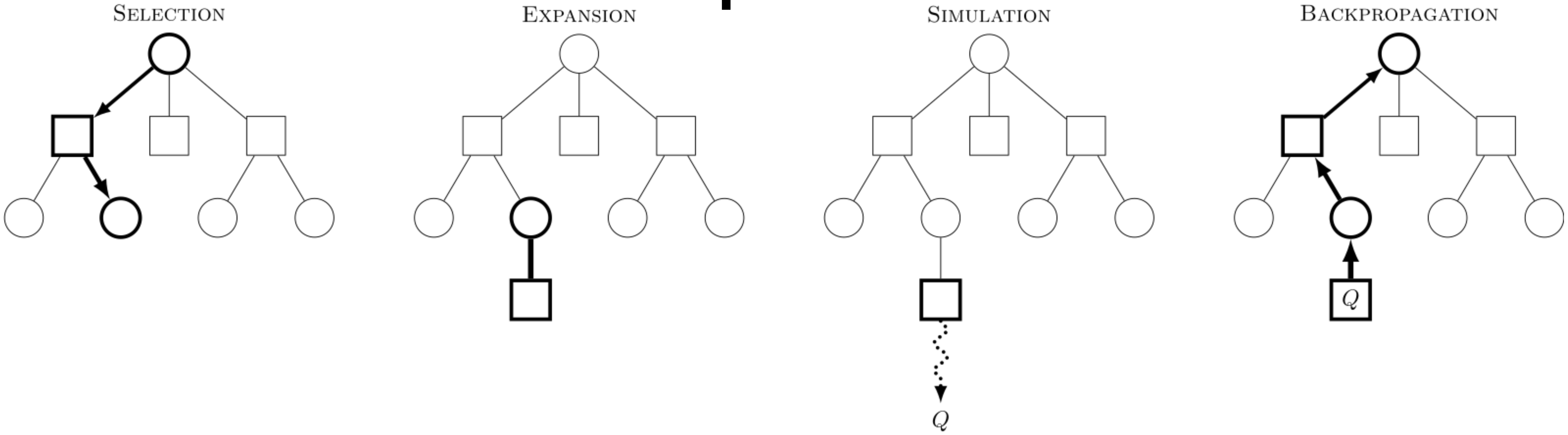
- **The update step for the t-th rollout ("backpropagation"):**
  - Suppose the Simulation ends at node C after K steps.
  - Update AvValue(s) on all s in the path from the root R to C (for player A):

$$AvValue(s) \leftarrow \frac{N(s)}{N(s)+1} AvValue(s) + \frac{1)}{N(s)+1} v_\theta(C)$$

$$N(s) \leftarrow N(s) + 1$$

(use negative values for player B)

AlphaZero



- **The update step for the t-th rollout ("backpropagation"):**
  - Suppose the Simulation ends at node C after K steps.
  - Update AvValue(s) on all s in the path from the root R to C (for player A):

$$AvValue(s) \leftarrow \frac{N(s)}{N(s)+1} AvValue(s) + \frac{1)}{N(s)+1} v_\theta(C)$$

$$N(s) \leftarrow N(s) + 1$$

(use negative values for player B)

- **Repeat all steps N times**, then **select "best" action at the root node R** (the game state).

AlphaZero: Learning

AlphaZero: Learning

- Input: dataset of M self-play games

AlphaZero: Learning

- Input: dataset of M self-play games
  - The point in the dataset is of the (s_t, a_t, R_t), which says action a_t was taken in state s_t and the game resulted in outcome R_t (e.g. win=1,loose=-1, draw=0)

AlphaZero: Learning

- Input: dataset of M self-play games
  - The point in the dataset is of the (s_t, a_t, R_t), which says action a_t was taken in state s_t and the game resulted in outcome R_t (e.g. win=1,loose=-1, draw=0)
- Supervised Learning: try learn θ so to predict the actions and rewards

$\theta_{t+1} \leftarrow$ $\min_{\theta} L(\theta)$

$$Loss(\theta) = \sum_t (v_\theta(s_t) - R_t)^2 - \log p_\theta(a_t \mid s_t)$$

$$Loss_{value}(\theta_1) = \sum_t (V_{\theta_1}(s_t) - R_t)^2$$

$$Loss_{policy}(\theta_2) = -\sum_t \log p_{\theta_2}(a_t \mid s_t)$$

AlphaZero: Learning

- Input: dataset of M self-play games
  - The point in the dataset is of the (s_t, a_t, R_t), which says action a_t was taken in state s_t and the game resulted in outcome R_t (e.g. win=1,loose=-1, draw=0)
- Supervised Learning: try learn θ so to predict the actions and rewards

$$Loss(\theta) = \sum_t (v_\theta(s_t) - R_t)^2 - \log p_\theta(a_t \mid s_t)$$

AlphaZero was trained solely via self-play, using 5,000 first-generation TPUs to generate the games and 64 second-generation TPUs to train the neural networks. In parallel, the in-training AlphaZero was periodically matched against its benchmark (Stockfish, elmo, or AlphaGo Zero) in

Comparing Monte Carlo tree search searches, AlphaZero searches just 80,000 positions per second in chess and 40,000 in shogi, compared to 70 million for Stockfish and 35 million for elmo. AlphaZero compensates for the lower number of evaluations by using its deep neural network to

Chess [edit]

In AlphaZero's chess match against Stockfish 8 (2016 TCEC world champion), each program was given one minute per move. Stockfish was allocated 64 threads and a hash size of 1 GB,[1] a setting that Stockfish's Tord Romstad later criticized as suboptimal.[7][note 1] AlphaZero was trained on chess for a total of nine hours before the match. During the match, AlphaZero ran on a single machine with four application-specific TPUs. In 100 games from the normal starting position, AlphaZero won 25 games as White, won 3 as Black, and drew the remaining 72.[8] In a series of twelve, 100-game matches (of unspecified time or resource constraints) against Stockfish starting from the 12 most popular human openings, AlphaZero won 290, drew 886 and lost 24.[1]

Shogi [edit]

AlphaZero was trained on shogi for a total of two hours before the tournament. In 100 shogi games against elmo (World Computer Shogi Championship 27 summer 2017 tournament version with YaneuraOu 4.73 search), AlphaZero won 90 times, lost 8 times and drew twice.[8] As in the chess games, each program got one minute per move, and elmo was given 64 threads and a hash size of 1 GB.[1]

Go [edit]

After 34 hours of self-learning of Go and against AlphaGo Zero, AlphaZero won 60 games and lost 40.[1][8]

Comparing Monte Carlo tree search searches, AlphaZero searches just 80,000 positions per second in chess and 40,000 in shogi, compared to 70 million for Stockfish and 35 million for elmo. AlphaZero compensates for the lower number of evaluations by using its deep neural network to

Chess [edit]

In AlphaZero's chess match against Stockfish 8 (2016 TCEC world champion), each program was given one minute per move. Stockfish was allocated 64 threads and a hash size of 1 GB,[1] a setting that Stockfish's Tord Romstad later criticized as suboptimal.[7][note 1] AlphaZero was trained on chess for a total of nine hours before the match. During the match, AlphaZero ran on a single machine with four application-specific TPUs. In 100 games from the normal starting position, AlphaZero won 25 games as White, won 3 as Black, and drew the remaining 72.[8] In a series of twelve, 100-game matches (of unspecified time or resource constraints) against Stockfish starting from the 12 most popular human openings, AlphaZero won 290, drew 886 and lost 24.[1]

Shogi [edit]

AlphaZero was trained on shogi for a total of two hours before the tournament. In 100 shogi games against elmo (World Computer Shogi Championship 27 summer 2017 tournament version with YaneuraOu 4.73 search), AlphaZero won 90 times, lost 8 times and drew twice.[8] As in the chess games, each program got one minute per move, and elmo was given 64 threads and a hash size of 1 GB.[1]

Go [edit]

After 34 hours of self-learning of Go and against AlphaGo Zero, AlphaZero won 60 games and lost 40.[1][8]

Cup

Year	Time Controls	Result	Ref
2018	30+10	1st	[63]
2019	30+5	2nd[note 1]	[64]
2019	30+5	2nd	[65]
2019	30+5	1st	[66]
2020	30+5	1st	[67]
2020	30+5	3rd	[68]
2020	30+5	1st	[69]
2021	30+5	1st	[70]
2021	30+5	1st	[71]
2022	30+3	1st	[72]
2023	30+3	2nd	[73]

Leela Chess Zero

Original author(s) Gian-Carlo Pascutto, Gary Linscott

Leela Chess Zero (abbreviated as **LCZero**, **lc0**) is a free, open-source, and deep neural network–based chess engine and volunteer computing project. Development has been spearheaded by programmer Gary Linscott, who is also a developer for the Stockfish chess engine. Leela Chess Zero was adapted from the Leela Zero Go engine,[1] which in turn was based on Google's AlphaGo Zero project.[2] One of the purposes of Leela Chess Zero was to verify the methods in the AlphaZero paper as applied to the game of chess.

