Adversarial Search And Games

Jacopo Mauro

Slide Based on Slides of the Artificial Intelligence: A Modern Approach book

Types of games

	Deterministic	Chance
Perfect Information	chess, checkers, go, othello	backgammon monopoly
Imperfect Information	battleships, cluedo	bridge, poker, scrabble nuclear war

Games vs. search problems

"Unpredictable" opponent ⇒ solution is a strategy specifying a move for every possible opponent reply

Time limits ⇒ unlikely to find goal, must approximate plan of attack

Two-player zero-sum games

Setting:

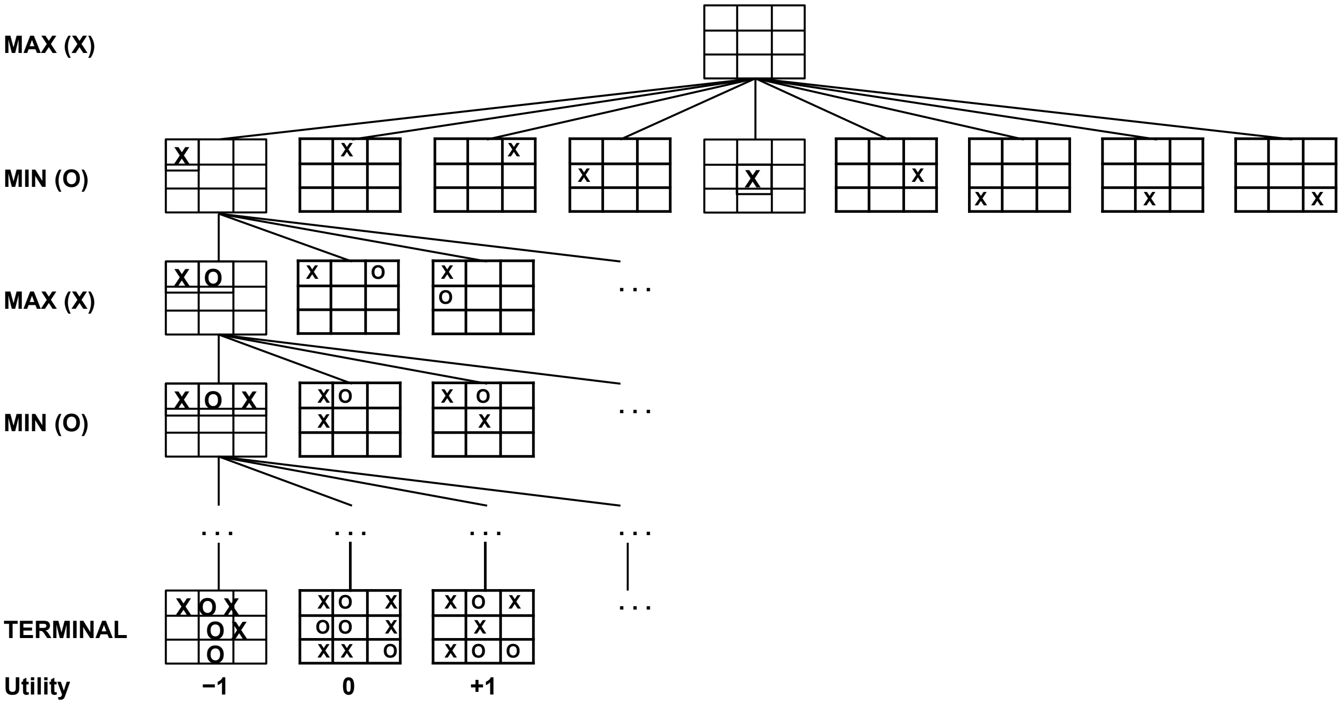
- Two players (one has the goal to max, one to min)
- Taking turns
- Fully observable

Moves: Action

Position: State

Zero sum: good for one player, bad for another (no win-win outcome).

