Playing and Solving Games

Zero-sum games with perfect information

R&N 6

- Definitions
- Game evaluation
- Optimal solutions
  - Minimax
  - Alpha-beta pruning
- Approximations
  - Heuristic evaluation functions
  - Cutoffs
  - Endgames
- Non-deterministic games (first take)

Types of Games (informal)

	Deterministic	Chance
Perfect Information	Chess, Checkers Go	Backgammon, Monopoly
Imperfect Information	Battleship	Bridge, Poker, Scrabble, Nuclear war

Types of Games (informal)

	Deterministic	Chance
Perfect Information	Chess, Checkers Go	Backgammon, Monopoly
Imperfect Information	Battleship	Bridge, Poker, Scrabble, Nuclear war

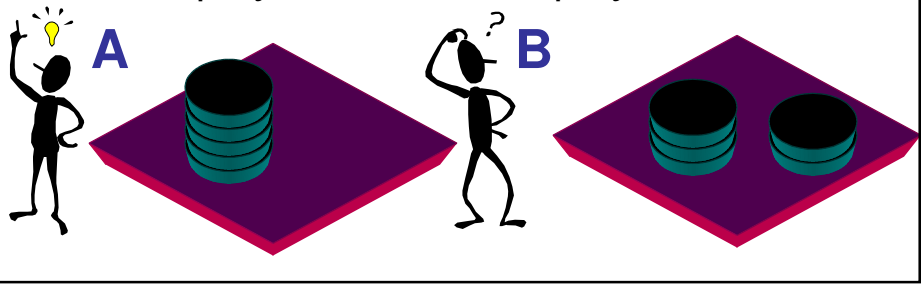
Note: This initial material uses the common definition of what a "game" is. More interesting is the generalization of the theory to scenarios that are far more useful to a wide range of decision making problems. Stay tuned….

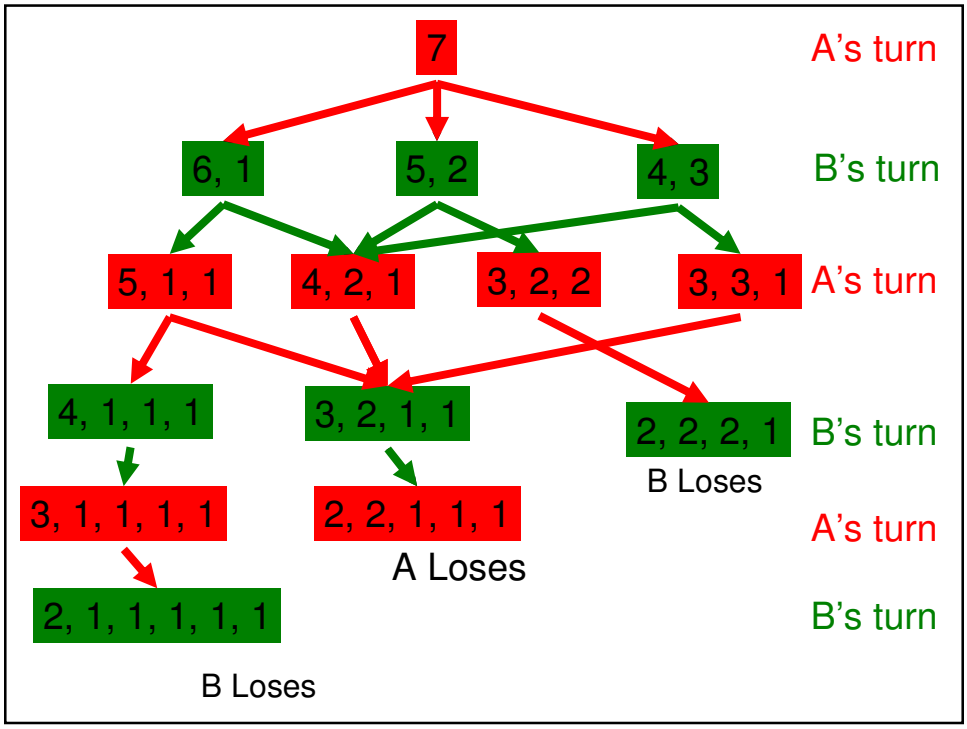
Definitions

- *Two-player game*: Player A and B. Player A starts.
- *Deterministic*: None of the moves/states are subject to chance (no random draws).
- *Perfect information*: Both players see all the states and decisions. Each decision is made *sequentially*.
- *Zero-sum*: Player's A gain is exactly equal to player B's loss. One of the player's must win or there is a draw (both gains are equal).

