CS 4700: Foundations of Artificial Intelligence

Fall 2019

Prof. Haym Hirsh

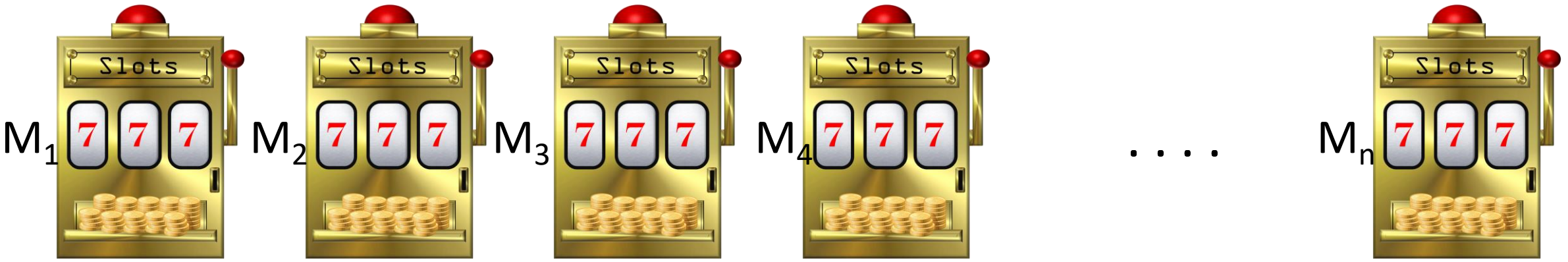
Lecture 12

October 28, 2019

Multi-Armed Bandit

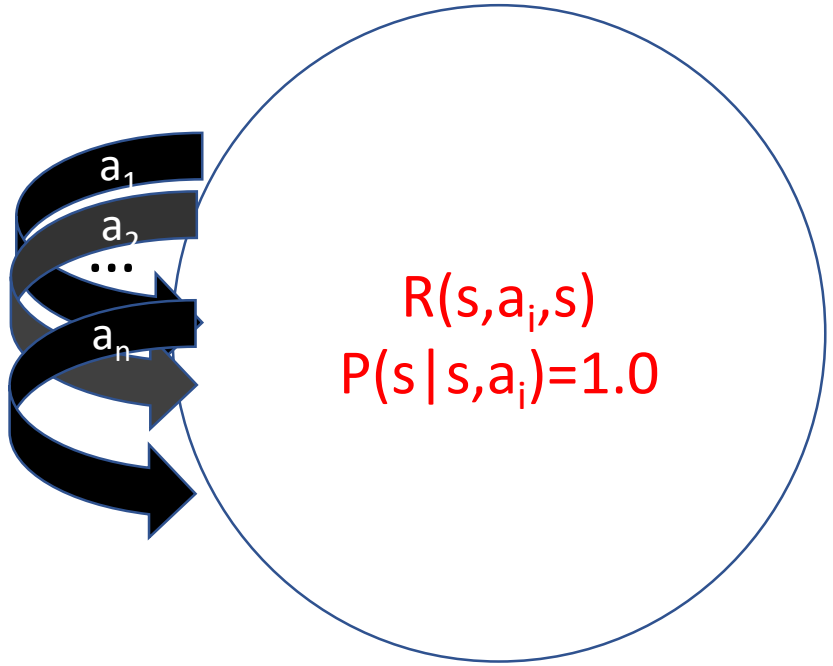
What strategy do I use to pick a sequence of a_i?

a_1 a_2 a_3 a_4 a_n



R_1 R_2 R_3 R_4 R_n

View Multi-Armed Bandit as a Single-State MDP



Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Algorithm:

Pull each arm once

For i ← 1 to n { $Sum_i \leftarrow R(arm_i)$; $N_i \leftarrow 1$ }; $N \leftarrow n$ /* Initialization */

Loop Forever

best ← $\underset{1 \le i \le n}{\text{argmax}} \left[\frac{Sum_i}{N_i} + \frac{g(N)}{\sqrt{N_i}} \right]$

$g(N) = \sqrt{2 \log (1 + N \log^2 N)}$

$g(N) = c \sqrt{\ln(N)}$

pull arm a_{best} and get reward r

$Sum_{best} \leftarrow Sum_{best} + r$; $N_{best} \leftarrow N_{best} + 1$; $N \leftarrow N+1$

Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Algorithm:

Pull each arm once

For i ← 1 to n { $\text{Sum}_i \leftarrow R(\text{arm}_i)$; $N_i \leftarrow 1$ }; $N \leftarrow n$ /* Initialization */

Loop Forever

best ← $\underset{1 \le i \le n}{\text{argmax}} \left[\frac{\text{Sum}_i}{N_i} + \sqrt{\frac{2 \ln(N)}{N_i}} \right]$

$g(N) = \sqrt{2 \log(1 + N \log^2 N)}$

$g(N) = c\sqrt{\ln(N)}$

$[c = \sqrt{2}]$

pull arm a_{best} and get reward r

$\text{Sum}_{best} \leftarrow \text{Sum}_{best} + r$; $N_{best} \leftarrow N_{best} + 1$; $N \leftarrow N+1$

Multi-Armed Bandits:
Upper Confidence Bound (UCB) Heuristic

Why $g(N) = \sqrt{2\ \log\left(1 + N \log^2 N\right)}$?

Why $g(N) = c\ \sqrt{\ln(N)}$?

Multi-Armed Bandits:
Upper Confidence Bound (UCB) Heuristic

Why g(N) = $\sqrt{2\ \log\left(1 + \mathrm{N}\ \log^2 \mathrm{N}\right)}$?