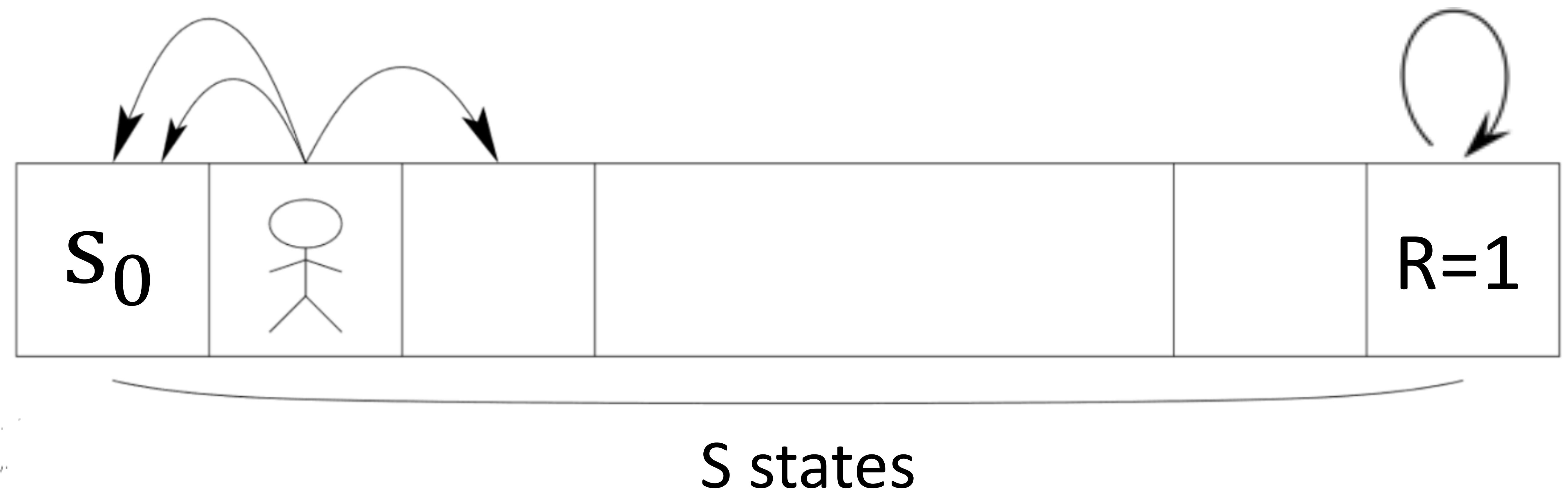
Comments:

- **Question:**
  When do we use rollout methods (MPC/AlphaZero) vs PG methods?

- **MuZero**
  - Basically AlphaZero but we don't know game rules.
  - We learn the transition function as we play.

Warmup for UCB-VI

How we do find $\pi^\star$ in an unknown MDP?

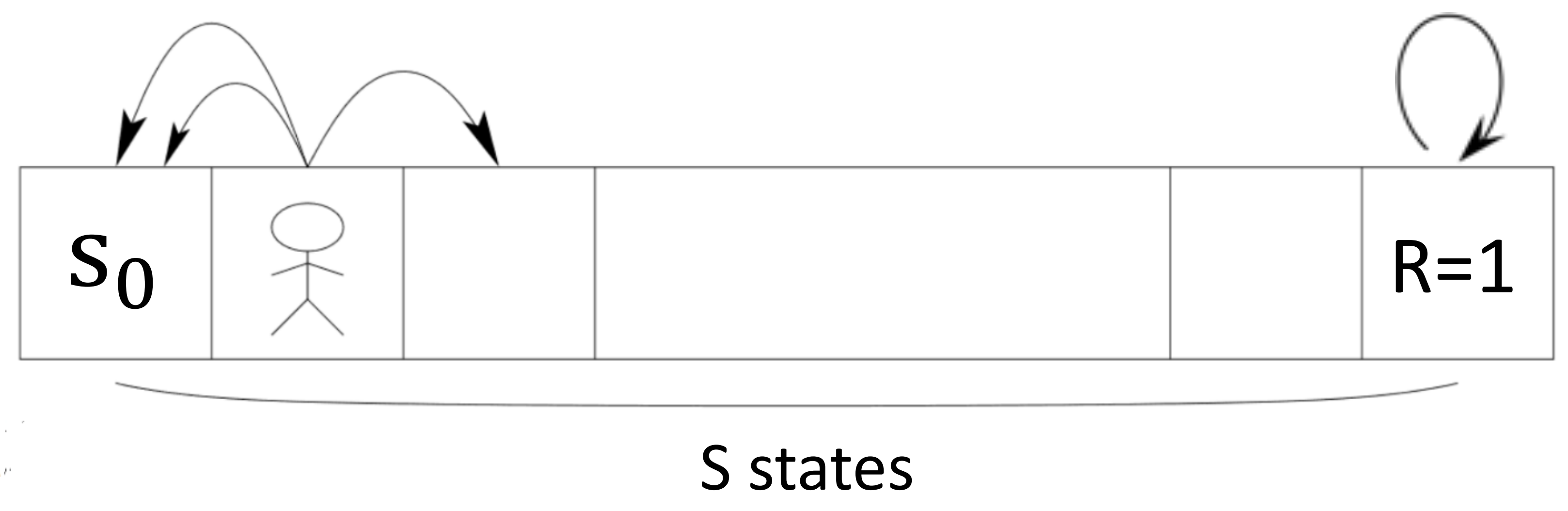


Thrun '92

- Episodic setting with an unknown MDP:
  - suppose we start at $s_0 \sim \mu$.
  - We act for H steps.
  - Then repeat.
- How do we find $\pi^\star$? near optimal
- How do get low regret?

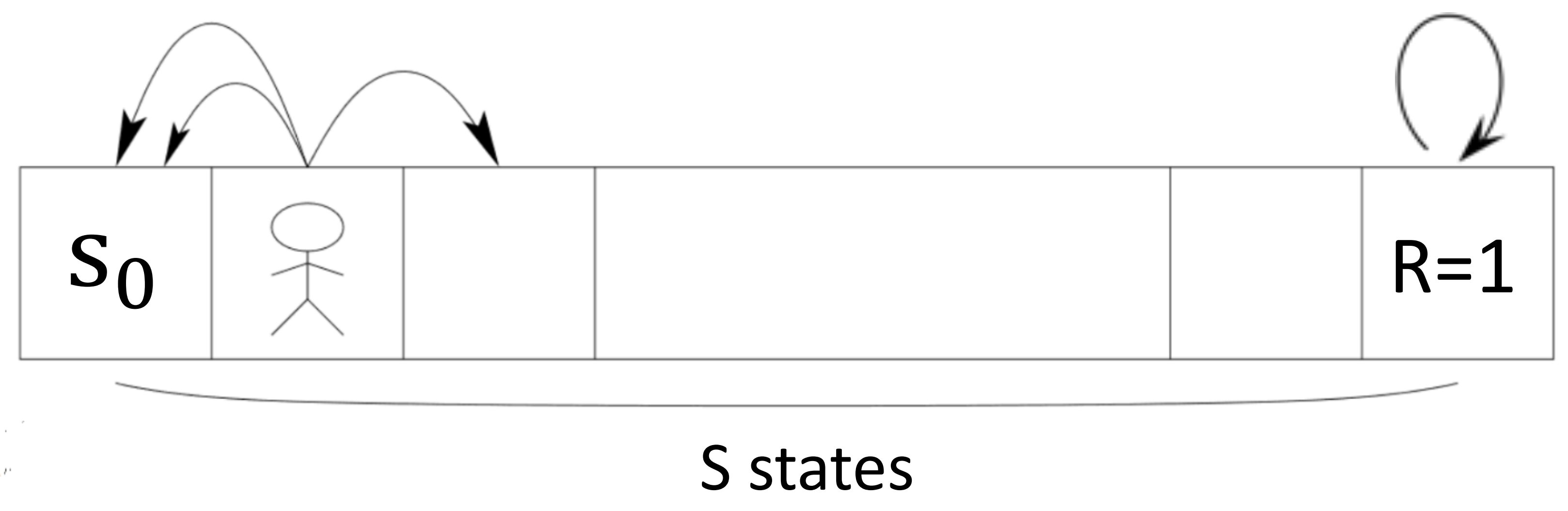
- Let's start with the setting where the MDP is deterministic.
  - So both $r(s,a)$ and $P(\cdot \mid s,a)$ are deterministic.