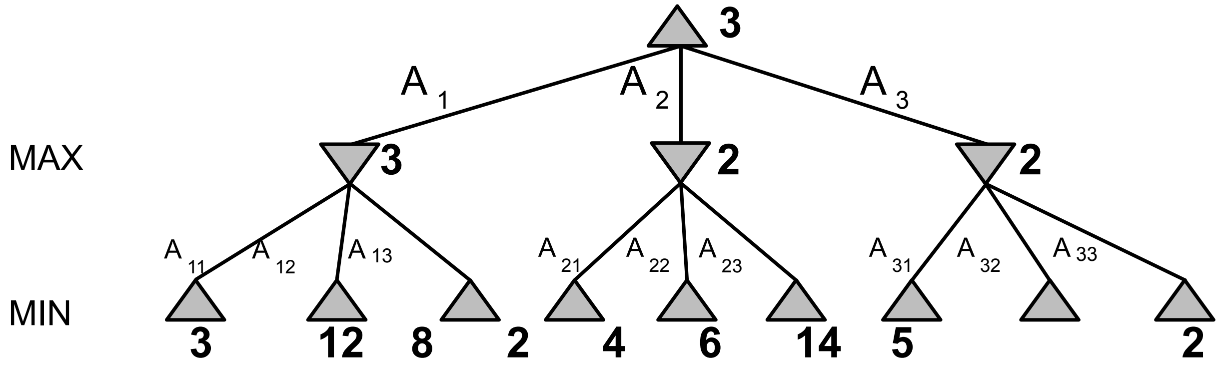
Game tree (2-player, deterministic, turns)



Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value = best achievable payoff against best play