Why g(N) = c $\sqrt{\ln(\mathrm{N})}$?

Average reward: $\frac{\sum_{i=1}^{\mathrm{n}} \mathrm{Sum}_i}{\mathrm{N}}$

Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Why g(N) = $\sqrt{2\ \log\left(1 + \mathrm{N}\ \log^2 \mathrm{N}\right)}$?

Why g(N) = c $\sqrt{\ln(\mathrm{N})}$?

Average reward: $\frac{\sum_{i=1}^{\mathrm{n}} \mathrm{Sum}_i}{\mathrm{N}}$

Average reward for a policy π: $\mu_{\mathrm{N}}^{\pi} = \mathrm{E}_{\pi}\left[\frac{\sum_{i=1}^{\mathrm{n}} \mathrm{Sum}_i}{\mathrm{N}}\right]$

Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Why g(N) = $\sqrt{2\ \log\left(1 + \mathrm{N}\ \log^2 \mathrm{N}\right)}$?

Why g(N) = c $\sqrt{\ln(\mathrm{N})}$?

Average reward: $\frac{\sum_{i=1}^{\mathrm{n}} \mathrm{Sum}_i}{\mathrm{N}}$

Average reward for a policy π: $\mu_{\mathrm{N}}^{\pi} = \mathrm{E}_{\pi}\left[\frac{\sum_{i=1}^{\mathrm{n}} \mathrm{Sum}_i}{\mathrm{N}}\right]$

Average expected reward for always picking optimal arm: μ^{best}

Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Why g(N) = $\sqrt{2\ \log\left(1 + \mathrm{N}\ \log^2 \mathrm{N}\right)}$?

Why g(N) = c $\sqrt{\ln(\mathrm{N})}$?

Average reward: $\frac{\sum_{i=1}^{\mathrm{n}} \mathrm{Sum}_i}{\mathrm{N}}$

Average reward for a policy π: $\mu_{\mathrm{N}}^{\pi} = \mathrm{E}_{\pi}\left[\frac{\sum_{i=1}^{\mathrm{n}} \mathrm{Sum}_i}{\mathrm{N}}\right]$

Average expected reward for always picking optimal arm: μ^{best}

Regret for a policy: $\mathrm{regret}_{\mathrm{N}}^{\pi} = \mu^{\mathrm{best}} - \mu_{\mathrm{N}}^{\pi}$

How much exploration costs you

Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Why $g(N) = \sqrt{2\ \log\left(1 + N \log^2 N\right)}$?

Why $g(N) = c\sqrt{\ln(N)}$?

Known Result:

$\text{Regret}_N^{\pi} = \Omega(\log(N))$

Your expected regret will grow at least logarithmically with N

Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Why $g(N) = \sqrt{2\ \log\left(1 + N \log^2 N\right)}$?

Why $g(N) = c\sqrt{\ln(N)}$?

Known Result:

$\text{Regret}_N^{\pi} = \Omega(\log(N))$

$\text{Regret}_N^{UCB(g(N))} = O(\log(N))$

Your expected regret will grow at least logarithmically with N

UCB with these g(N) functions have regret that grows at worst logarithmically with N

Monte Carlo Tree Search (MCTS)
(Section 5.4)

Monte Carlo Tree Search (MCTS)
(Section 5.4)

Application of multi-armed bandits

Timeline of Key Ideas in Game Tree Search

1948	Alan Turing	Look ahead and use an evaluation function
1950	Claude Shannon	Game tree search
1956	John McCarthy	Alpha-beta pruning
1959	Arthur Samuel	Learn evaluation function (Reinforcement learning)

…

1997: Deep Blue defeats Gary Kasparov (3½–2½)

Game tree search with alpha-beta pruning
plus lots of enhancements

1997: Deep Blue defeats Gary Kasparov (3½–2½)

(Not “deep” as in “deep learning”)

(Deep as in its ancestor, Deep Thought)

Game tree search with alpha-beta pruning
plus lots of enhancements

Go?

