Monte Carlo Tree Search

AI+X
INSTITUTE FOR ARTIFICIAL INTELLIGENCE

Alpha-Beta Pruning Weakness

- When the branching factor is high, minimax with Alpha-Beta pruning must run in "depth-limited" mode
  - Stop when maximum depth is reached, then use an heuristic function to evaluate the current state of the game
  - Example:
    - The branching factor of Go is 361
    - The search cannot go deeper than 4-5 moves
    - It is difficult to define a good evaluation function for Go
      - material value is not a strong indicator
      - most positions are in flux until the endgame

Monte Carlo Algorithms

- Randomized algorithms named after the Casino de Monte-Carlo in Monaco
- Example
  - Estimate the area of a circle with radius 1 by randomly sampling points within a 2×2 square and measuring their distance to the center of the square. Points whose distance is smaller than or equal to 1 are inside the circle.
  - Area = 4 × |number of points in circle| / |number of sampled points|

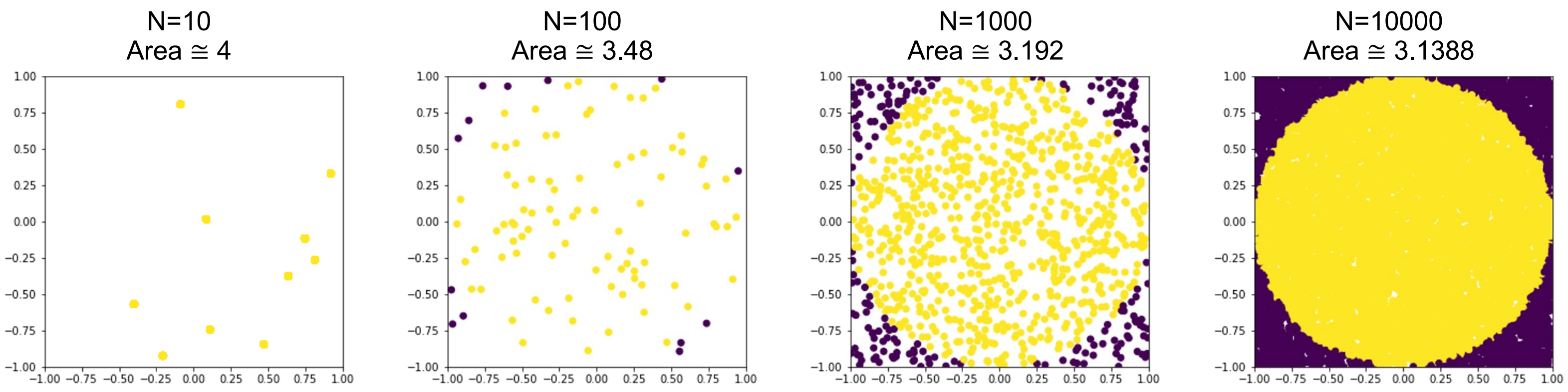

Figure 1. Monte Carlo algorithms.

Knowledge Check 1

Monte Carlo simulations can be used for many applications. For instance, we can estimate the probability of having two people with the same birthday in a group of N people using the code on the right. For which value of N this probability is higher than 50%?

```
import random

def rand():
  return random.randint(1, 365)

def montecarlo(N, simulations=10000):
   count = 0
   for i in range(simulations):
       birthdays = set([rand() for _ in range(N)])
       if len(birthdays) < N:
           count += 1
   return count/simulations
```

- A 13
- B 23
- C 33
- D 43

Monte Carlo Tree Search

- The value of a state is estimated as the average utility over a number of simulations of complete games starting from the state
- A simulation (or playout) chooses moves first for one player, than for the other, repeating until a terminal position is reached
- At the terminal position, the rules of the game determine who has won or lost, and by what score
- For games in which the only outcomes are a win or a loss, “average utility” is the same as “win percentage”

Monte Carlo Tree Search

- Just choosing moves randomly during playouts is not good enough for most games
  - Use a playout policy to bias the moves towards good ones
  - Example
    - Prioritize capture moves in chess
- Starting multiple playouts from a state to choose the next move is not good enough for most games

MCTS uses a search tree and a selection policy to balance exploration and **exploitation.**

Monte Carlo Tree Search

Repeat:

01 **Selection**
Starting at the root of the search tree, choose moves guided by the selection policy until a leaf is reached.

02 **Expansion**
Grow the search tree by generating a new child for the selected node.

03 **Simulation**
Perform a playout from the newly generated node, choosing moves for both players according to the playout policy.