Example

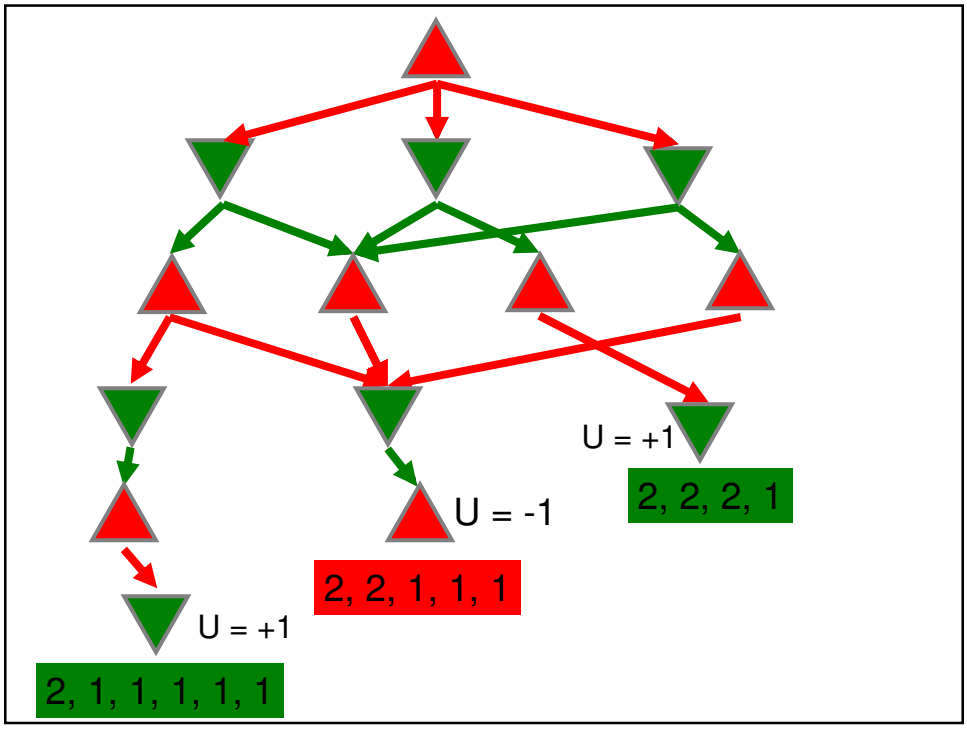
- Initially a stack of pennies stands between two players
- Each player divides one of the current stacks into two unequal stacks.
- The game ends when every stack contains one or two pennies
- The first player who cannot play loses





Search Problem

- *States*: Board configuration + next player to move
- *Successor*: List of states that can be reached from the current state through of legal moves
- *Terminal state*: States at which the games ends
- *Payoff/Utility*: Numerical value assigned to each terminal state. Example:
  - U(s) = +1 for A win, -1 for B win, 0 for draw
- *Game value:* The value of a terminal that will be reached assuming optimal strategies from both players (*minimax* value)
- *Search*: Find move that maximizes game value from current state



Optimal (minimax) Strategies

- Search the game tree such that:
  - A's turn to move → find the move that yields maximum payoff from the corresponding subtree → This is the move most favorable to A
  - B's turn to move → find the move that yields minimum payoff (best for B) from the corresponding subtree → This is the move most favorable to B

Minimax

Minimax (*s*)

If *s* is terminal

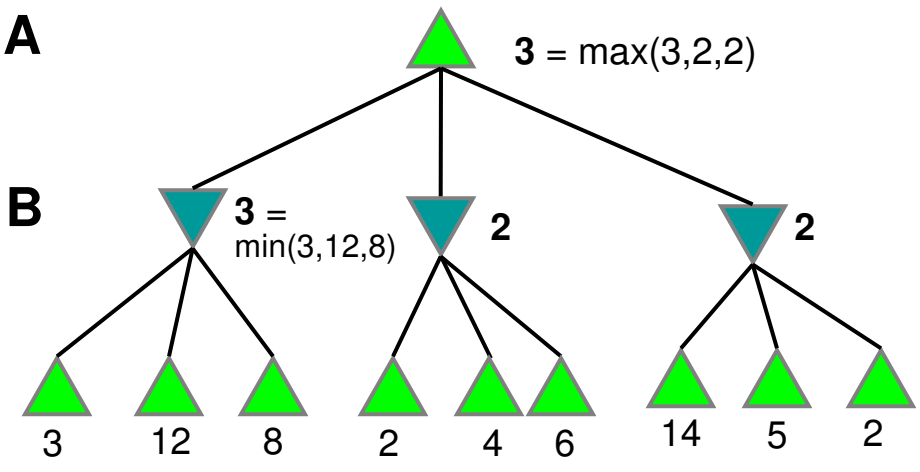
Return *U*(*s*)

