UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

Artificial Intelligence

Adversarial Search

LESSON 8

prof. Antonino Staiano

M.Sc. In ''Machine Learning e Big Data'' - University Parthenope of Naples

Adversarial Search

- Game theory
- Optimal Decisions in Games
  - Minimax decisions
  - $\alpha - \beta$ pruning
  - Monte Carlo Tree Search

  - Games of chance
  - Few words on Games of imperfect information
  - Limitations of game search algorithms

Monte Carlo Tree Search

- For more complex games, like Go, alpha-beta search with limited depth remains an unfeasible way to walk
- A possible effective and different alternative is Monte Carlo Tree Search (MCTS)
  - It doesn't use a heuristic evaluation function
  - Starting from a state, its value is estimated as an average utility over several simulations (also called playouts) of the game
  - The simulation chooses alternating moves for the players until a terminal position is reached

Playout Policy

- How do we choose what moves to make during the playout?
- Playout policy helps to decide what moves to make during the playout
  - It biases the moves toward good ones
    - Learned by self-play by using neural networks (e.g., Go)
    - Game-specific heuristics (e.g., Chess, Othello)

Pure Monte Carlo Search

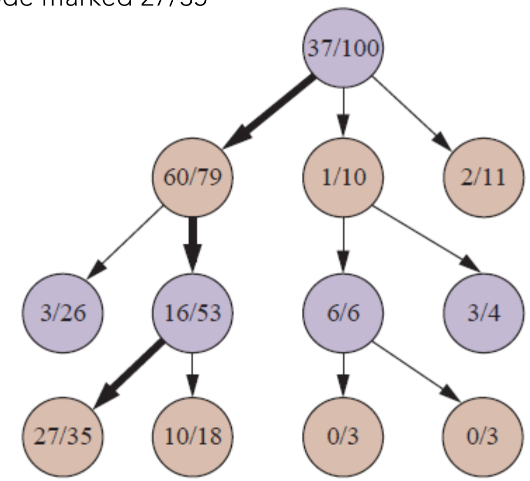
- Given a playout policy
  - From what positions do we start the playouts?
  - How many playouts do we allocate to each position?
- Pure Monte Carlo search
  - N simulations starting from the current state of the game
  - Track among the possible moves the one with the highest win percentage
  - As N increases, this strategy could converge to optimal play
    - However, in general, it is not enough
- A selection policy is needed to selectively focus the computational resources on the important parts of the game tree
  - A trade-off between exploration and exploitation

MCTS

- Iteratively maintains a search tree and grows it at each step, performing
  - Selection
    - Starting from the root, chooses, according to the selection strategy, a move to a successor node
    - Repeat the process going down the tree to a leaf
  - Expansion
    - Grows the search tree by generating a new child of the selected node
  - Simulation
    - Performs a playout from the newly generated child node
  - Back-propagation
    - The result of the simulation updates all the nodes going up to the root
- The four steps are repeated for a set number of iterations, or until the allotted time has expired
  - then return the move with the highest number of playouts

MCTS steps: Selection

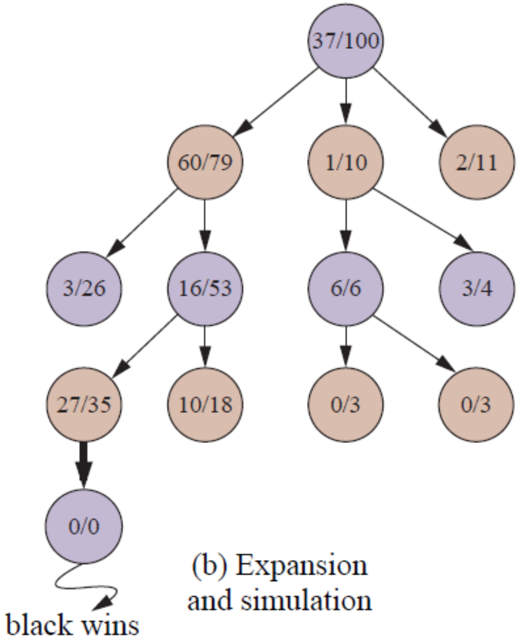
- A search tree with the root representing a state where white has just moved, and white has won 37 out of the 100 playouts
  - The thick arrow shows the selection of a move by black that leads to a node where black has won 60/79 playouts
    - This is the best win percentage among the three moves, so selecting it is an example of exploitation
    - But it would also have been reasonable to select the 2/11 node for exploration purposes
    - Selection continues to the leaf node marked 27/35



