CS 161
Discussion 5
Local Search & Adversarial Search

Game Search

Games:

- Require making some decision when calculating the optimal decision is infeasible

How to choose a good move when time is limited?

Game Search

- Pruning
  - Ignore portions of the search tree that make no difference to the final choice
- Evaluation functions
  - approximate the true utility of a state without doing a complete search

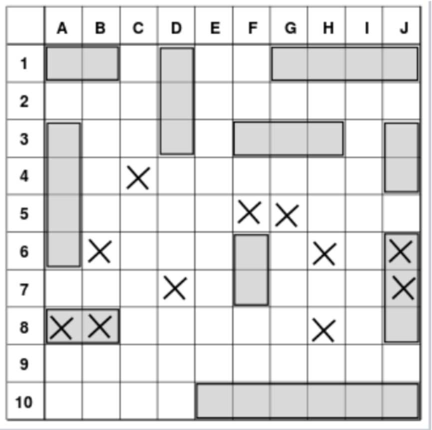
Types of Games

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

Types of Games

Go: Perfect and Deterministic

Monopoly: Perfect, Chance Introduced



Battleship: Imperfect and Deterministic

Bridge: Imperfect, Chance Introduced

Games with Two Players

- S_0: The **initial state**, which specifies how the game is set up at the start.
- $\textsc{Player}(s)$: Defines which player has the move in a state.
- $\textsc{Actions}(s)$: Returns the set of legal moves in a state.
- $\textsc{Result}(s, a)$: The **transition model**, which defines the result of a move.
- $\textsc{Terminal-Test}(s)$: A **terminal test**, which is true when the game is over and false otherwise. States where the game has ended are called **terminal states**.
- $\textsc{Utility}(s, p)$: A **utility function** (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p. In chess, the outcome is a win, loss, or draw, with values $+1$, 0, or $\frac{1}{2}$. Some games have a wider variety of possible outcomes; the payoffs in backgammon range from 0 to $+192$. A **zero-sum game** is (confusingly) defined as one where the total payoff to all players is the same for every instance of the game. Chess is zero-sum because every game has payoff of either $0 + 1$, $1 + 0$ or $\frac{1}{2} + \frac{1}{2}$. "Constant-sum" would have been a better term, but zero-sum is traditional and makes sense if you imagine each player is charged an entry fee of $\frac{1}{2}$.

Optimal Decisions

What is an optimal solution in adversarial search?

- Normal search:
    - a sequence of actions leading to a goal state
- Adversarial search:
    - Find a contingent strategy
        - First move, moves in the states resulting from the other guy's possible moves, ...