04 **Back-Propagation**
Use the result of the simulation to update the winning statistics of all search tree nodes going up to the root.

Selection Policy

- The value of a state is estimated as the average utility over a number of simulations of complete games starting from the stat
- Upper confidence bounds applied to trees (UCT)
  - The policy ranks each possible move based on an upper confidence bound formula called UCB1

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\mathrm{PARENT}(n))}{N(n)}}$$

U(n) is the total utility of all playouts through node *n*
N(n) is the number of playouts through node *n*
p is the parent node of *n*
C is a constant that balances exploitation and exploration ($C \approx 2^{1/2}$)

Selection Policy

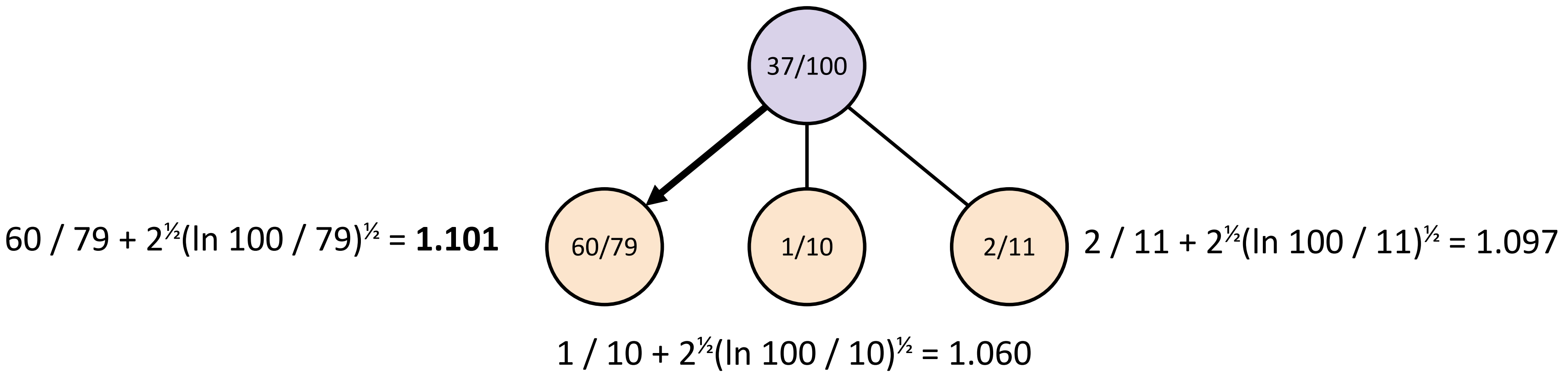

Figure 2. UCT selection policy.