(a) Selection

MCTS steps: Expansion and Simulation

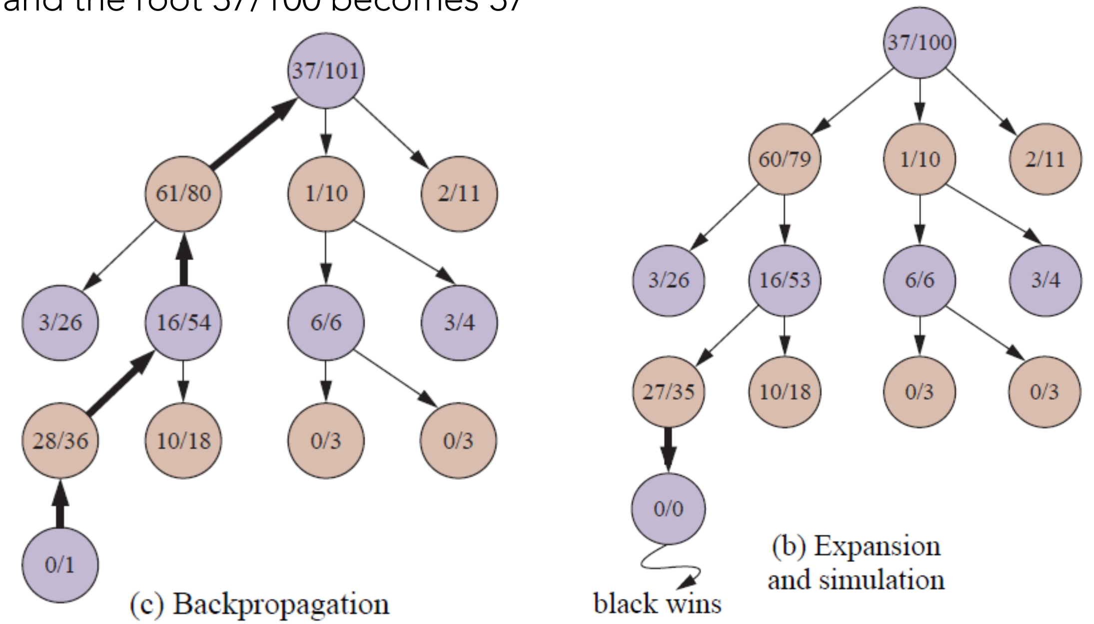
- The Expansion step grows the search tree by generating a new child of the selected node
  - the new node marked with 0/0
- Simulation moves for both players according to the playout policy
  - The moves are not recorded in the search tree
    - The simulation here results in a win for black



(b) Expansion and simulation

MCTS steps: Backpropagation

- The simulation results are used to update all the search tree nodes going up to the root
- Black nodes are incremented in both the number of wins and the number of playouts
  - 27/35 becomes 28/36 and 60/79 becomes 61/80
- Since white lost, the white nodes are incremented in the number of playouts only
  - 16/53 becomes 16/54 and the root 37/100 becomes 37



Selection Policy

- Upper Confidence Bounds to Trees (UCT)
  - Ranks each possible move according to the Upper Confidence Bound (UCB1) formula

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

  - where
    - U(n) is the total utility of all playouts that went through node n
    - N(n) is the number of playouts through node n
      - The first term is the average utility, the exploitation term
    - PARENT(n) is the parent node of n in the tree