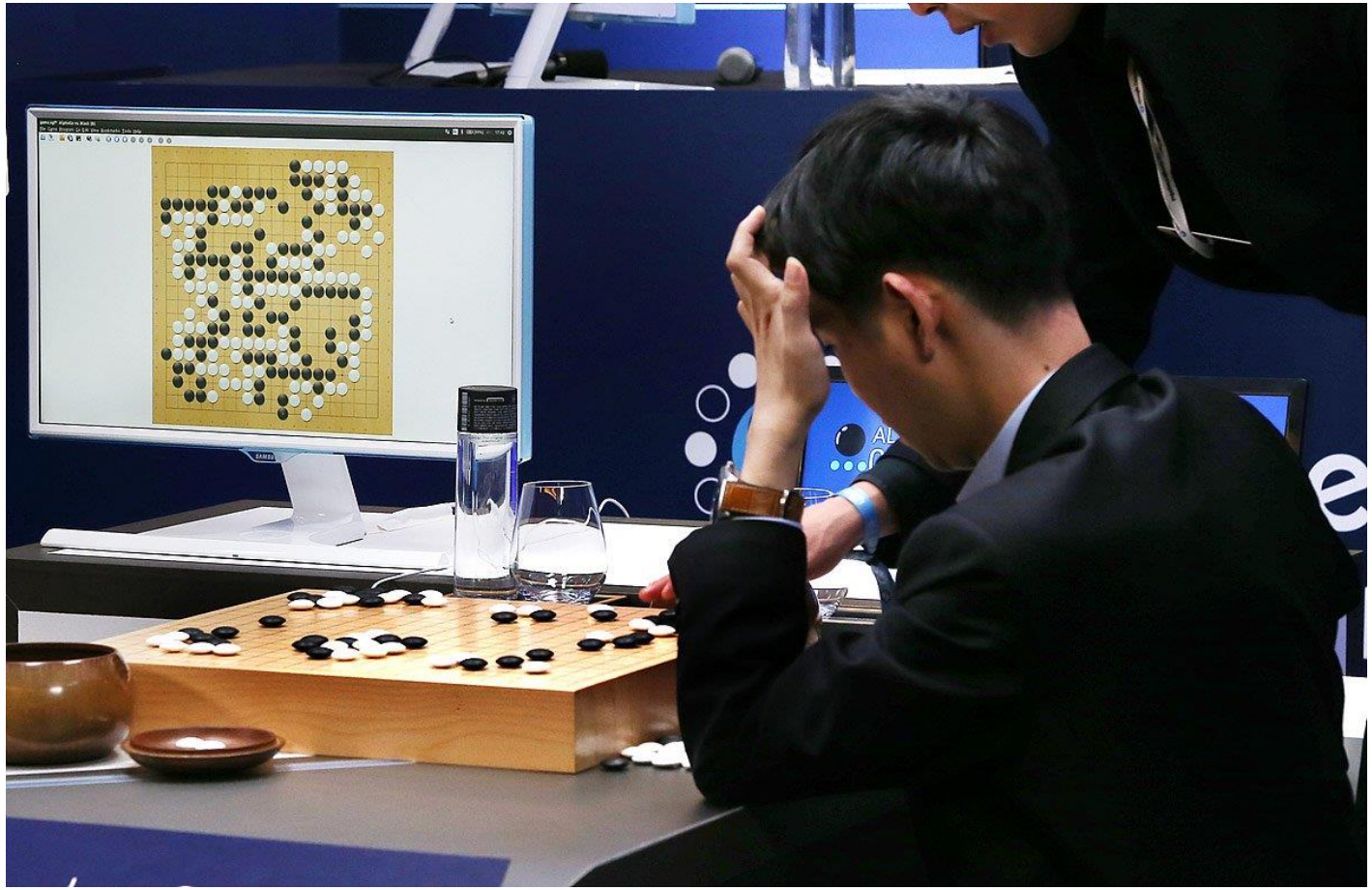
Branching factor is in the 100s

Evaluation function is difficult because payoff may be very far away

Need new ideas

1983	Bruce Ballard	Lookahead for probabilistic moves
1987	Bruce Abramson	Evaluation by expected outcome (repeated simulation)
1992	Gerald Tesauro	TD-Gammon (reinforcement learning, self-play)
1992	Bernd Brugmann	Monte Carlo Go (simulated annealing)
1999	U of Alberta	Simulation in Poker
1999	Matt Ginsberg	Simulation in Bridge
2002	Brian Sheppard	Simulation in Scrabble
2006	Levente Kocsis and Csaba Szepesvár	Multi-armed bandits for Monte-Carlo tree search

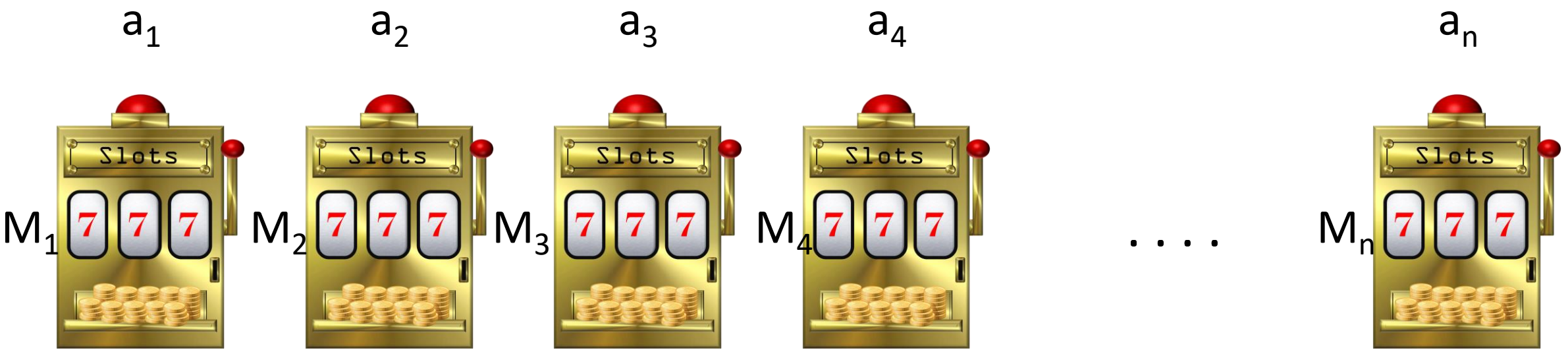
2016: AlphaGo defeats Lee Sedol (4-1)