Minimax algorithm

```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state)
  return move

function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

Properties of minimax

Complete: Yes, if tree is finite (chess has specific rules for this)

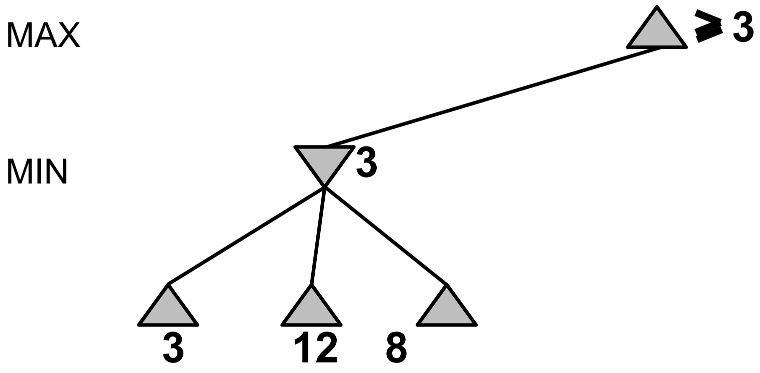
Optimal: Yes, against an optimal opponent.

Time complexity: Consider all possible moves ($O(b^m)$, for chess, $b \approx 35$, $m \approx 100$ for “reasonable” games)

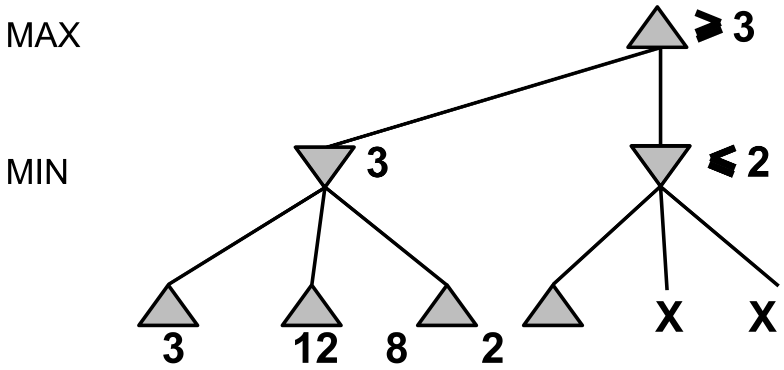
Space complexity: $O(bm)$

Can we do better? Do we need to explore every path?

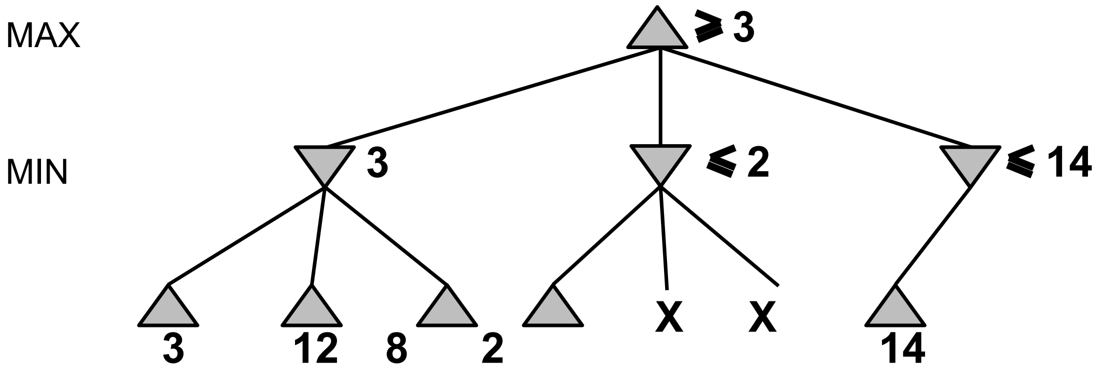
α-β Pruning



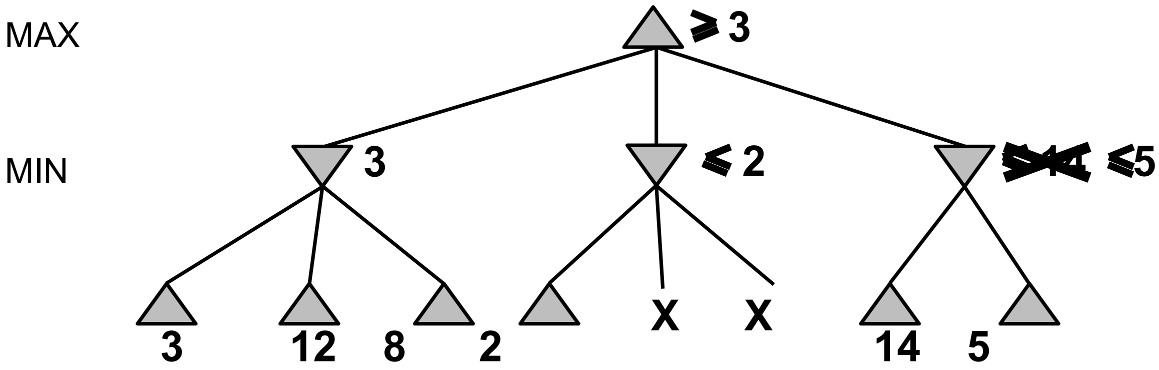
α-β Pruning



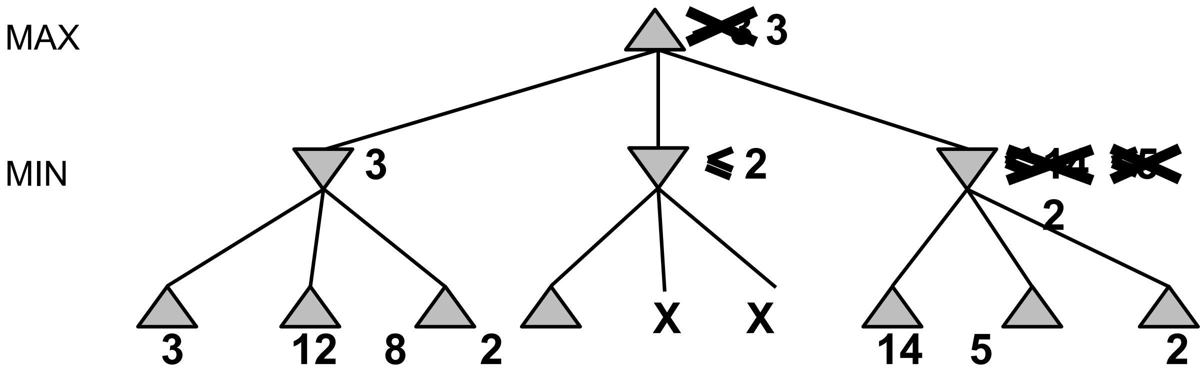
α-β Pruning



α-β Pruning



α-β Pruning



Why is it called α-β ?

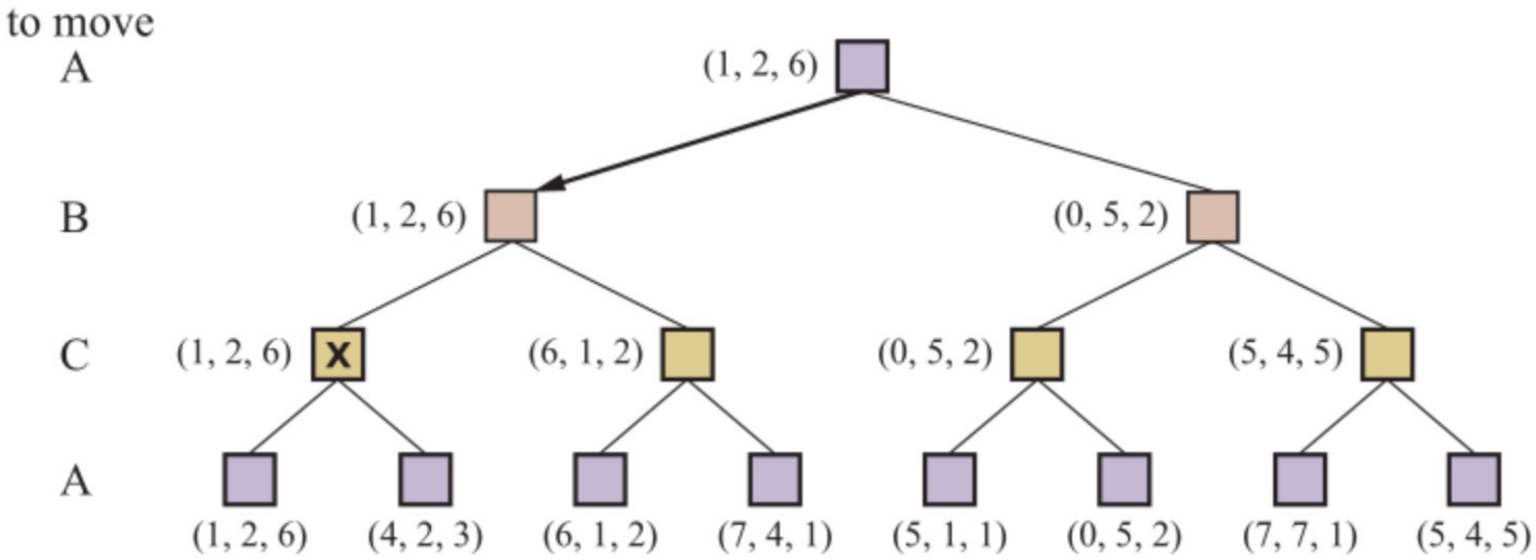
Alpha (α)

- best value that the maximizing player can guarantee at any point
- branch is pruned if the minimizing player can provide a value less than α (maximizing player would never choose it)

Beta (β)

- best value that the minimizing player can guarantee at any point.
- branch is pruned if the maximizing player can provide a value greater than β (minimizing player would never choose it)

More than 2 players



Limited Computation Time

Idea: cut at a certain depth

- Requires heuristic that provides an estimate of a position’s desirability

How it Works

- Define a Depth Limit + Apply the Evaluation Function
- Approximates which player has a better position
- Backpropagate Scores Using Minimax
  - Maximizing player chooses the highest evaluation
  - Minimizing player chooses the lowest evaluation