UCT MCTS Algorithm

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree ←  NODE(state)
  while IS-TIME-REMAINING() do
       leaf  ←  SELECT(tree)
       child ←  EXPAND(leaf )
       result ←  SIMULATE(child)
       BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

- The time to compute a playout is linear in the depth of the game tree
  - because only one move is taken at each choice point

On Monte Carlo Search

- Monte Carlo search has an advantage over alpha–beta for games where
  - the branching factor is very high (e.g, Go)
  - when it is difficult to define a good evaluation function
  - It relies on the aggregate of many playouts and thus is not as vulnerable to a single error
- MCTS and evaluation functions could be combined
  - playout for a certain number of moves, then truncate the playout and apply an evaluation function
- Monte Carlo search can be applied to brand-new games
  - No experience to consider for defining an evaluation function
  - No additional information and only the game rules
  - Good policies can be learned using neural networks trained by self-play alone

Deterministic Games in Practice

- Checkers:
  - Chinook ended the 40-year reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions
- Chess:
  - Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997
  - Deep Blue ran alpha–beta search at over 100 million positions per second, and used very sophisticated evaluation, and undisclosed methods for extending some lines of search
- Othello:
  - human champions refuse to compete against computers, which are too good
    - Programs have been at superhuman level since 1997
- Go:
  - Alphago by DeepMind defeated human champion Lee Sedol (2015) . In go, b > 300, so most programs use pattern knowledge bases to suggest plausible moves
  - In 2018, AlphaZero surpassed Alphago by learning through self-play without any expert human knowledge and without access to any past games
    - It does rely on humans to define the basic architecture as Monte Carlo tree search with deep neural networks and reinforcement learning, and to encode the rules of the game

Types of Games

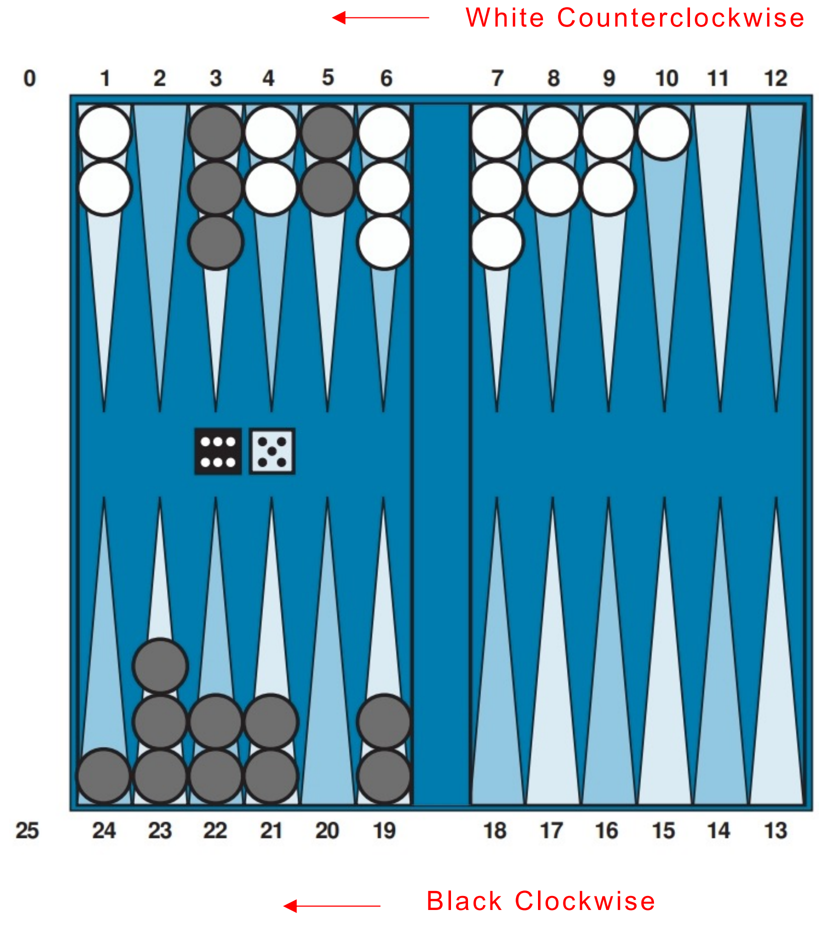
	deterministic	chance
perfect information	chess, checkers, go, othello	Backgammon, monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble