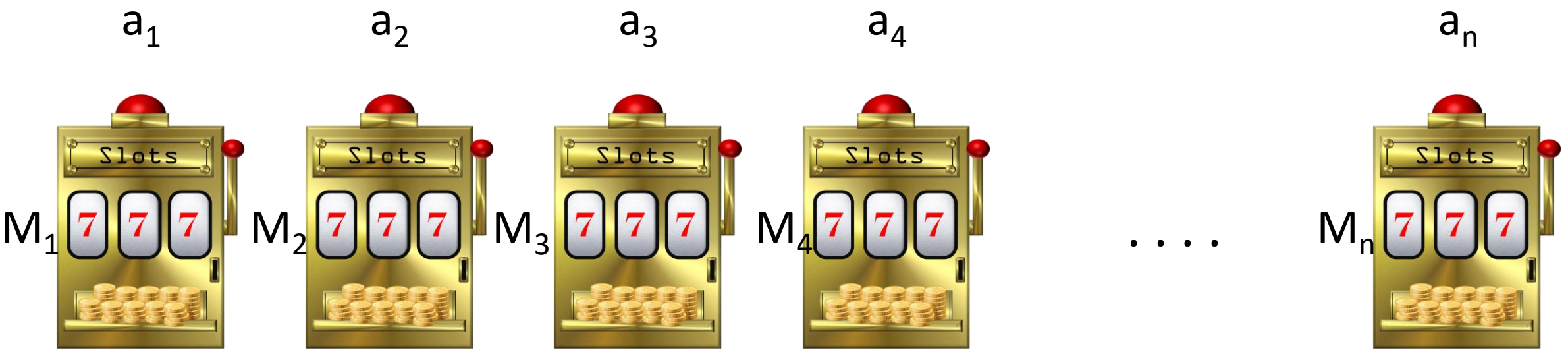
Key ideas of Monte Carlo Tree Search:

1. View move selection as a multi-armed bandit problem

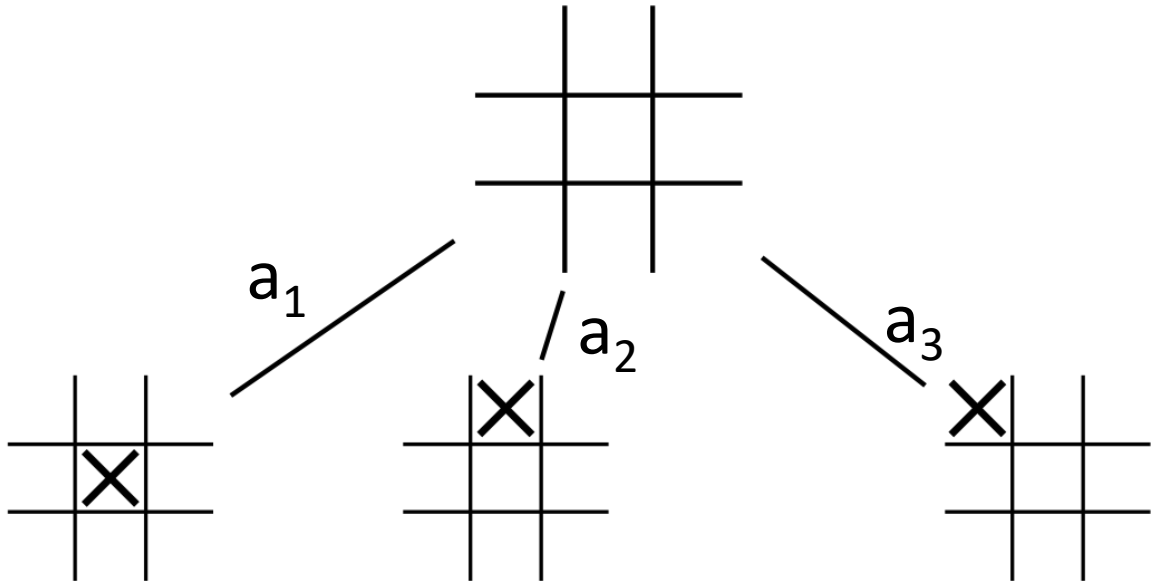
Multi-Armed Bandit



Multi-Armed Bandit for Game Tree Search



Multi-Armed Bandit for Game Tree Search

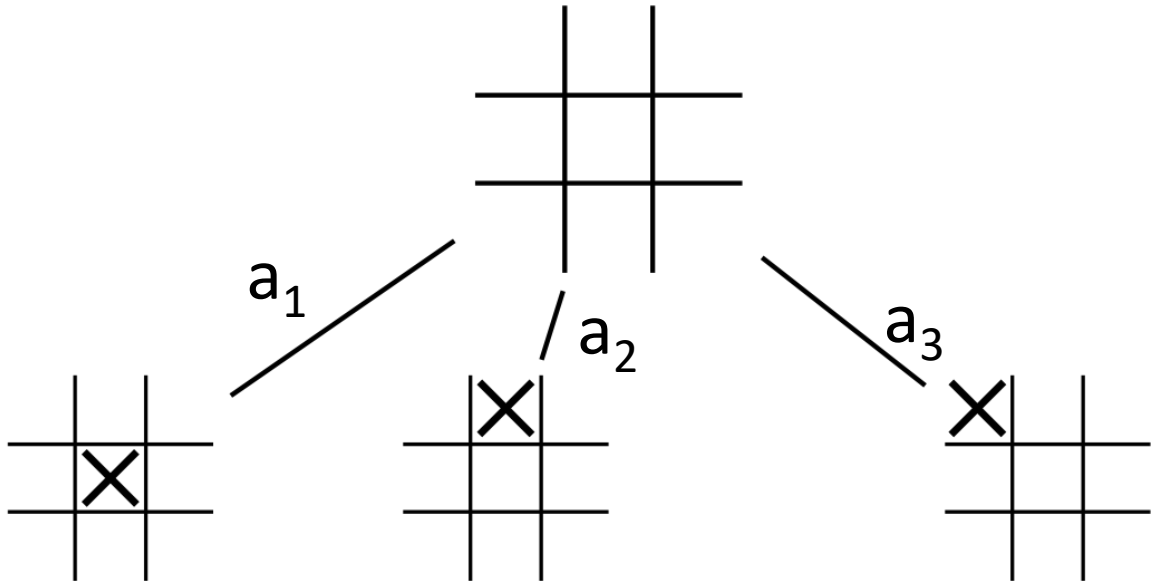


What move should I try?