If next move is *A*

Return $\max_{s' \in Succs(s)} Minimax(s')$

Else

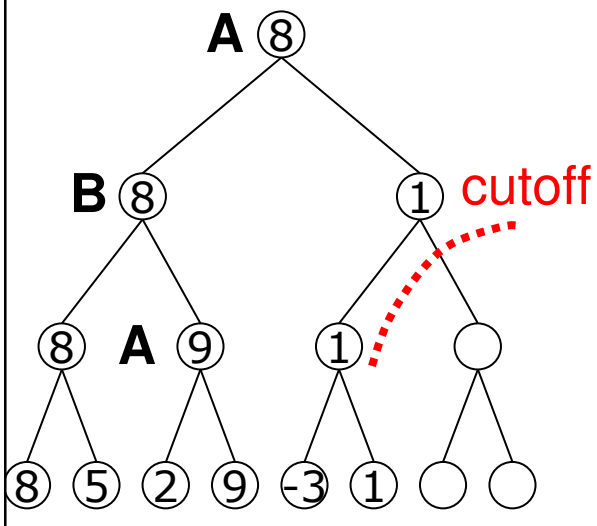
Return $\min_{s' \in Succs(s)} Minimax(s')$



Minimax Properties

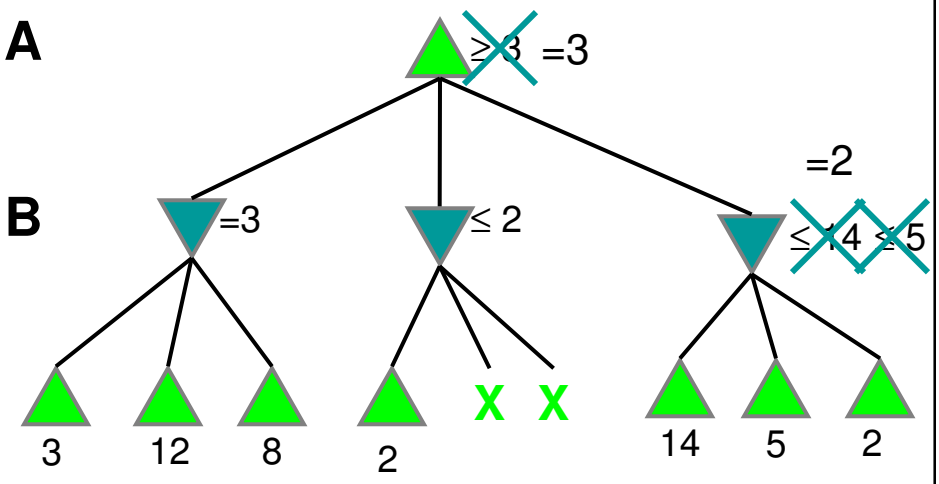
- *Complete:* If finite game
- *Optimal:* If opponent plays optimally
- *Complexity:* Essentially DFS, so:
  - Time: O(B^m)
  - Space: O(Bm)
  - B = number of possible moves from any state (branching factor)
  - m = depth of search (length of game)

Pruning



- The value at A move is 8 (so far)
- If A moves right, the value there is 1 (so far)
- B will *never increase* the value at this node; it will always be less than 8
- B can *ignore* the remaining nodes

Pruning



αβ Pruning

- Maintain:
  - α = Best value found so far at A nodes, including those at current node
  - β = Best value found so far at B nodes, including those at current node
- If at a B node: No need to expand this node any further if α >= β because there is no way that a descendant of the current node can yield a better value

```
Minimax (s,α,β)
    If s is terminal
        Return U(s)
    If A node
        For each s' in Succs(s)
            α = Max(α, Minimax(s',α,β)
            If (α >= β) Return β
        Return α
    If B node
        For each s' in Succs(s)
            β = Min(β, Minimax(s',α,β)
            If (α >= β) Return α
        Return β
```

```
Minimax (s,α,β)
    If s is terminal
        Return U(s)
    If A node
        For each s' in Succs(s)
            α = Max(α, Minimax(s',α,β)
            If (α >= β) Return β
        Return α
    If B node
        For each s' in Succs(s)
            β = Min(β, Minimax(s',α,β)
            If (α >= β) Return α
        Return β
```