Stochastic Games

- Stochastic games include a random element
- Backgammon is an example that combines luck and skill
- The goal of the game is to move all one's pieces off the board
  - Black moves clockwise toward 25
  - White moves counterclockwise toward 0
  - A piece can move to any position unless multiple opponent pieces are there
    - if there is one opponent, it is captured and must start over

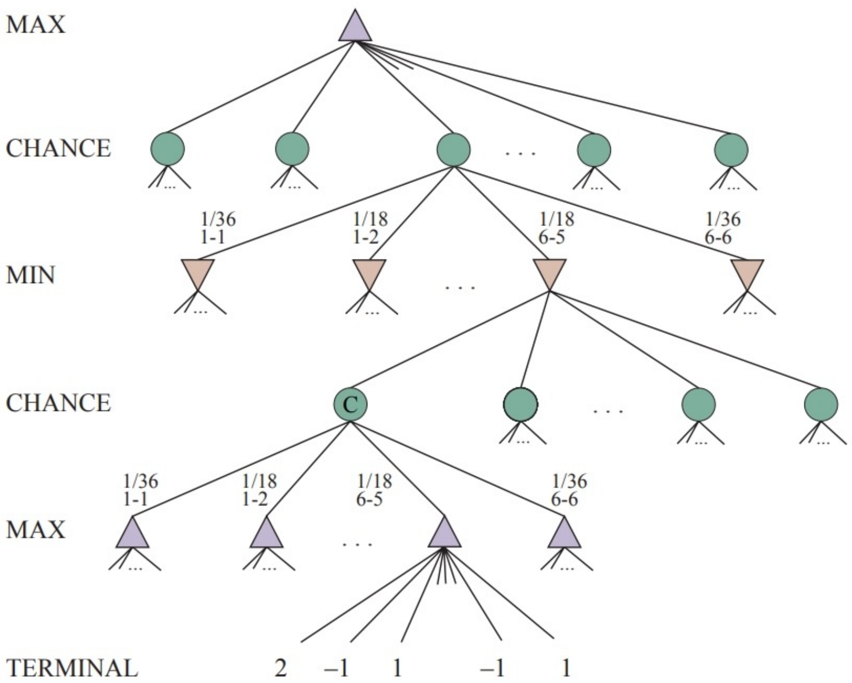
Non-deterministic games: Backgammon

- Black has rolled 6–5 and must choose among four legal moves:
  - (5–11,5–10)
  - (5–11,19–24)
  - (5–10,10–16)
  - (5–11,11–16)
- (5–11,11–16) means
  - move one piece from position 5 to 11 and then move a piece from 11 to 16
- Now, Black knows what moves can be made but does not know what White is going to roll and thus does not know what White's legal moves will be
- so Black cannot construct a standard game tree