Key ideas of Monte Carlo Tree Search:

1. View move selection as a multi-armed bandit problem
2. Evaluate moves by simulating games

Multi-Armed Bandit for Game Tree Search



What move should I try on each simulated game?

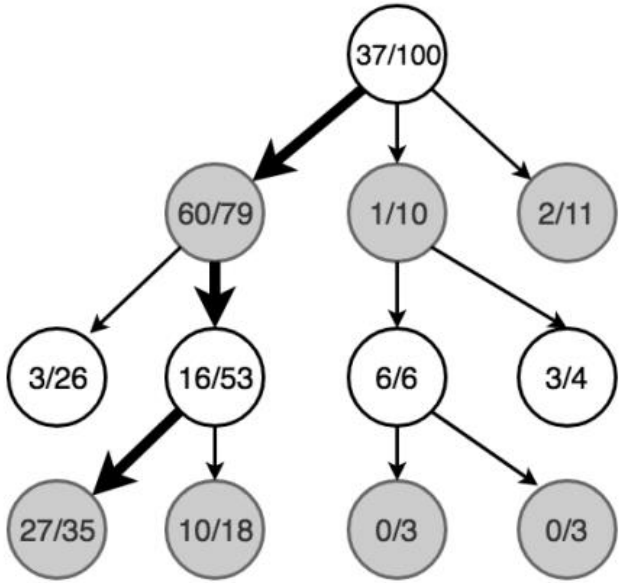
Monte-Carlo Tree Search (MCTS) Terms

- Leaf node: A state in the game tree that has successors for which no games have been simulated
- Terminal node: End of game state
- Playout/rollout: Simulating a game from a leaf node to a terminal node

Three Steps in MCTS

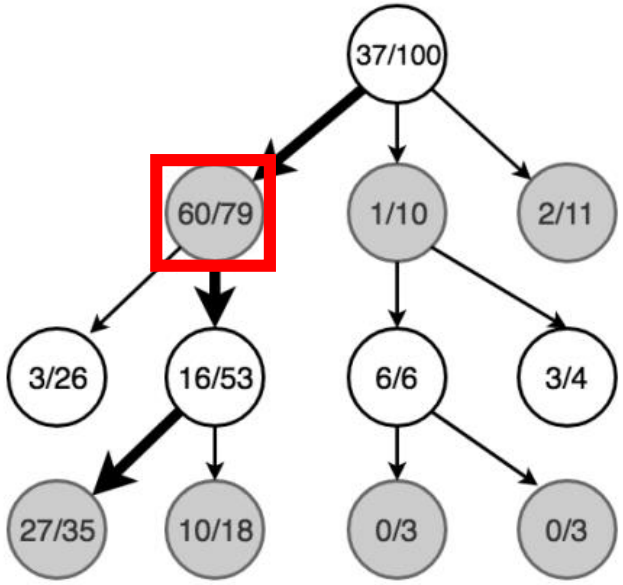
- Selection: Make move choices until a leaf node S is reached