Optimal Decisions - MINIMAX

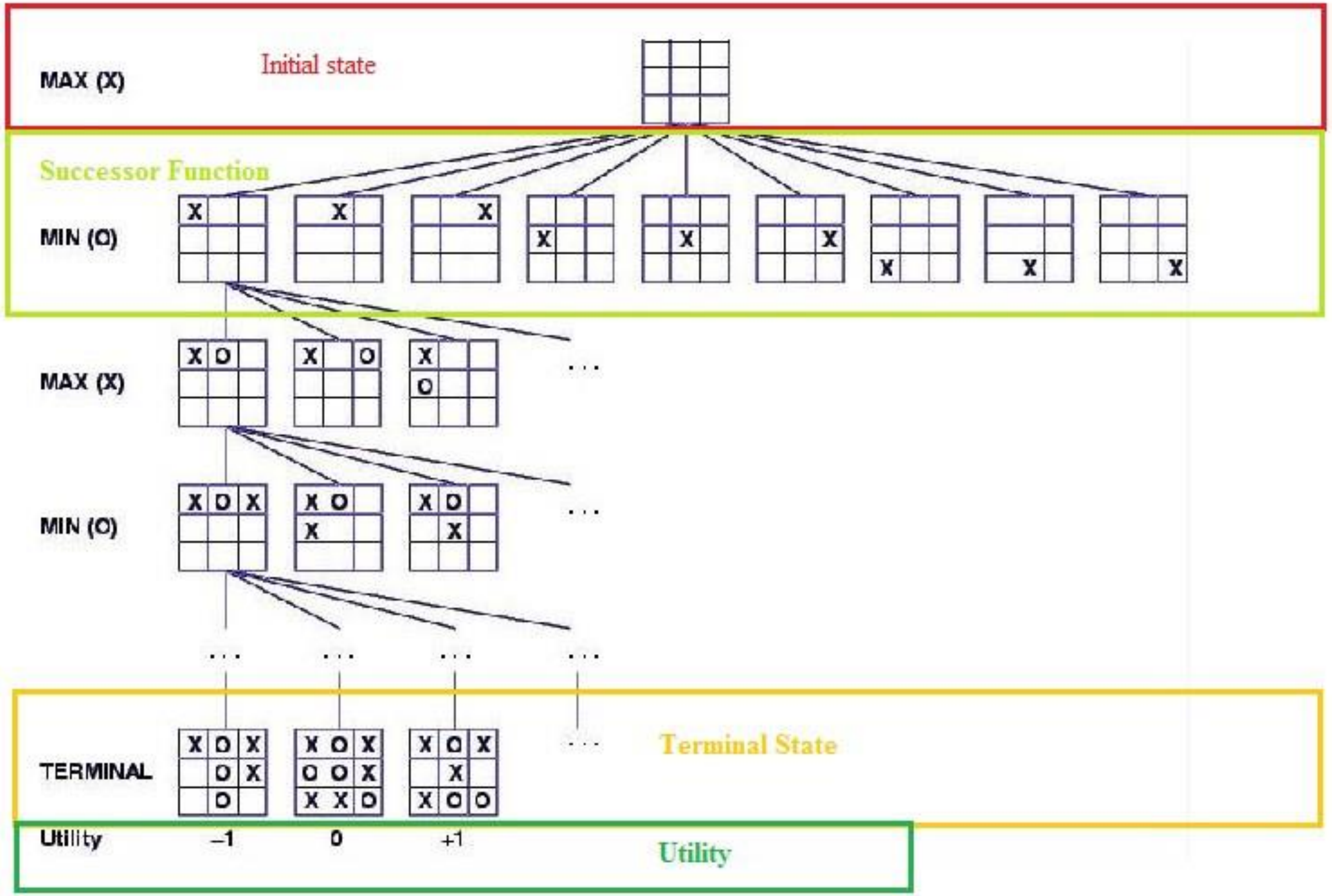
Given a game tree, how to determine the optimal strategy?

MINIMAX(n)

- The utility (for MAX)
- Assume both players play *optimally* from there to end of game
  - Given a choice, MAX prefers to move to a state of maximum value, whereas MIN prefers a state of minimum value.

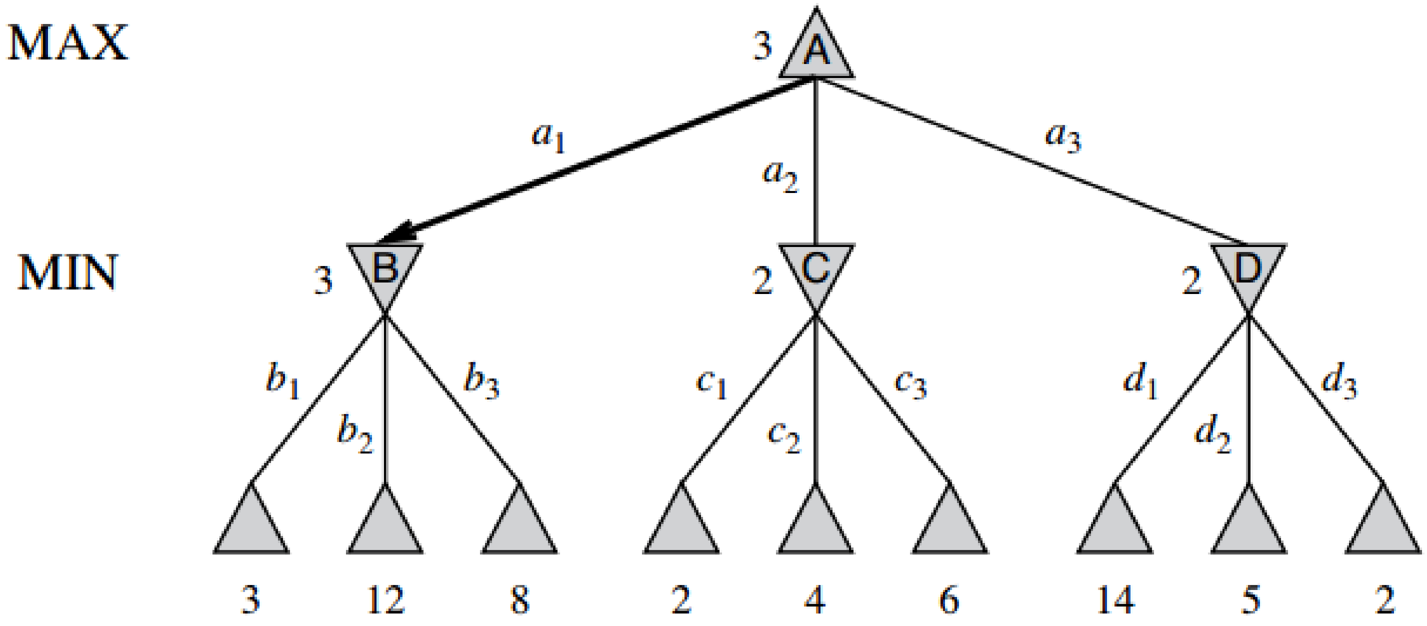
$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