Non-deterministic games in general

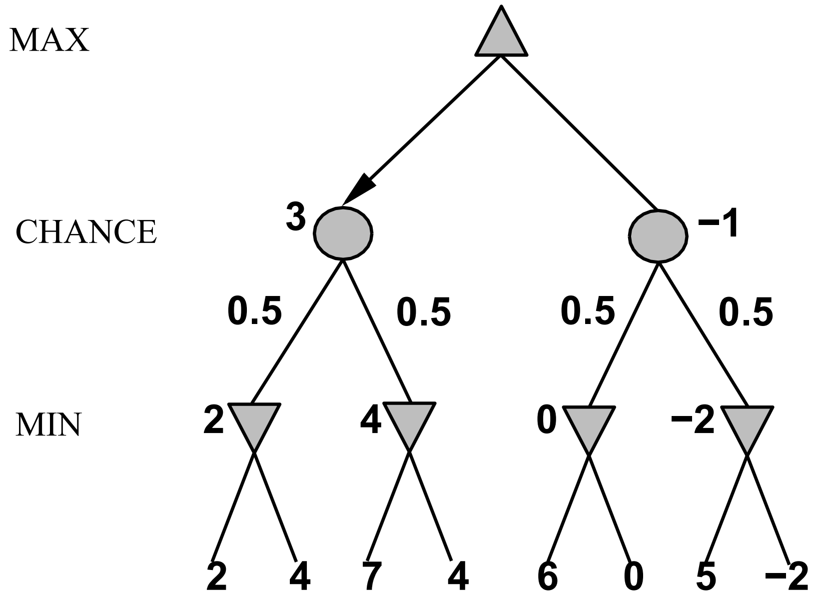
- A game tree in backgammon must include chance nodes in addition to MAX and MIN nodes
- In non-deterministic games, chance introduced by dice, card-shuffling



- The branches leading from each chance node denote the possible dice rolls
  - each branch is labeled with the roll and its probability
  - There are 36 ways to roll two dice, each equally likely
    - 6–5 is the same as a 5–6, there are only 21 distinct rolls
    - The six doubles (1–1 through 6–6) each have a probability of 1/36, so P(1–1) = 1/36
    - The other 15 distinct rolls each have a 1/18 probability

Non-deterministic games in general

- Simplified example with coin-flipping:



ExpectMiniMax

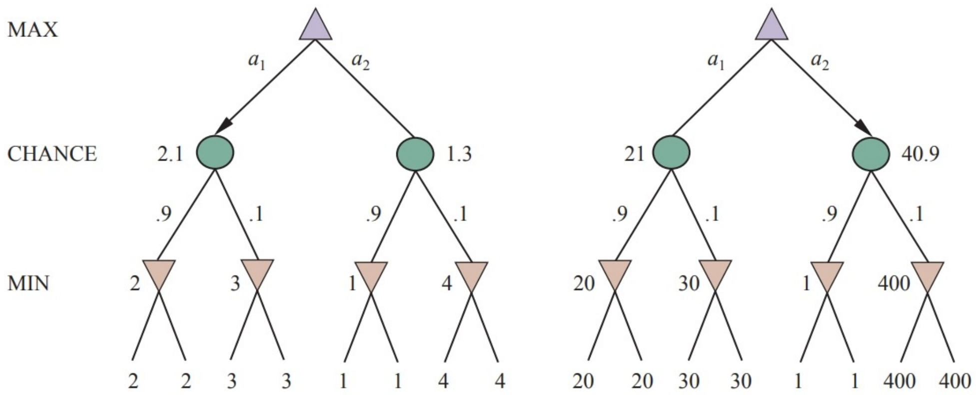
- We want to pick the move that leads to the best position
  - However, positions do not have definite minimax values
  - We can only calculate the expected value of a position: the average over all possible outcomes of the chance nodes
  - Expectminimax value
    - Generalization of the minimax value for deterministic games

$$
\text{EXPECTIMINIMAX}(s) =
\begin{cases}
\text{UTILITY}(s, \quad) & \text{if IS-TERMINAL}(s) \\
\max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\
\min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \\
\sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if TO-MOVE}(s) = \text{CHANCE}
\end{cases}
$$

    - r is a possible dice roll
    - RESULTS(s,r) same state as s with r as the result of the dice roll