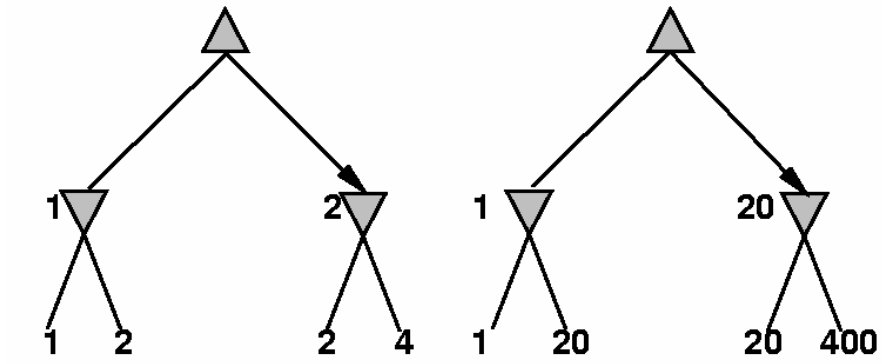
Same as default minimax

Prune if no better solution can be found

Properties

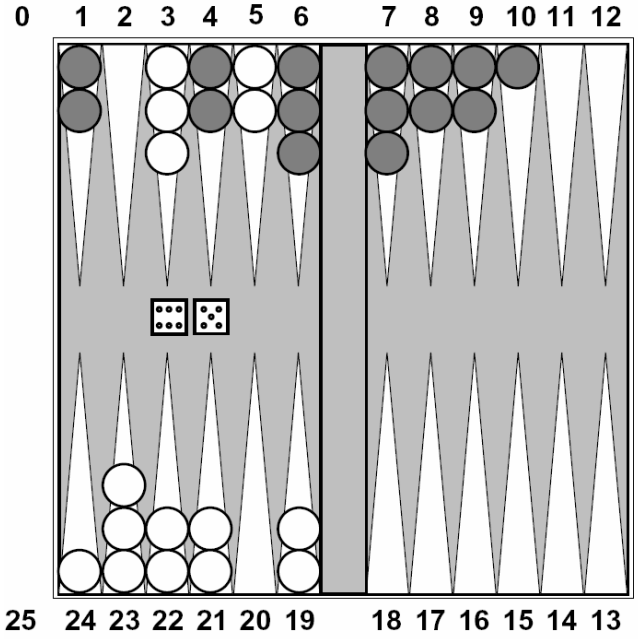
- Guaranteed to find same solution
- O($B^{m/2}$) with proper ordering of the nodes → At "A" node, the successor are in order from high to low score
- Use heuristic evaluation functions to cut off search early
- Example: Weighted sum of number of pieces (material value of state)
- Stop search based on cutoff test (e.g., maximum depth)
- Iterative deepening search to limit DFS
- Solve by brute-force dynamic programming when the number of states is small

Choice of Value?