Monte Carlo Tree Search

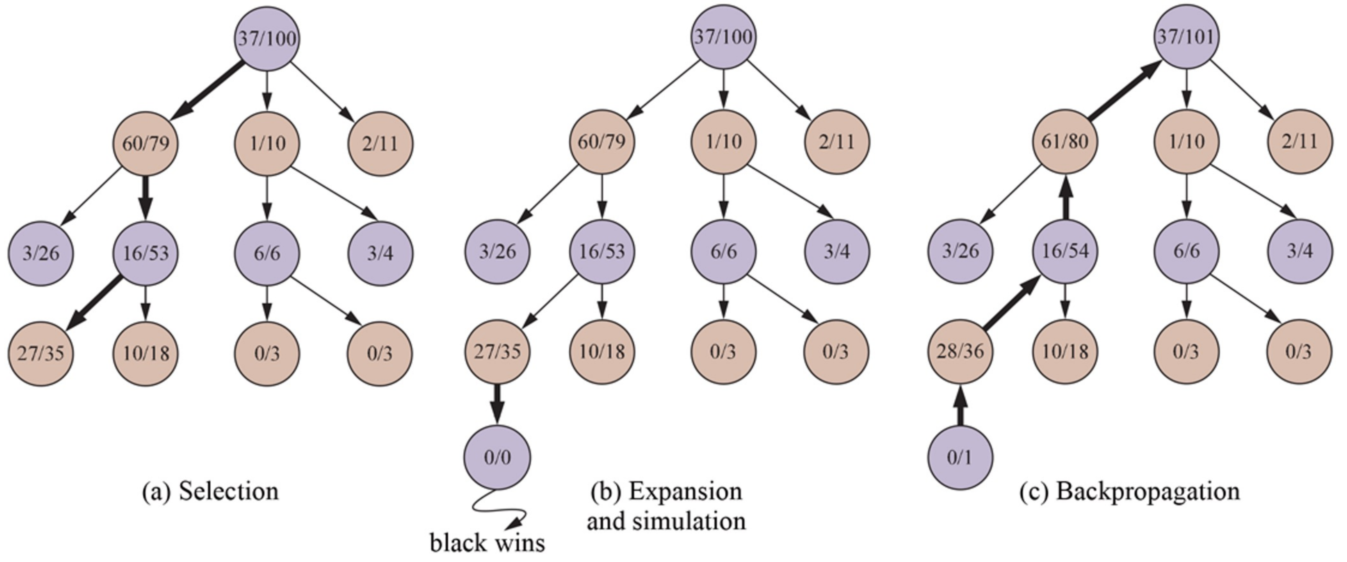

Figure 3. One iteration of the process of choosing a move with MCTS using the UCT selection policy.

Monte Carlo Tree Search

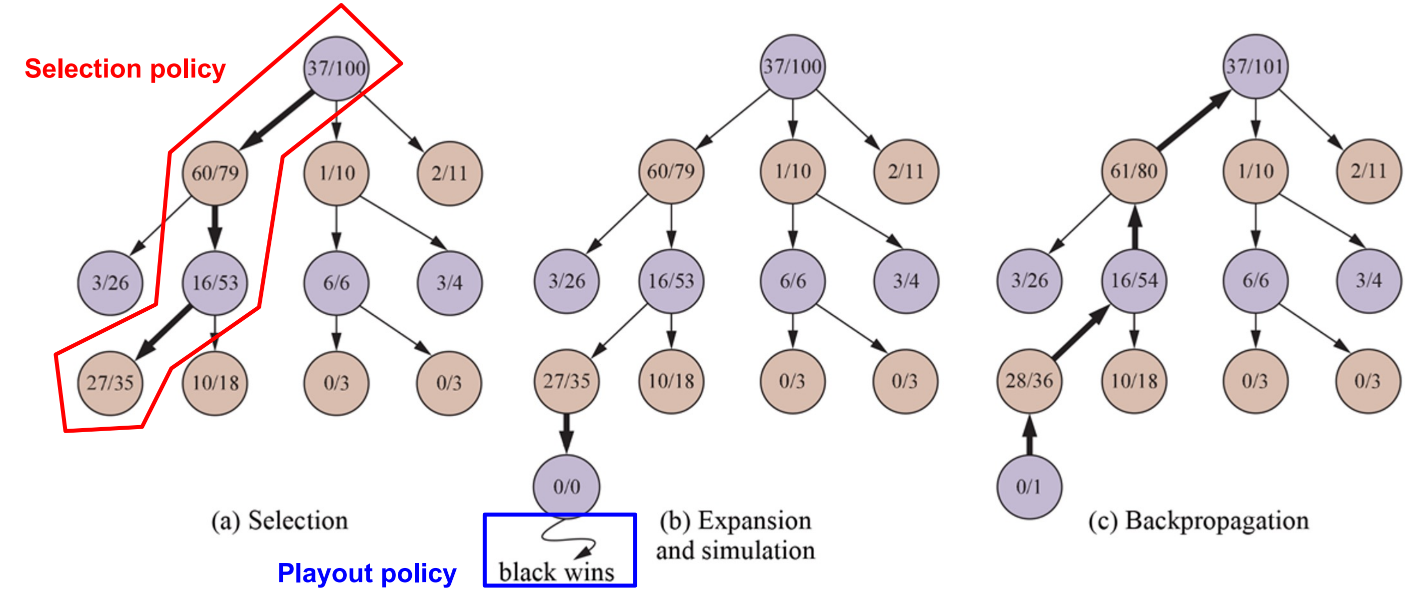

Figure 4. One iteration of the process of choosing a move with MCTS using the UCT selection policy.

Knowledge Check 2

When the average game length is high, the playout simulation might take too long to reach a terminal state. For those problems, one possible solution is stopping the simulation early and:

- A picking a random winner.
- B considering this playout a draw.
- C using a heuristic function to estimate the utility.
- D restarting the playout with a different random seed.

You have reached the end of the lecture.

Image/Figure References

Figure 1. Monte Carlo algorithms.
Figure 2. UCT selection policy. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 3. One iteration of the process of choosing a move with MCTS using the UCT selection policy. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 4. One iteration of the process of choosing a move with MCTS using the UCT selection policy. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.