Algorithm: ExploreThenExploit (For Det MDPs)



Thrun '92

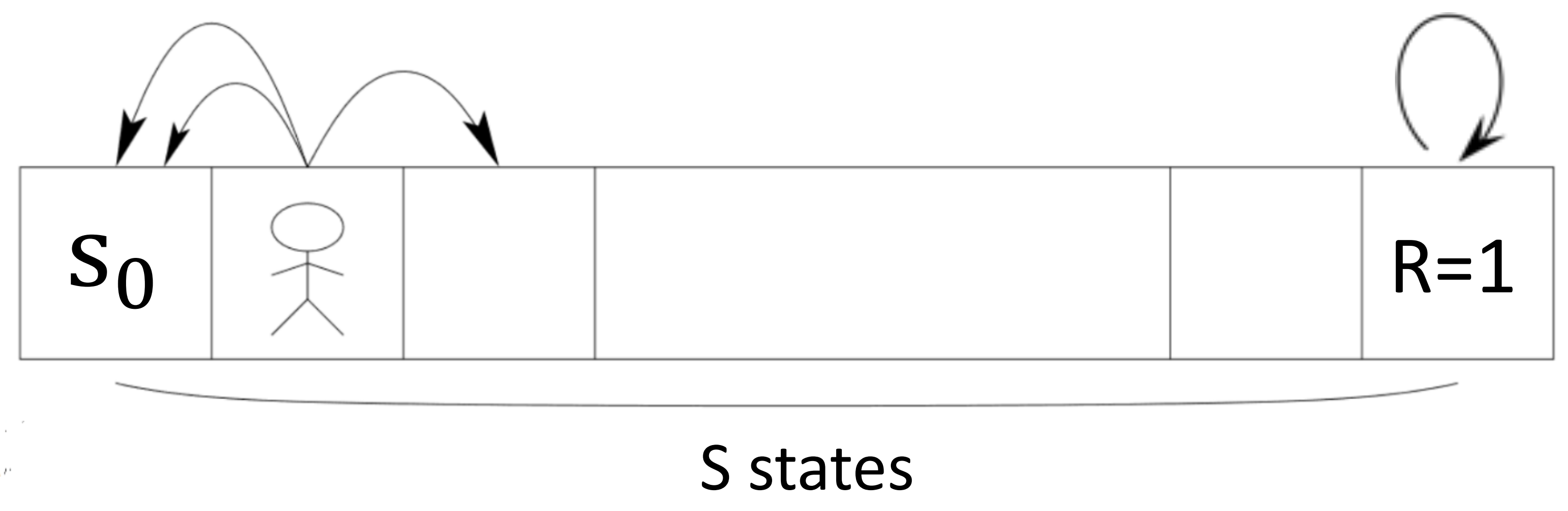
Algorithm: ExploreThenExploit (For Det MDPs)



Thrun '92

- Let's say a state-action pair (s,a) is known if both NextState(s, a) and $r(s, a)$ are known.

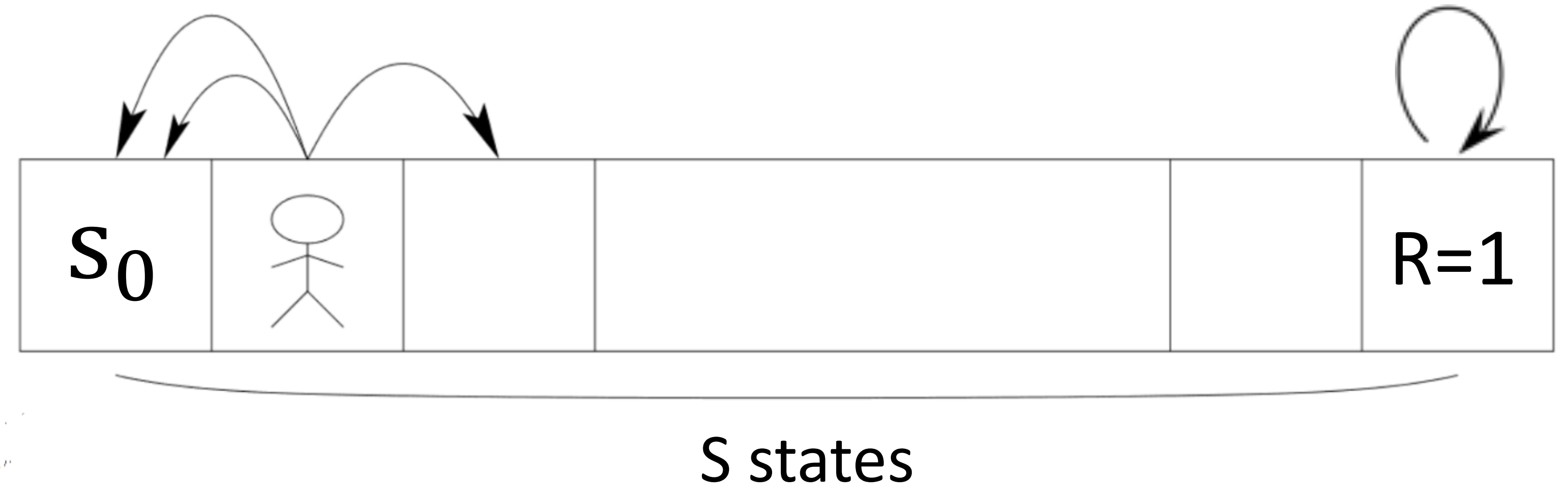
Algorithm: ExploreThenExploit (For Det MDPs)



Thrun '92

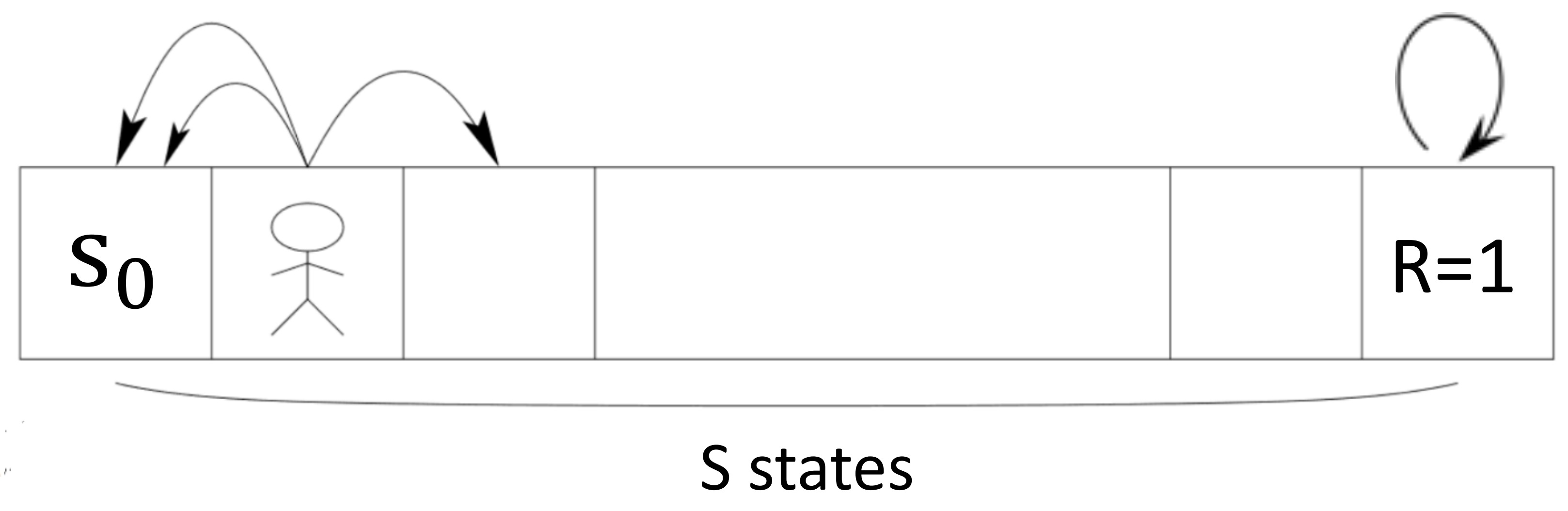
- Let's say a state-action pair (s,a) is known if both NextState(s, a) and $r(s, a)$ are known.
  - When is (s,a) known at episode N?