Three Steps in MCTS



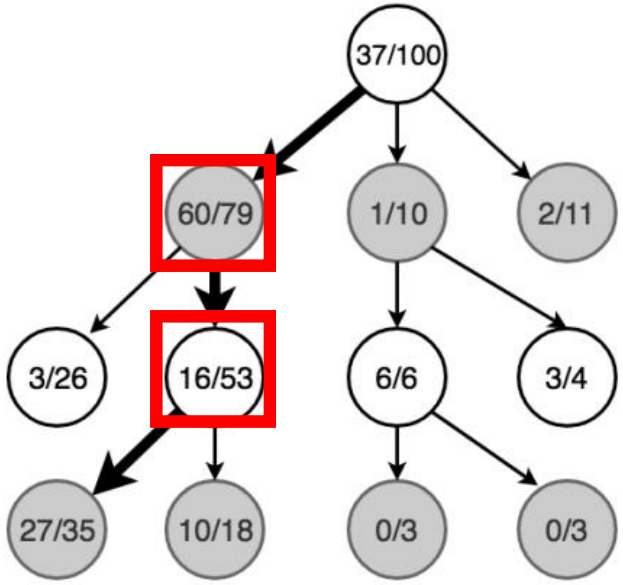
(a) Selection

Three Steps in MCTS



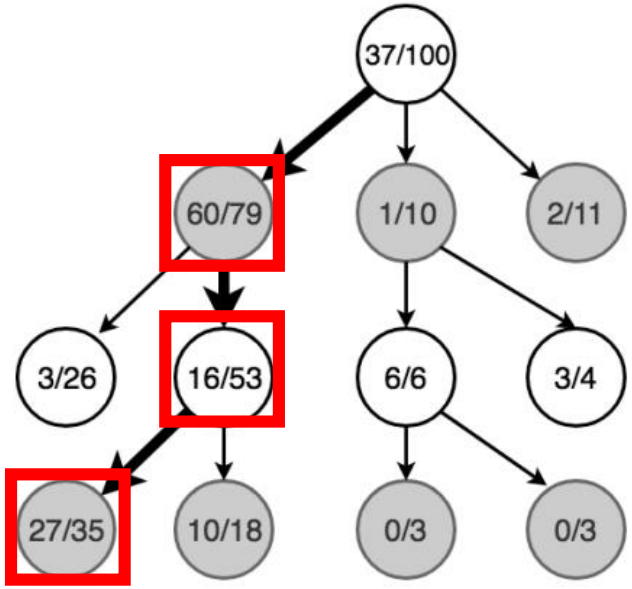
(a) Selection

Three Steps in MCTS



(a) Selection

Three Steps in MCTS



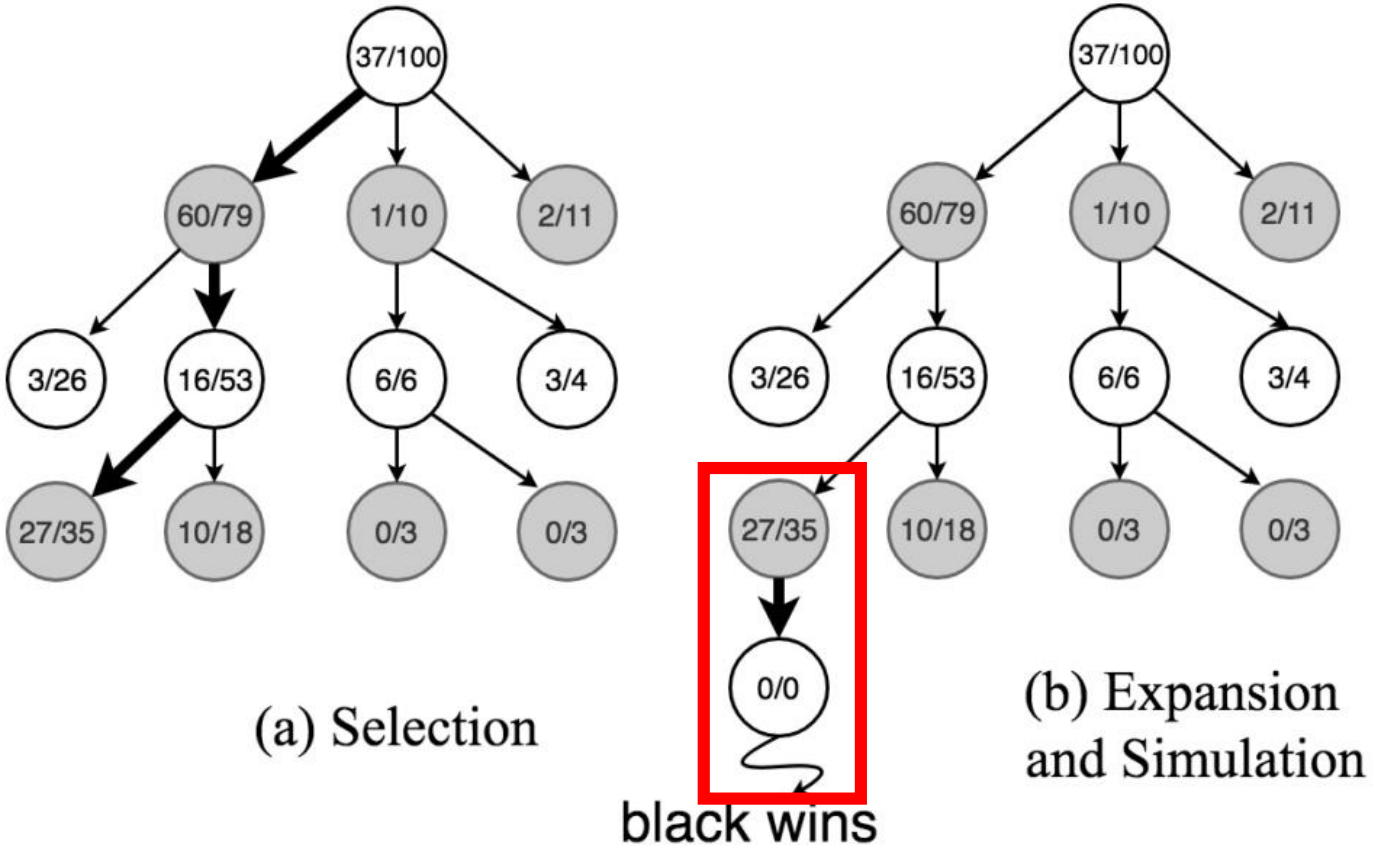
(a) Selection

Three Steps in MCTS

- Selection: Make move choices until a leaf node S is reached
- Expansion: Create a new successor state S’ for an untried action