- Problem: horizon effect
  - Minimax fails to see beyond its depth limit → make short-sighted moves

Forward Pruning

Idea: avoid expanding moves

- Minimax evaluates too many moves
- Selects only the most promising moves, skipping weaker ones to save time
  - Beam Search – keeps only the top N moves based on a heuristic score
- Risk in not seeing a good move

Monte Carlo Tree Search

Combines random sampling and incremental tree building to evaluate potential moves

Value of a state is estimated as the average utility over number of simulations

Playout:

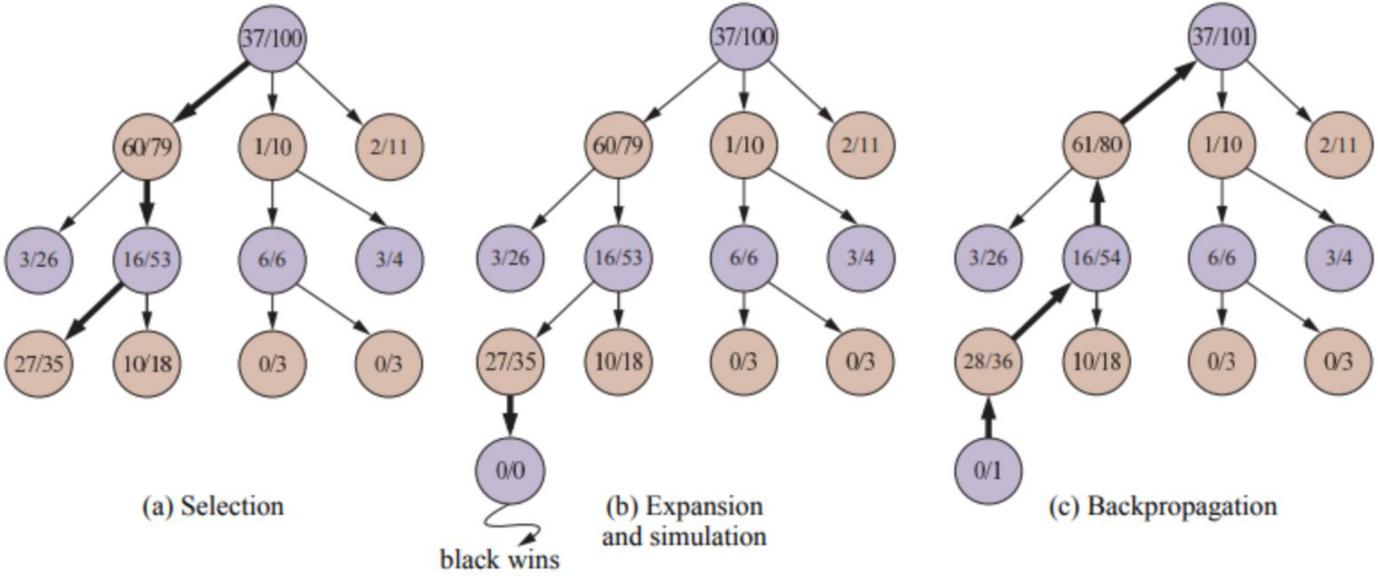
- Simulation that chooses moves to do
- Repeat until terminal position reached.

Monte Carlo Tree Search

Steps:

- Selection: navigate the tree using a balance of exploitation (best average score) and exploration (least visited nodes).
- Expansion: Add new child nodes when unexplored moves are encountered.
- Simulation (Rollout): Simulate random playouts from the new node to estimate outcomes.
- Backpropagation: Update the scores of all nodes along the path to the root based on the simulation result.

Monte Carlo Tree Search



Selection Policy

Effective selection policy is called “Upper Confidence bounds applied to Trees” (UCT)

UCT ranks each possible move based on an upper confidence bound formula

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

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
- PARENT(n) is the parent node of n in the tree
- C is a constant balances exploitation and exploration (theoretical argument that should be √2 but in practice try and choose the one that performs best)

Monte Carlo Tree Search