Algorithm: ExploreThenExploit (For Det MDPs)



- Let's say a state-action pair (s,a) is known if both NextState(s, a) and $r(s, a)$ are known.
  - When is (s,a) known at episode N?
  - Let K_N be the set of known state-action pairs at episode N.

Algorithm: ExploreThenExploit (For Det MDPs)

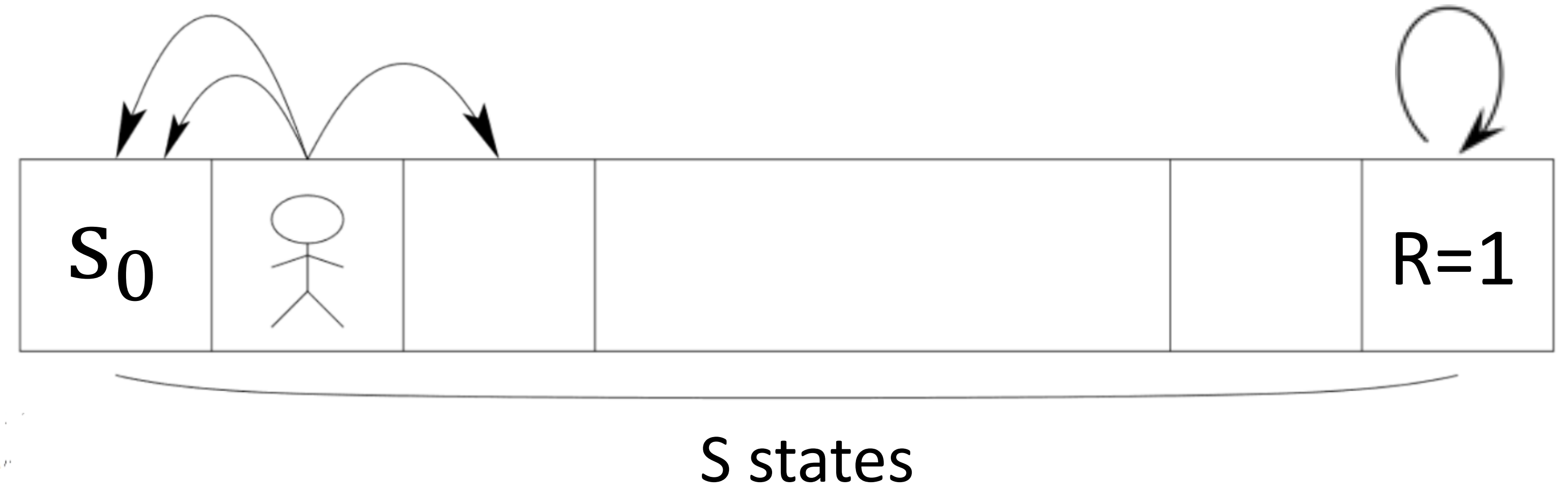


Thrun '92

- Let's say a state-action pair (s,a) is known if both NextState(s, a) and $r(s, a)$ are known.
  - When is (s,a) known at episode N?
  - Let K_N be the set of known state-action pairs at episode N.

- Init: $K_0 = \emptyset$

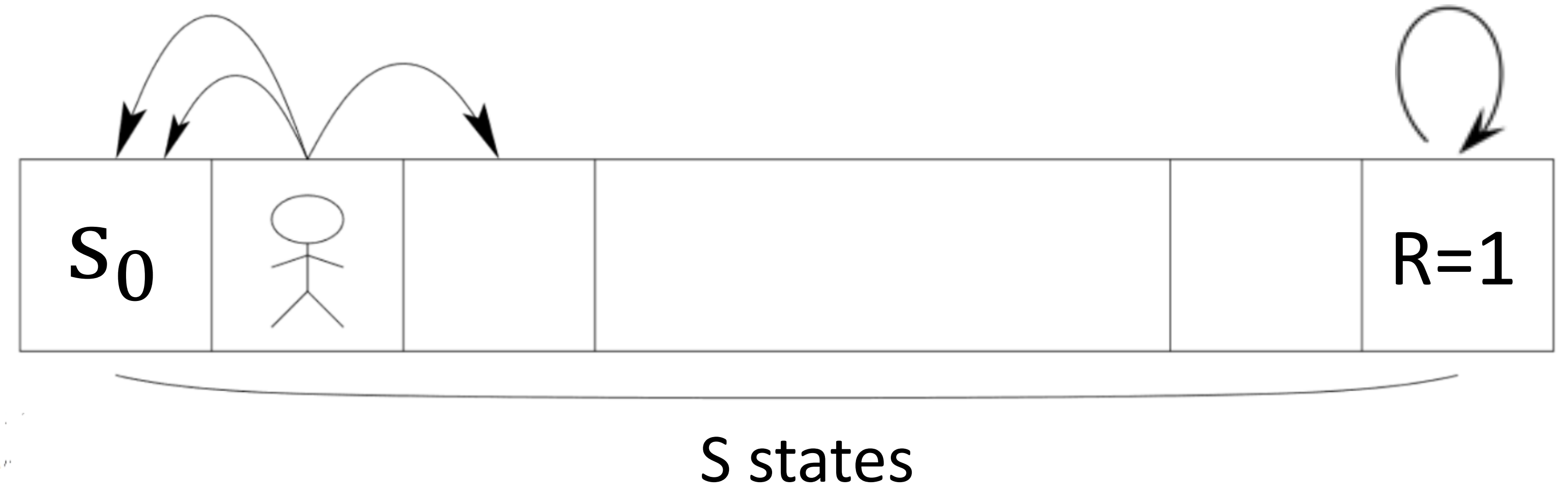
Algorithm: ExploreThenExploit (For Det MDPs)



- Let's say a state-action pair (s,a) is known if both NextState(s, a) and $r(s, a)$ are known.
  - When is (s,a) known at episode N?
  - Let K_N be the set of known state-action pairs at episode N.

- Init: $K_0 = \emptyset$
- While not terminated

Algorithm: ExploreThenExploit (For Det MDPs)

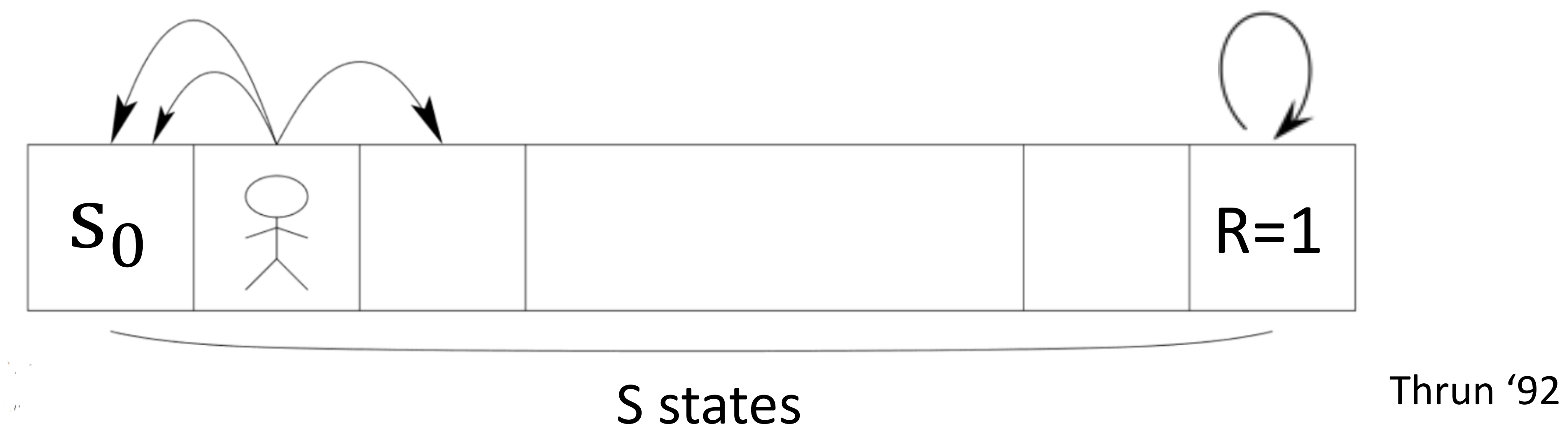


Thrun '92

- Let's say a state-action pair (s,a) is known if both NextState(s, a) and $r(s, a)$ are known.
  - When is (s,a) known at episode N?
  - Let K_N be the set of known state-action pairs at episode N.