Evaluation Functions

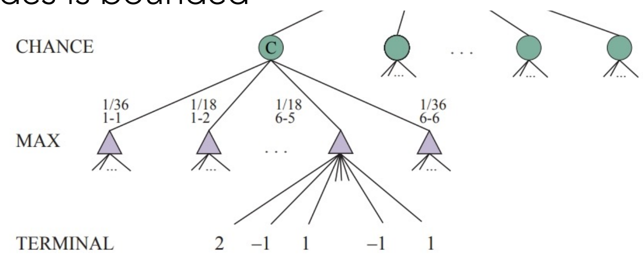
- Expectminimax could be approximated by cutting the search off at some point and then applying an evaluation function to each leaf
  - The chance nodes, however, need to be involved in evaluating the value of the positions



  - The program behaves differently if some of the evaluation values change, even if the preference order remains the same
    - So, the evaluation function must return values that are a positive linear transformation of the probability of winning

Non-Deterministic Games in Practice

- Because expectiminimax considers all the possible dice-roll sequences, it will take $O(b^m n^m)$
  - where n is the number of distinct rolls
- Even for small depths d, it would be unfeasible to look ahead very far
  - In backgammon n is 21 and b is usually around 20, but in some situations can be as high as 4000 for dice rolls that are doubles
- However, alpha-beta pruning can be applied also to game trees with chance nodes
- if we put bounds on the possible values of the utility function, then we can arrive at bounds for the average without looking at every number
  - For example, if all utility values are between −2 and +2; then the value of leaf nodes is bounded
    - We can place an upper bound on the value of a chance node without looking at all its children



Partially Observable Games

- In deterministic partially observable games, uncertainty about the state of the board arises entirely from a lack of access to the opponent's choices
- This class includes games such as Battleship
  - each player's ships are placed in locations hidden from the opponent
- In stochastic partially observable games, the missing information is generated by the random dealing of cards
  - bridge, whist, hearts, and poker

Card Games

- At first sight, it might seem that these card games are just like dice games
  - the cards are dealt randomly and determine the moves available to each player, but all the “dice” are rolled at the beginning!
  - it suggests an algorithm:
    - treat the start of the game as a chance node with every possible deal as an outcome and then use the EXPECTIMINIMAX formula to pick the best move
  - Note that in this approach the only chance node is the root node
  - Then, the game becomes fully observable

Property Analysis

- The intuition that the value of an action is the average of its values in all actual states is WRONG
- With partial observability, the value of an action depends on the information state or belief state the agent is in
  - optimal play requires reasoning about the current and future belief states of each player
- Can generate and search a tree of information states
- This leads to rational behaviors such as
  - Acting to obtain information
  - Signaling to one's partner
  - Acting randomly to minimize information disclosure

Stochastic and/or Partially Observable Games in Practice

- Backgammon
  - BKG (1980) used a manually constructed evaluation function and searched at depth 1 only
    - First program to defeat a human world champion
  - TD-Gammon (1995) learned its evaluation function using NNs trained by self-play
- Poker
  - Game theory (2015) to determine the exact optimal strategy for a version of poker with just two players
  - In 2017, champion poker players were beaten at heads-up (two players) no-limit Texas hold 'em in two separate matches against the programs Libratus and DeepStack
  - In 2019, Pluribus defeated top-ranked professional human players in Texas hold 'em games with six players
- Bridge
  - GIB program, based on Monte Carlo simulation, won the computer championship and did surprisingly well against expert human players
  - In the 21st century, the computer bridge championship has been dominated by two commercial programs, JACK and WBRIDGE5

Limitations of Game Search Algorithms

- Alpha–beta search vulnerable to errors in the heuristic function
- Waste of computational time for deciding the best move where it is obvious (meta-reasoning)
  - Both alpha-beta and MCST
- The reasoning is done on individual moves
  - Humans reason on abstract levels
- Possibility to incorporate Machine Learning into the game search process

Summary

- Minimax algorithm: selects optimal moves by a depth-first enumeration of the game tree
- Alpha–beta algorithm: greater efficiency by eliminating subtrees
- Evaluation function: a heuristic that estimates the utility of state
- Monte Carlo tree search (MCTS): no heuristic, play game to the end with rules and repeated multiple times to determine optimal moves during playout