  Simulation: Play a game until you reach a terminal node

Three Steps in MCTS

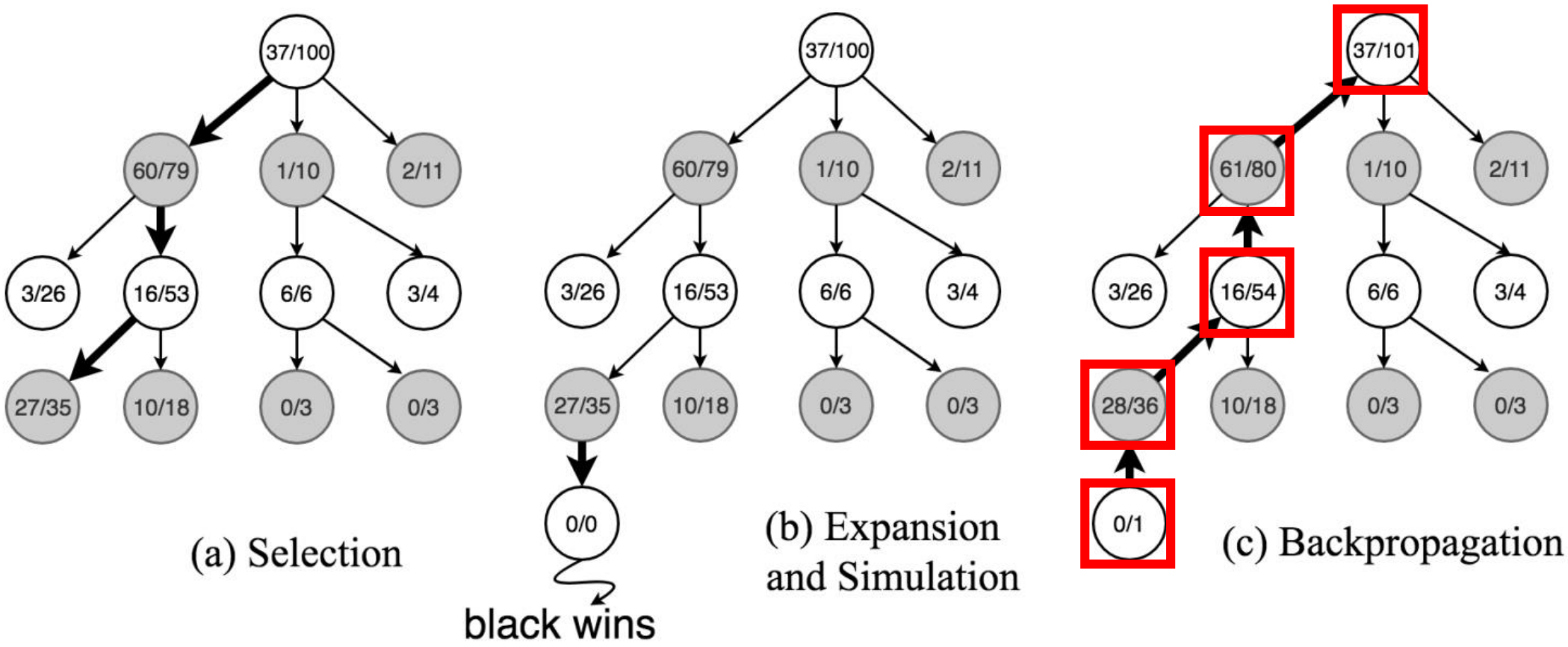


Three Steps in MCTS

- Selection: Make move choices until a leaf node S is reached
- Expansion: Create a new successor state S’ for an untried action

  Simulation: Play a game until you reach a terminal node
- Backpropagation: Update game statistics for the path from S’ up to the root

Three Steps in MCTS



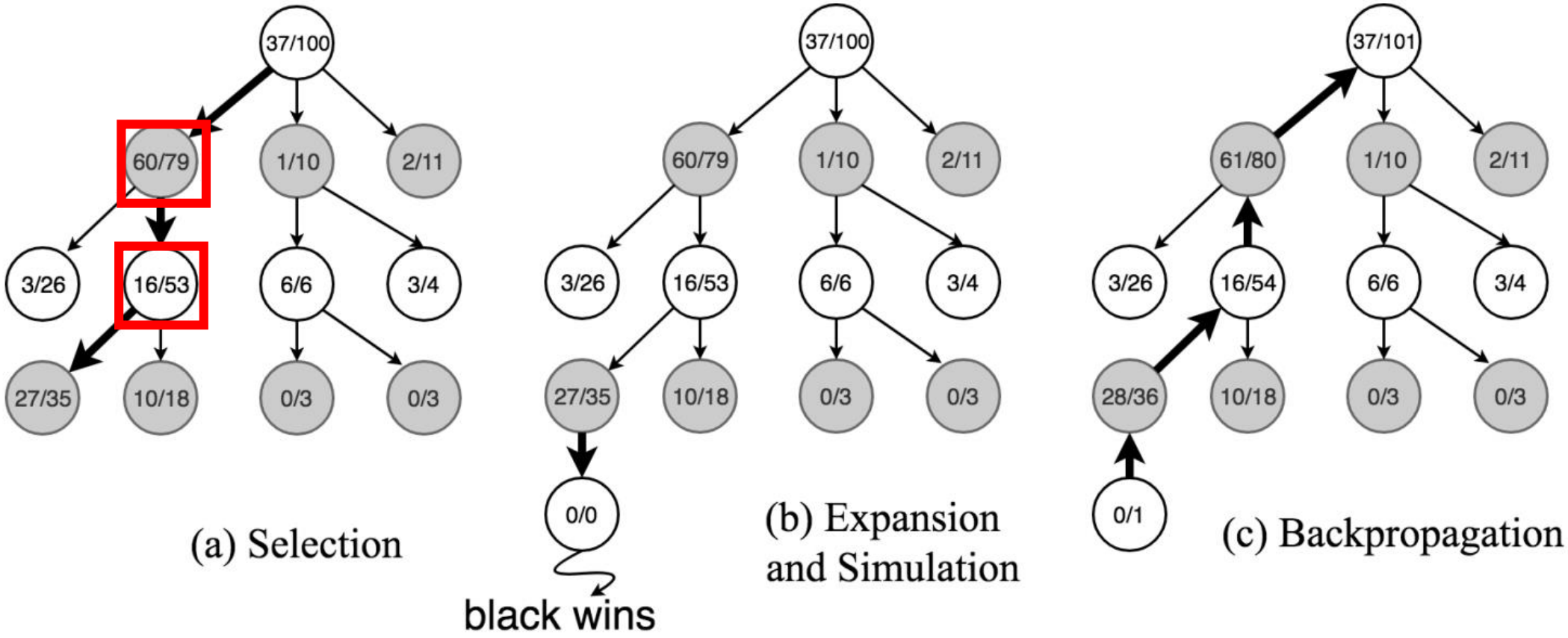
MCTS(state):

while TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACKPROPAGATE(result, child)
return $\underset{a \in A}{\operatorname{argmax}}$ #playouts(apply(a,state))

Which move gives the game state with most playouts

Three Steps in MCTS

How do we pick moves?