- Init: $K_0 = \emptyset$
- While not terminated
  - If there exists $(s, a) \notin K_N$, compute the shortest path $\tilde{\pi}$ to (s, a)

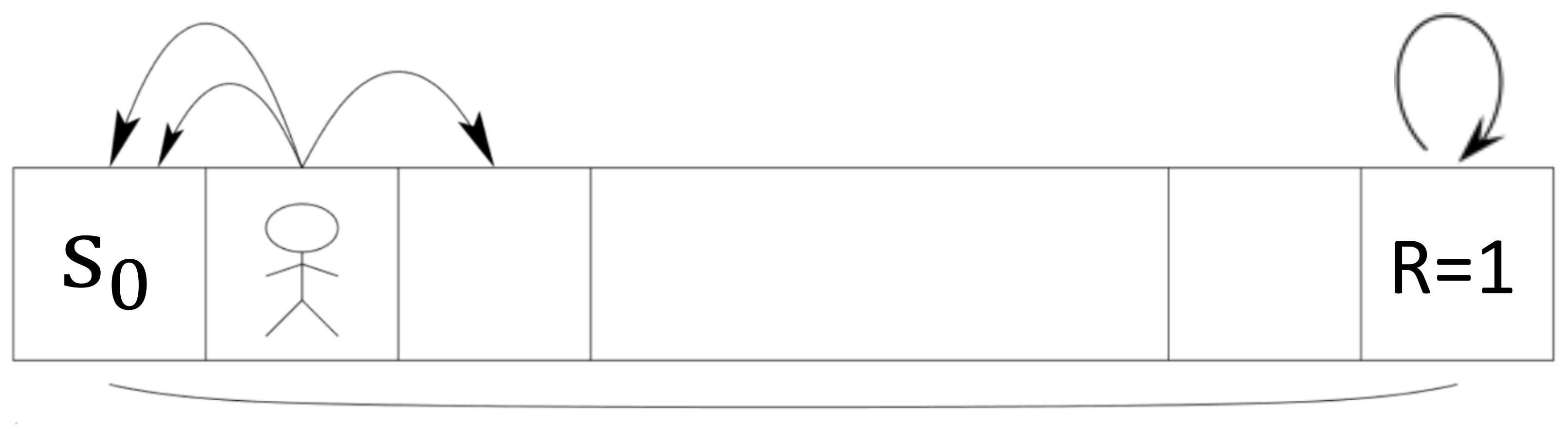
Algorithm: ExploreThenExploit (For Det MDPs)



- Let's say a state-action pair (s,a) is known if both NextState(s, a) and $r(s, a)$ are known.
  - When is (s,a) known at episode N?
  - Let K_N be the set of known state-action pairs at episode N.

- Init: $K_0 = \emptyset$
- While not terminated
  - If there exists $(s, a) \notin K_N$, compute the shortest path $\widetilde{\pi}$ to (s, a)
    - Execute $\widetilde{\pi}$ visits (s, a) and update $K_{N+1} = K_N \cup (s, a)$

Algorithm: ExploreThenExploit (For Det MDPs)

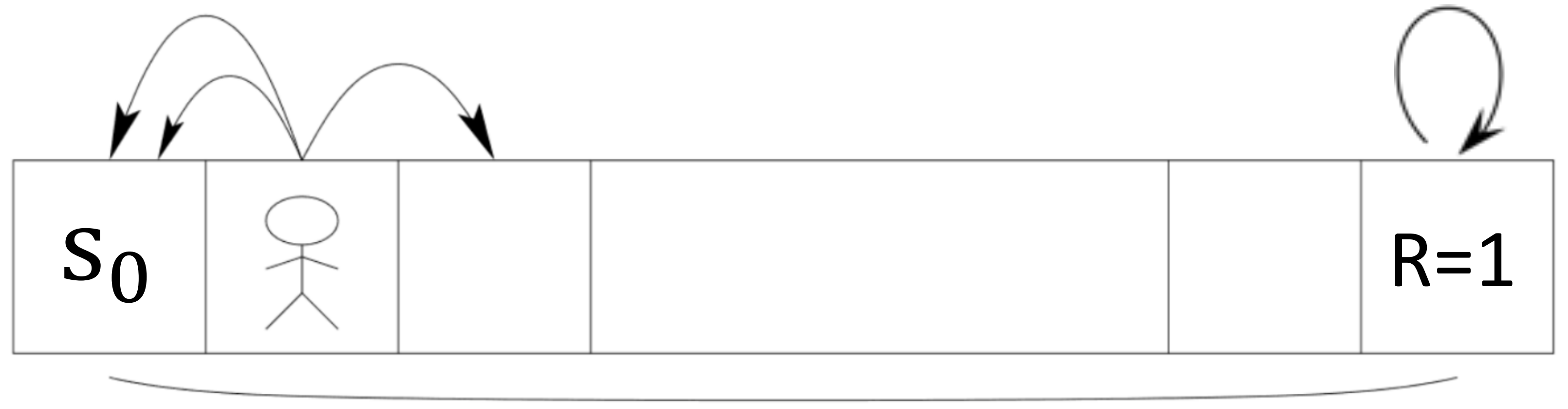


Thrun '92

- Let's say a state-action pair (s,a) is known if both NextState(s, a) and $r(s, a)$ are known.
  - When is (s,a) known at episode N?
  - Let K_N be the set of known state-action pairs at episode N.

- Init: $K_0 = \emptyset$
- While not terminated
  - If there exists $(s, a) \notin K_N$, compute the shortest path $\widetilde{\pi}$ to (s, a)
    - Execute $\widetilde{\pi}$ visits (s, a) and update $K_{N+1} = K_N \cup (s, a)$
  - Else: terminate

Algorithm: ExploreThenExploit (For Det MDPs)



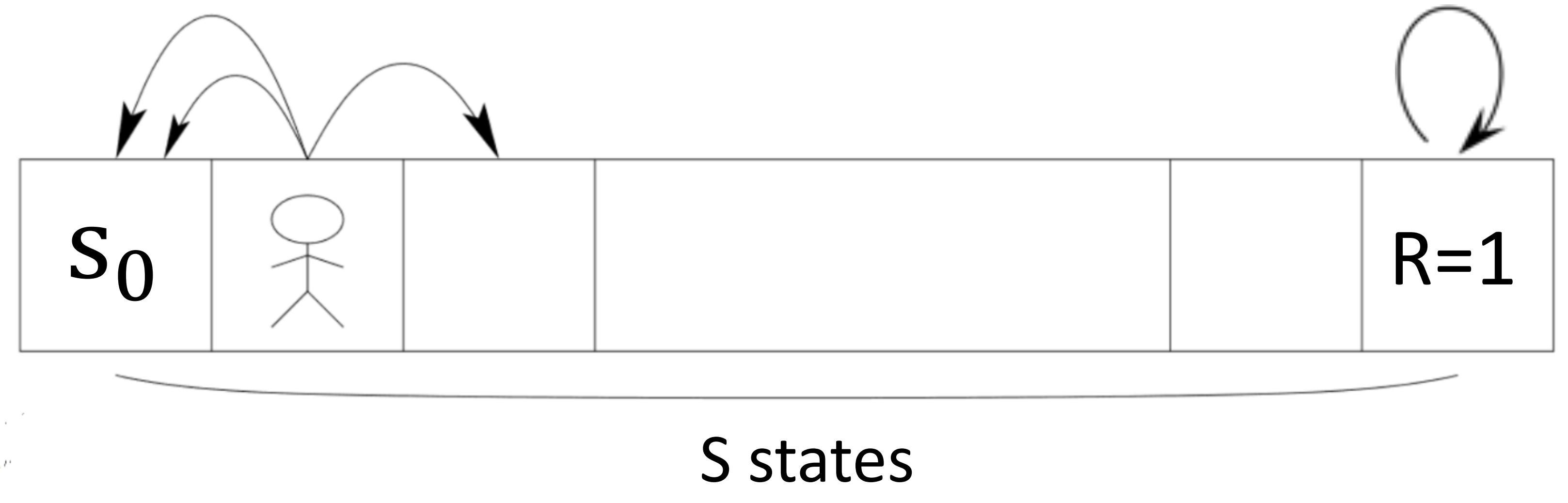
Thrun '92

- Let's say a state-action pair (s,a) is known if both NextState(s, a) and $r(s, a)$ are known.
  - When is (s,a) known at episode N?
  - Let K_N be the set of known state-action pairs at episode N.