```
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
```

Resource limits

Cutoff-Test Instead of Terminal-Test

- Cutoff-Test: Stops the search at a predefined depth or when a time limit is reached.
- Rationale: Full search to terminal states is often infeasible

Eval Function Instead of Utility

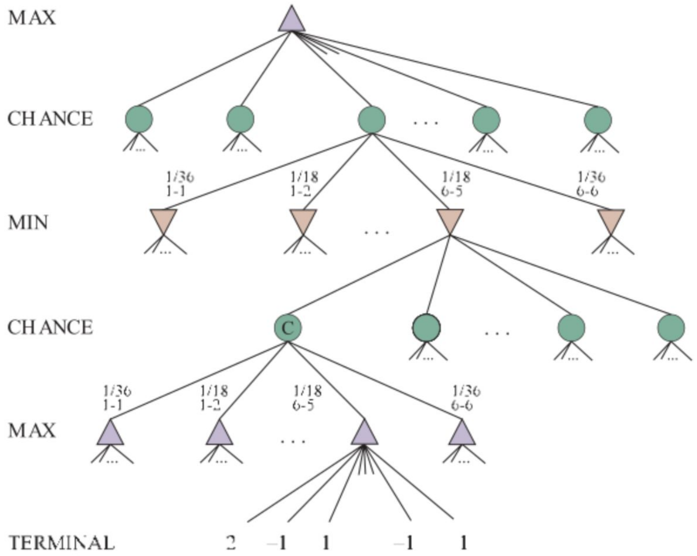
- Utility Function: Returns the exact value of a terminal state (e.g., win/loss/draw in chess).
- Evaluation Function (Eval): Estimates the "desirability" of a non-terminal position based on features like material advantage, board control, etc.
- Allows assessing states without reaching the end of the game.

Nondeterministic/Stochastic Games

In nondeterministic games, chance introduced by dice, card-shuffling

expectiminimax value = compute the expected value for change nodes

- value over all outcomes, weighted by probability of each chance action



Games of imperfect information

E.g., card games, where opponent’s initial cards are unknown

Typically we can calculate a probability for each possible deal

Seems just like having one big dice roll at the beginning of the game

Idea:

- compute the minimax value of each action in each deal
- choose the action with highest expected value over all deals
- special case: if an action is optimal for all deals, it’s optimal

Limitations of Game Search Algorithms

Alpha–beta search vulnerable to errors in the heuristic function.

Waste of computational time for deciding best move where it is obvious (meta-reasoning).

Reasoning done on individual moves. Humans reason on abstract levels.

Possibility to incorporate Machine Learning into game search process.

Homework

Read Chapter 6