Remember This? (UCB)

Algorithm:

Pull each arm once

For i ← 1 to n { $Sum_i \leftarrow R(arm_i)$; $N_i \leftarrow 1$ }; $N \leftarrow n$ /* Initialization */

Loop Forever

best ← $\underset{1 \le i \le n}{\text{argmax}} \left[\frac{Sum_i}{N_i} + c\sqrt{\frac{\ln N}{N_i}} \right]$

pull arm a_{best} and get reward r

$Sum_{best} \leftarrow Sum_{best} + r$; $N_{best} \leftarrow N_{best} + 1$; $N \leftarrow N+1$

Picking a Move During Selection and Expansion (UCT – Upper Confidence bound applied to Trees)

Sum_i = # of wins

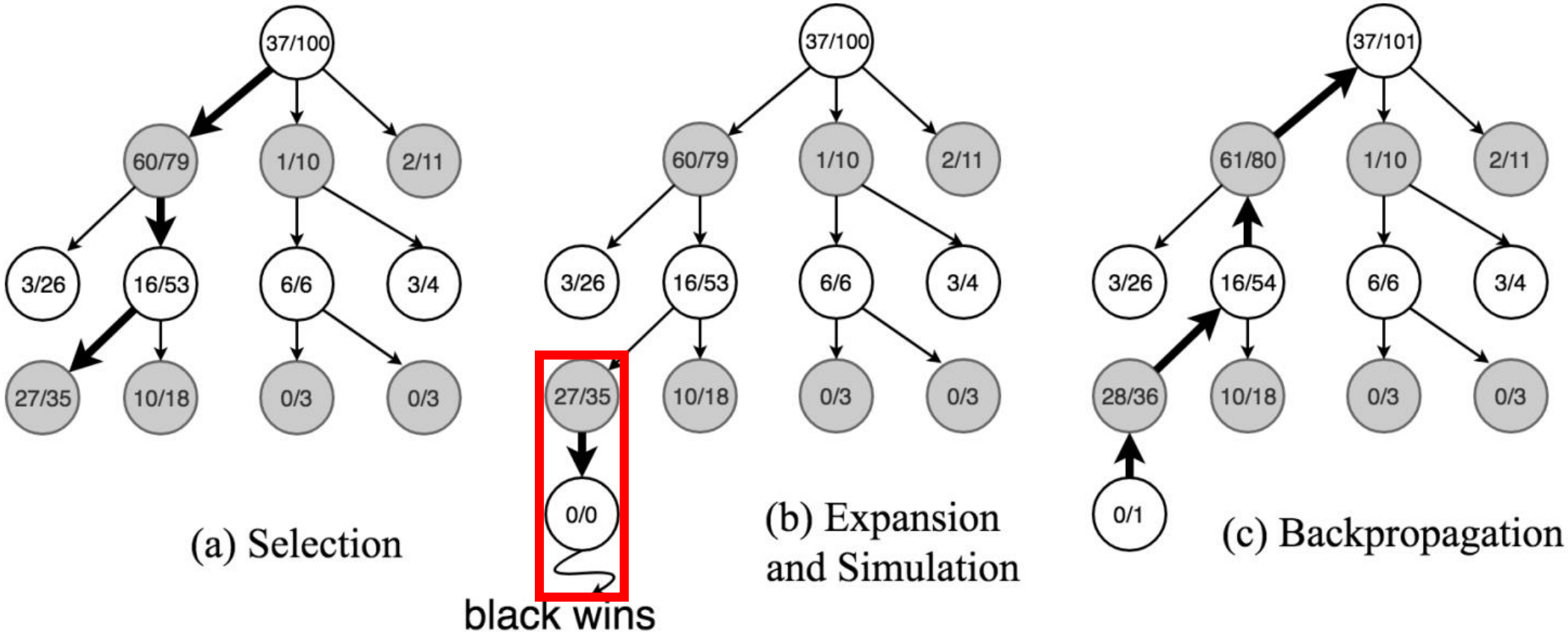
N_i = # of times i was tried

N = # of simulations thus far (N(parent(i))

$$best \leftarrow \underset{1 \le i \le n}{\operatorname{argmax}} \left[\frac{Sum_i}{N_i} + c\sqrt{\frac{\ln N}{N_i}} \right]$$

Lets you control how much exploration

Three Steps in MCTS



How do we pick moves?

Picking a Move During Simulation

- Light playout: Pick uniformly at random
- Heavy playout: Make a biased selection
  - Simulation statistics
  - Game knowledge

Trade off: Slower run time vs missing a move

Benefits

- Doesn’t use an evaluation function!
- Time is linear in depth
- Handles large number of actions
- Let’s you make a move when a timer goes off (to manage time)
  (“anytime algorithm”)

AlphaGo / AlphaZero

- Truncated playouts and used (learned) evaluation function
- UCB with additional term for (learned) probability of win