- Init: $K_0 = \varnothing$
- While not terminated
  - If there exists $(s, a) \notin K_N$, compute the shortest path $\widetilde{\pi}$ to (s, a)
    - Execute $\widetilde{\pi}$ visits (s, a) and update $K_{N+1} = K_N \cup (s, a)$
  - Else: terminate
- Return: the optimal policy in the known MDP.

in our "known" MDP

Algorithm: ExploreThenExploit (For Det MDPs)



Thrun '92

- Let's say a state-action pair (s,a) is known if both NextState(s, a) and $r(s, a)$ are known.
  - When is (s,a) known at episode N?
  - Let K_N be the set of known state-action pairs at episode N.

- Init: $K_0 = \varnothing$
- While not terminated
  - If there exists $(s, a) \notin K_N$, compute the shortest path $\widetilde{\pi}$ to (s, a)
    - Execute $\widetilde{\pi}$ visits (s, a) and update $K_{N+1} = K_N \cup (s, a)$
  - Else: terminate
- Return: the optimal policy in the known MDP.

Theorem: Assuming $H \geq |S|$, this algorithm returns an optimal policy in most ?? trajectories.

Comments:

Comments:

- Shortest path computation:
  How do we formulate this as computing an optimal policy in some modified MDP?

Comments:

- Shortest path computation:
  How do we formulate this as computing an optimal policy in some modified MDP?

- How do we modify the algorithm for general H?

Comments:

- Shortest path computation:
  How do we formulate this as computing an optimal policy in some modified MDP?

- How do we modify the algorithm for general H?

- What is the regret of this algorithm?

$$\sum_{t=1}^{T}\left(V_0(\pi^t) - V_0(\pi^*)\right) \leq SAH$$

(Rest of) Today

- Why we don't want to treat MDPs as big bandits
- UCB-VI for tabular MDPs

Exploration in MDP: make it a bandit and do UCB?

Q: given a discrete MDP, how many unique policies we have?

Exploration in MDP: make it a bandit and do UCB?

Q: given a discrete MDP, how many unique policies we have?

$$\left(|A|^{|S|} \right)^{H}$$

Exploration in MDP: make it a bandit and do UCB?

Q: given a discrete MDP, how many unique policies we have?

$$\left(|A|^{|S|}\right)^{H}$$

So treating each policy as an "arm" and running UCB gives us regret $\tilde{O}(\sqrt{|A|^{|S|H} N})$

Exploration in MDP: make it a bandit and do UCB?

Q: given a discrete MDP, how many unique policies we have?

$$\left(|A|^{|S|}\right)^{H}$$

So treating each policy as an "arm" and running UCB gives us regret $\tilde{O}(\sqrt{|A|^{|S|H}N})$

This seems bad, so are MDPs just super hard or can we do better?

An example of MDP as bandit

$$S = \{a, b\}, \quad A = \{1,2\}, \quad H = 2$$

All state transitions happen with probability 1/2 for all actions

Reward function:
$$r(a,1) = r(b,1) = 0$$
$$r(a,2) = r(b,2) = 1$$

An example of MDP as bandit

$$S = \{a, b\}, \quad A = \{1,2\}, \quad H = 2$$

All state transitions happen with probability 1/2 for all actions

Reward function: $$\begin{aligned} r(a,1) = r(b,1) = 0 \\ r(a,2) = r(b,2) = 1 \end{aligned}$$

Suppose we have a lot of data already on a policy $\pi^{(1)}$ that always takes action 1 and a policy $\pi^{(2)}$ that always takes action 2 (note $\pi^{(2)} = \pi^\star$)

An example of MDP as bandit

$$S = \{a, b\}, \quad A = \{1,2\}, \quad H = 2$$

All state transitions happen with probability 1/2 for all actions

Reward function: $$r(a,1) = r(b,1) = 0$$ $$r(a,2) = r(b,2) = 1$$

Suppose we have a lot of data already on a policy $\pi^{(1)}$ that always takes action 1 and a policy $\pi^{(2)}$ that always takes action 2 (note $\pi^{(2)} = \pi^\star$)

What do we know about a policy $\pi^{(3)}$ which always takes action 1 in the first time step, and always takes action 2 at the second time step?

An example of MDP as bandit

$$S = \{a, b\}, \quad A = \{1,2\}, \quad H = 2$$

All state transitions happen with probability 1/2 for all actions

Reward function: $r(a,1) = r(b,1) = 0$
$r(a,2) = r(b,2) = 1$

Suppose we have a lot of data already on a policy $\pi^{(1)}$ that always takes action 1 and a policy $\pi^{(2)}$ that always takes action 2 (note $\pi^{(2)} = \pi^\star$)

What do we know about a policy $\pi^{(3)}$ which always takes action 1 in the first time step, and always takes action 2 at the second time step?

Everything: we have a lot of data on every state-action reward and transition!

An example of MDP as bandit

$$S = \{a, b\}, \quad A = \{1,2\}, \quad H = 2$$

All state transitions happen with probability 1/2 for all actions

Reward function: $$r(a,1) = r(b,1) = 0$$ $$r(a,2) = r(b,2) = 1$$

Suppose we have a lot of data already on a policy $\pi^{(1)}$ that always takes action 1 and a policy $\pi^{(2)}$ that always takes action 2 (note $\pi^{(2)} = \pi^\star$)

What do we know about a policy $\pi^{(3)}$ which always takes action 1 in the first time step, and always takes action 2 at the second time step?

Everything: we have a lot of data on every state-action reward and transition!

If we treat the MDP as a bandit, we treat $\pi^{(3)}$ as a new "arm" about which we know nothing…

An example of MDP as bandit

$$S = \{a, b\}, \quad A = \{1,2\}, \quad H = 2$$

$|A|^{|S|H} = 2^4 = 16$

All state transitions happen with probability 1/2 for all actions

Reward function: $r(a,1) = r(b,1) = 0$
$r(a,2) = r(b,2) = 1$

Suppose we have a lot of data already on a policy $\pi^{(1)}$ that always takes action 1 and a policy $\pi^{(2)}$ that always takes action 2 (note $\pi^{(2)} = \pi^\star$)

What do we know about a policy $\pi^{(3)}$ which always takes action 1 in the first time step, and always takes action 2 at the second time step?

Everything: we have a lot of data on every state-action reward and transition!

If we treat the MDP as a bandit, we treat $\pi^{(3)}$ as a new "arm" about which we know nothing…

Today

- Why we don’t want to treat MDPs as big bandits
- UCB-VI for tabular MDPs

Recall: Value Iteration (VI)

VI = DP is a backwards in time approach for computing the optimal policy:

$$\pi^\star = \{\pi_0^\star, \pi_1^\star, \ldots, \pi_{H-1}^\star\}$$

Recall: Value Iteration (VI)

VI = DP is a backwards in time approach for computing the optimal policy:

$$\pi^\star = \{\pi_0^\star, \pi_1^\star, \ldots, \pi_{H-1}^\star\}$$

1. Start at $H - 1$,

Recall: Value Iteration (VI)

VI = DP is a backwards in time approach for computing the optimal policy:

$$\pi^\star = \{\pi_0^\star, \pi_1^\star, \ldots, \pi_{H-1}^\star\}$$

1. Start at $H-1$,

$$Q_{H-1}^\star(s,a) = r(s,a)$$

Recall: Value Iteration (VI)

VI = DP is a backwards in time approach for computing the optimal policy:

$$\pi^\star = \{\pi_0^\star, \pi_1^\star, \ldots, \pi_{H-1}^\star\}$$

1. Start at $H - 1$,

$$Q_{H-1}^\star(s, a) = r(s, a) \qquad \pi_{H-1}^\star(s) = \arg\max_a Q_{H-1}^\star(s, a)$$

Recall: Value Iteration (VI)

VI = DP is a backwards in time approach for computing the optimal policy:

$$\pi^\star = \{\pi_0^\star, \pi_1^\star, \ldots, \pi_{H-1}^\star\}$$

1. Start at $H-1$,

$$Q_{H-1}^\star(s,a) = r(s,a) \qquad \pi_{H-1}^\star(s) = \arg\max_a Q_{H-1}^\star(s,a)$$

$$V_{H-1}^\star = \max_a Q_{H-1}^\star(s,a) = Q_{H-1}^\star(s, \pi_{H-1}^\star(s))$$

Recall: Value Iteration (VI)