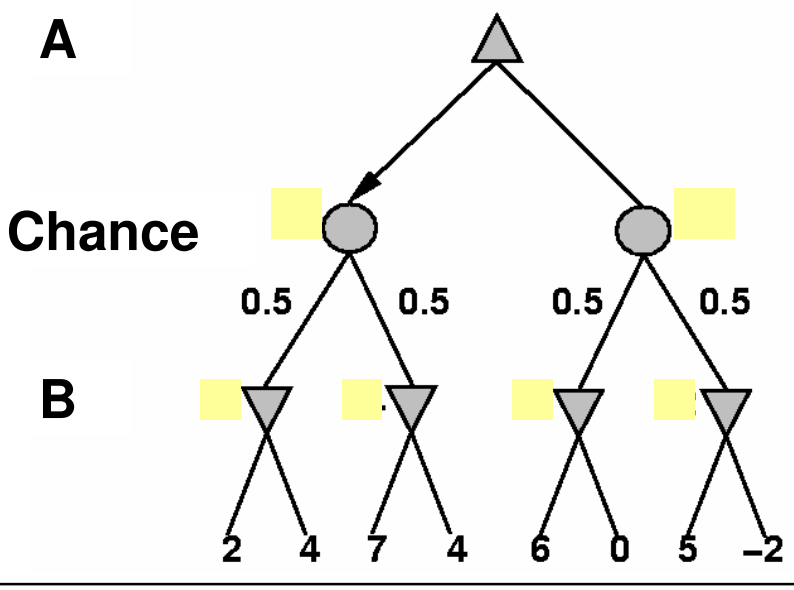


- Absolute game value is different in the two cases
- Minimax solution is the same
- Only the relative ordering of values matters, not the absolute values → *ordinal utility values*
- True only for *deterministic* games
- Evaluation functions can be any function that preserves the ordering of the utility values

Non-Deterministic Games



Non-Deterministic Games

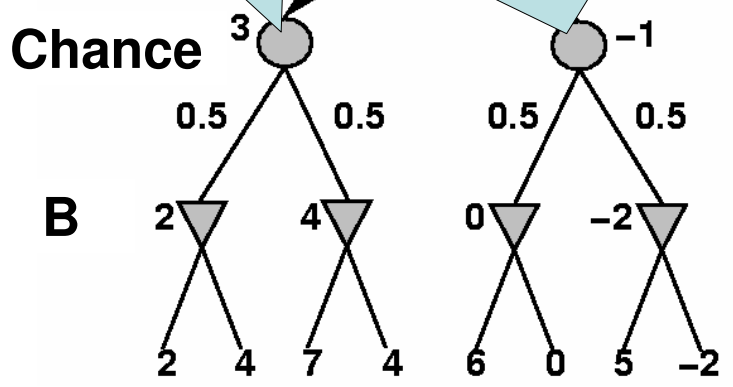


Non-Deterministic Games

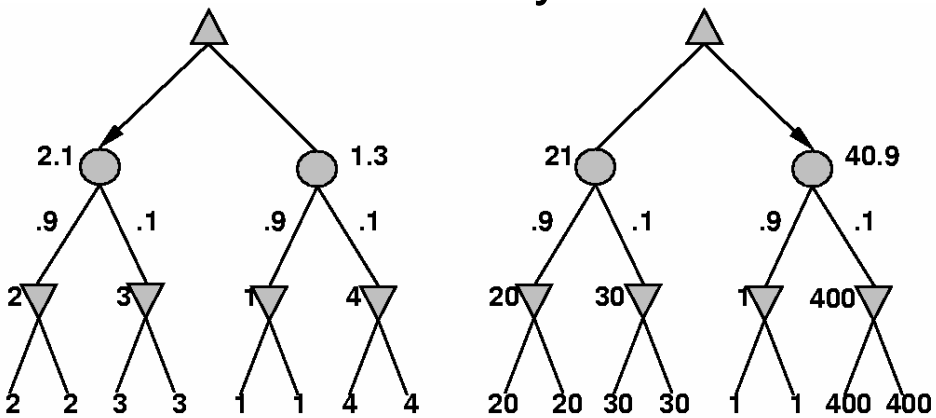
Includes states where neither player makes a choice. A random decision is made (e.g., rolling dice)

Use expected value of successors at chance nodes:

$$\sum_{s' \in Succs(s)} p(s')MiniMax(s')$$



Choice of Utility Values



- Different utility values may yield radically different result even though the order is the same → Absolute utility values do matter
- Utility should be proportional to actual payoff, it is not sufficient to follow the same order
- Think of choosing between 2 lotteries with same odds but radically different payoff distributions
- Implication: Evaluation functions must be *linear positive* functions of utility
- Kind of obvious but important consideration for later developments

Non-Deterministic Minimax

Minimax (*s*)

If *s* is terminal

Return *U*(*s*)

If next move is *A:* Return $\max_{s' \in Succs(s)} Minimax(s')$

If next move is *B* Return $\min_{s' \in Succs(s)} Minimax(s')$

If chance node Return $\sum_{s' \in Succs(s)} p(s') Minimax(s')$

Properties

- α-β pruning can be extended provided that the utility values are bounded → We don't need to evaluate all the children of a chance node to bound the average
- Less effective
- Different outcomes depending on exact values of utility, not just ordering

- Definitions
- Game evaluation
- Optimal solutions
  - Minimax
  - Alpha-beta pruning
- Approximations
  - Heuristic evaluation functions
  - Cutoffs
  - Endgames
- Non-deterministic games