VI = DP is a backwards in time approach for computing the optimal policy:

$$\pi^\star = \{\pi_0^\star, \pi_1^\star, \ldots, \pi_{H-1}^\star\}$$

1. Start at $H-1$,

$$Q_{H-1}^\star(s,a) = r(s,a) \qquad \pi_{H-1}^\star(s) = \arg\max_a Q_{H-1}^\star(s,a)$$

$$V_{H-1}^\star = \max_a Q_{H-1}^\star(s,a) = Q_{H-1}^\star(s, \pi_{H-1}^\star(s))$$

2. Assuming we have computed $V_{h+1}^\star$, $h \leq H-2$, i.e., assuming we know how to perform optimally starting at $h+1$, then:

Recall: Value Iteration (VI)

VI = DP is a backwards in time approach for computing the optimal policy:

$$\pi^\star = \{\pi_0^\star, \pi_1^\star, \ldots, \pi_{H-1}^\star\}$$

1. Start at $H-1$,

$$Q_{H-1}^\star(s,a) = r(s,a) \qquad \pi_{H-1}^\star(s) = \arg\max_a Q_{H-1}^\star(s,a)$$

$$V_{H-1}^\star = \max_a Q_{H-1}^\star(s,a) = Q_{H-1}^\star(s, \pi_{H-1}^\star(s))$$

2. Assuming we have computed $V_{h+1}^\star$, $h \le H-2$, i.e., assuming we know how to perform optimally starting at $h+1$, then:

$$Q_h^\star(s,a) = r(s,a) + \mathbb{E}_{s' \sim P(s,a)} V_{h+1}^\star(s')$$

Recall: Value Iteration (VI)

VI = DP is a backwards in time approach for computing the optimal policy:

$$\pi^\star = \{\pi_0^\star, \pi_1^\star, \ldots, \pi_{H-1}^\star\}$$

1. Start at $H-1$,

$$Q_{H-1}^\star(s,a) = r(s,a) \qquad \pi_{H-1}^\star(s) = \arg\max_a Q_{H-1}^\star(s,a)$$

$$V_{H-1}^\star = \max_a Q_{H-1}^\star(s,a) = Q_{H-1}^\star(s, \pi_{H-1}^\star(s))$$

2. Assuming we have computed $V_{h+1}^\star$, $h \le H-2$, i.e., assuming we know how to perform optimally starting at $h+1$, then:

$$Q_h^\star(s,a) = r(s,a) + \mathbb{E}_{s' \sim P(s,a)} V_{h+1}^\star(s')$$

$$\pi_h^\star(s) = \arg\max_a Q_h^\star(s,a),$$

Recall: Value Iteration (VI)

VI = DP is a backwards in time approach for computing the optimal policy:

$$\pi^\star = \{\pi_0^\star, \pi_1^\star, \ldots, \pi_{H-1}^\star\}$$

1. Start at $H-1$,

$$Q_{H-1}^\star(s,a) = r(s,a) \qquad \pi_{H-1}^\star(s) = \arg\max_a Q_{H-1}^\star(s,a)$$

$$V_{H-1}^\star = \max_a Q_{H-1}^\star(s,a) = Q_{H-1}^\star(s, \pi_{H-1}^\star(s))$$

2. Assuming we have computed $V_{h+1}^\star$, $h \leq H-2$, i.e., assuming we know how to perform optimally starting at $h+1$, then:

$$Q_h^\star(s,a) = r(s,a) + \mathbb{E}_{s' \sim P(s,a)} V_{h+1}^\star(s')$$

$$\pi_h^\star(s) = \arg\max_a Q_h^\star(s,a), \qquad V_h^\star = \max_a Q_h^\star(s,a)$$

Recall: UCB

For $t = 0,\ldots,T-1$:

Choose the arm with the highest upper confidence bound, i.e.,

$$a_t = \arg\max_{k\in\{1,\ldots,K\}} \hat{\mu}_t^{(k)} + \sqrt{\ln(2TK/\delta)/2N_t^{(k)}}$$

Recall: UCB

For $t = 0,\ldots,T-1$:

Choose the arm with the highest upper confidence bound, i.e.,

$$a_t = \arg\max_{k\in\{1,\ldots,K\}} \hat{\mu}_t^{(k)} + \sqrt{\ln(2TK/\delta)/2N_t^{(k)}}$$

<u>High-level summary</u>: estimate action quality, add exploration bonus, then argmax

UCBVI: Tabular optimism in the face of uncertainty

Assume reward function $r_h(s, a)$ known

Inside iteration n :

UCBVI: Tabular optimism in the face of uncertainty

Assume reward function $r_h(s, a)$ known

Inside iteration n :

Use all previous data to estimate transitions $\widehat{P}_1^n, \ldots, \widehat{P}_{H-1}^n$

UCBVI: Tabular optimism in the face of uncertainty

Assume reward function $r_h(s, a)$ known

Inside iteration n :

Use all previous data to estimate transitions $\widehat{P}_1^n, \ldots, \widehat{P}_{H-1}^n$

Design reward bonus $b_h^n(s, a), \forall s, a, h$

UCBVI: Tabular optimism in the face of uncertainty

Assume reward function $r_h(s, a)$ known

Inside iteration n :

Use all previous data to estimate transitions $\widehat{P}_1^n, \ldots, \widehat{P}_{H-1}^n$

Design reward bonus $b_h^n(s, a), \forall s, a, h$

Optimistic planning with learned model: $\pi^n = \mathrm{VI}\left(\{\widehat{P}_h^n, r_h + b_h^n\}_{h=1}^{H-1}\right)$

UCBVI: Tabular optimism in the face of uncertainty

Assume reward function $r_h(s, a)$ known

Inside iteration n :

Use all previous data to estimate transitions $\widehat{P}^n_1, \ldots, \widehat{P}^n_{H-1}$

Design reward bonus $b^n_h(s, a), \forall s, a, h$

Optimistic planning with learned model: $\pi^n = \text{VI}\left(\{ \widehat{P}^n_h, r_h + b^n_h \}_{h=1}^{H-1} \right)$

Collect a new trajectory by executing π^n in the true system $\{P_h\}_{h=0}^{H-1}$ starting from s_0

Model Estimation

Let us consider the **very beginning** of episode n:

$$\mathcal{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h$$

Model Estimation

Let us consider the **very beginning** of episode n:

$$\mathscr{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h$$

Let's also maintain some statistics using these datasets:

Model Estimation

Let us consider the **very beginning** of episode n:

$$\mathcal{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h$$

Let's also maintain some statistics using these datasets:

$$N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s,a,h, \quad N_h^n(s,a,s') = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i, s_{h+1}^i) = (s,a,s')\}, \forall s,a,h$$

Model Estimation

Let us consider the **very beginning** of episode n:

$$\mathcal{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h$$

Let's also maintain some statistics using these datasets:

$$N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s,a,h, \quad N_h^n(s,a,s') = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i, s_{h+1}^i) = (s,a,s')\}, \forall s,a,h$$

Estimate model $\widehat{P}_h^n(s' \mid s,a), \forall s,a,s',h$:

$$\widehat{P}_h^n(s' \mid s,a) = \frac{N_h^n(s,a,s')}{N_h^n(s,a)}$$

Reward Bonus Design and Value Iteration

Let us consider the very beginning of episode n:

$$\mathcal{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h, \quad N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s, a, h,$$

Reward Bonus Design and Value Iteration

Let us consider the very beginning of episode n:

$$\mathcal{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h, \quad N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s, a, h,$$

$$b_h^n(s,a) = cH\sqrt{\frac{\log\left(SAHN/\delta\right)}{N_h^n(s,a)}}$$

Reward Bonus Design and Value Iteration

Let us consider the very beginning of episode n:

$$\mathcal{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h, \quad N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s, a, h,$$

$$b_h^n(s,a) = cH\sqrt{\frac{\log\left(SAHN/\delta\right)}{N_h^n(s,a)}}$$

Encourage to explore new state-actions

Reward Bonus Design and Value Iteration

Let us consider the very beginning of episode n:

$$\mathcal{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h, \quad N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s, a, h,$$

$$b_h^n(s,a) = cH\sqrt{\frac{\log\left(SAHN/\delta\right)}{N_h^n(s,a)}}$$

Encourage to explore new state-actions

Value Iteration (aka DP) at episode n using $\{\widehat{P}_h^n\}_h$ **and** $\{r_h + b_h^n\}_h$

Reward Bonus Design and Value Iteration

Let us consider the very beginning of episode n:

$$\mathscr{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h, \quad N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s, a, h,$$

$$b_h^n(s,a) = cH\sqrt{\frac{\log\left(SAHN/\delta\right)}{N_h^n(s,a)}}$$

Encourage to explore new state-actions

Value Iteration (aka DP) at episode n using $\{\widehat{P}_h^n\}_h$ **and** $\{r_h + b_h^n\}_h$

$$\widehat{V}_H^n(s) = 0, \forall s$$

Reward Bonus Design and Value Iteration

Let us consider the very beginning of episode n:

$$\mathcal{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h, \quad N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s, a, h,$$

$$b_h^n(s,a) = cH\sqrt{\frac{\log(SAHN/\delta)}{N_h^n(s,a)}}$$

Encourage to explore new state-actions

Value Iteration (aka DP) at episode n using $\{\widehat{P}_h^n\}_h$ **and** $\{r_h + b_h^n\}_h$

$$\widehat{V}_H^n(s) = 0, \forall s \qquad \widehat{Q}_h^n(s,a) = \min\left\{ r_h(s,a) + b_h^n(s,a) + \widehat{P}_h^n(\cdot \mid s,a) \cdot \widehat{V}_{h+1}^n, \quad H \right\}, \forall s, a$$

Reward Bonus Design and Value Iteration

Let us consider the very beginning of episode n:

$$\mathscr{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h, \quad N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s, a, h,$$

$$b_h^n(s,a) = cH\sqrt{\frac{\log(SAHN/\delta)}{N_h^n(s,a)}}$$

Encourage to explore new state-actions

Value Iteration (aka DP) at episode n using $\{\widehat{P}_h^n\}_h$ **and** $\{r_h + b_h^n\}_h$

$$\widehat{V}_H^n(s) = 0, \forall s \qquad \widehat{Q}_h^n(s,a) = \min\left\{r_h(s,a) + b_h^n(s,a) + \widehat{P}_h^n(\cdot\,|\,s,a)\cdot\widehat{V}_{h+1}^n, \quad H\right\}, \forall s, a$$

$$\widehat{V}_h^n(s) = \max_a \widehat{Q}_h^n(s,a), \quad \pi_h^n(s) = \arg\max_a \widehat{Q}_h^n(s,a), \forall s$$

Reward Bonus Design and Value Iteration

Let us consider the very beginning of episode n:

$$\mathcal{D}_h^n = \{s_h^i, a_h^i, s_{h+1}^i\}_{i=1}^{n-1}, \forall h, \quad N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s, a, h,$$

$$b_h^n(s,a) = cH\sqrt{\frac{\log(SAHN/\delta)}{N_h^n(s,a)}}$$

Encourage to explore new state-actions

Value Iteration (aka DP) at episode n using $\{\widehat{P}_h^n\}_h$ **and** $\{r_h + b_h^n\}_h$

$$\widehat{V}_H^n(s) = 0, \forall s \qquad \widehat{Q}_h^n(s,a) = \min\left\{r_h(s,a) + b_h^n(s,a) + \widehat{P}_h^n(\cdot \mid s,a) \cdot \widehat{V}_{h+1}^n, \quad H\right\}, \forall s, a$$

$$\widehat{V}_h^n(s) = \max_a \widehat{Q}_h^n(s,a), \quad \pi_h^n(s) = \arg\max_a \widehat{Q}_h^n(s,a), \forall s \qquad \left\|\widehat{V}_h^n\right\|_\infty \le H, \forall h, n$$

UCBVI: Put All Together

For $n = 1 \to N$:

1. Set $N_h^n(s,a) = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i) = (s,a)\}, \forall s, a, h$

2. Set $N_h^n(s,a,s') = \sum_{i=1}^{n-1} \mathbf{1}\{(s_h^i, a_h^i, s_{h+1}^i) = (s,a,s')\}, \forall s, a, a', h$

3. Estimate $\widehat{P}^n$: $\widehat{P}_h^n(s'|s,a) = \dfrac{N_h^n(s,a,s')}{N_h^n(s,a)}, \forall s, a, s', h$

4. Plan: $\pi^n = VI\left(\{\widehat{P}_h^n, r_h + b_h^n\}_h\right)$, with $b_h^n(s,a) = cH\sqrt{\dfrac{\log(SAHN/\delta)}{N_h^n(s,a)}}$

5. Execute π^n : $\{s_0^n, a_0^n, r_0^n, \ldots, s_{H-1}^n, a_{H-1}^n, r_{H-1}^n, s_H^n\}$

High-level Idea: Exploration Exploitation Tradeoff

Upper bound per-episode regret: $V_0^\star(s_0) - V_0^{\pi^n}(s_0) \leq \widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$

High-level Idea: Exploration Exploitation Tradeoff

Upper bound per-episode regret: $V_0^\star(s_0) - V_0^{\pi^n}(s_0) \leq \widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$

1. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is small?

High-level Idea: Exploration Exploitation Tradeoff

Upper bound per-episode regret: $V_0^\star(s_0) - V_0^{\pi^n}(s_0) \leq \widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$

1. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is small?

Then π^n is close to $\pi^\star$, i.e., we are doing exploitation

High-level Idea: Exploration Exploitation Tradeoff

Upper bound per-episode regret: $V_0^\star(s_0) - V_0^{\pi^n}(s_0) \le \widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$

1. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is small?

Then π^n is close to $\pi^\star$, i.e., we are doing exploitation

2. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is large?

High-level Idea: Exploration Exploitation Tradeoff

Upper bound per-episode regret: $V_0^\star(s_0) - V_0^{\pi^n}(s_0) \leq \widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$

1. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is small?

Then π^n is close to $\pi^\star$, i.e., we are doing exploitation

2. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is large?

$$\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0) \leq \sum_{h=0}^{H-1} \mathbb{E}_{s,a\sim d_h^{\pi^n}} \left[b_h^n(s,a) + (\widehat{P}_h^n(\cdot \mid s,a) - P_h(\cdot \mid s,a)) \cdot \widehat{V}_{h+1}^n \right]$$ must be large

High-level Idea: Exploration Exploitation Tradeoff

Upper bound per-episode regret: $V_0^\star(s_0) - V_0^{\pi^n}(s_0) \leq \widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$

1. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is small?

Then π^n is close to $\pi^\star$, i.e., we are doing exploitation

2. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is large?

Not obvious

$\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0) \leq \sum_{h=0}^{H-1} \mathbb{E}_{s,a\sim d_h^{\pi^n}} \left[b_h^n(s,a) + (\widehat{P}_h^n(\cdot \mid s,a) - P_h(\cdot \mid s,a)) \cdot \widehat{V}_{h+1}^n \right]$ must be large

High-level Idea: Exploration Exploitation Tradeoff

Upper bound per-episode regret: $V_0^\star(s_0) - V_0^{\pi^n}(s_0) \leq \widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$

1. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is small?

Then π^n is close to $\pi^\star$, i.e., we are doing exploitation

2. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is large?

Not obvious

$$\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0) \leq \sum_{h=0}^{H-1} \mathbb{E}_{s,a\sim d_h^{\pi^n}} \left[b_h^n(s,a) + (\widehat{P}_h^n(\cdot \mid s,a) - P_h(\cdot \mid s,a)) \cdot \widehat{V}_{h+1}^n \right]$$ must be large

We collect data at steps where bonus is large or model is wrong, i.e., exploration

High-level Idea: Exploration Exploitation Tradeoff

Upper bound per-episode regret: $V_0^\star(s_0) - V_0^{\pi^n}(s_0) \leq \widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$

1. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is small?

Then π^n is close to $\pi^\star$, i.e., we are doing exploitation

2. What if $\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0)$ is large?

Not obvious

$$\widehat{V}_0^n(s_0) - V_0^{\pi^n}(s_0) \leq \sum_{h=0}^{H-1} \mathbb{E}_{s,a\sim d_h^{\pi^n}}\left[b_h^n(s,a) + (\widehat{P}_h^n(\cdot|s,a) - P_h(\cdot|s,a)) \cdot \widehat{V}_{h+1}^n\right]$$ must be large

We collect data at steps where bonus is large or model is wrong, i.e., exploration

$$\mathbb{E}\left[\text{Regret}_N\right] := \mathbb{E}\left[\sum_{n=1}^{N}\left(V^\star - V^{\pi^n}\right)\right] \leq \widetilde{O}\left(H^2\sqrt{SAN}\right)$$

Today

- ✓ Why we don’t want to treat MDPs as big bandits
- ✓ UCB-VI for tabular MDPs

Summary:

UCBVI algorithm applies UCB idea to MDPs to achieve exploration/exploitation trade-off

Attendance:
bit.ly/3RcTC9T

UCB-VI

Feedback:
bit.ly/3RHtlxy