Tic-Tac-Toe



MAX (X)
Initial state
Successor Function
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Terminal State
Utility
–1
0
+1
Utility

Optimal Decisions



Complexity of minimax

- b: number of legal moves at each point
- m: depth of the tree
- Depth-first search:
  - time complexity: $O(b^m)$
  - space complexity: linear in m and b

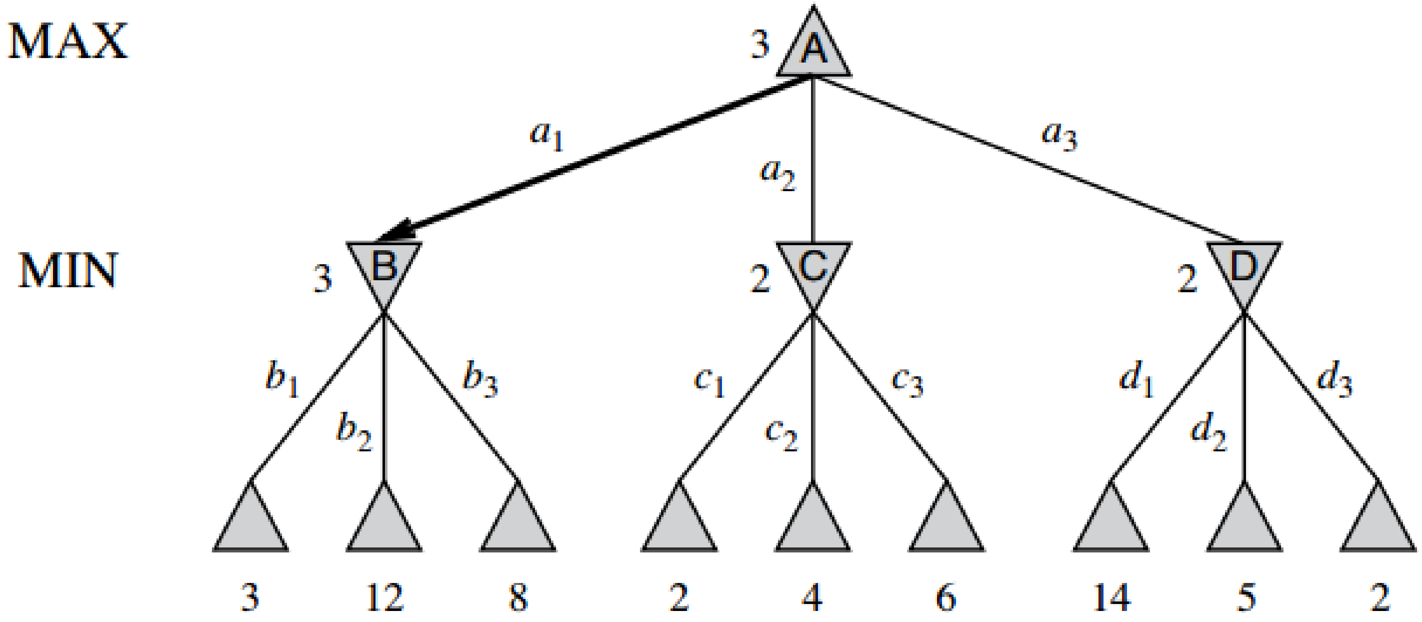
Alpha-beta pruning

- Minimax: a way of finding an optimal move in a two player game.
- Alpha-beta pruning: finding the optimal minimax solution while avoiding searching subtrees of moves which won't be selected.

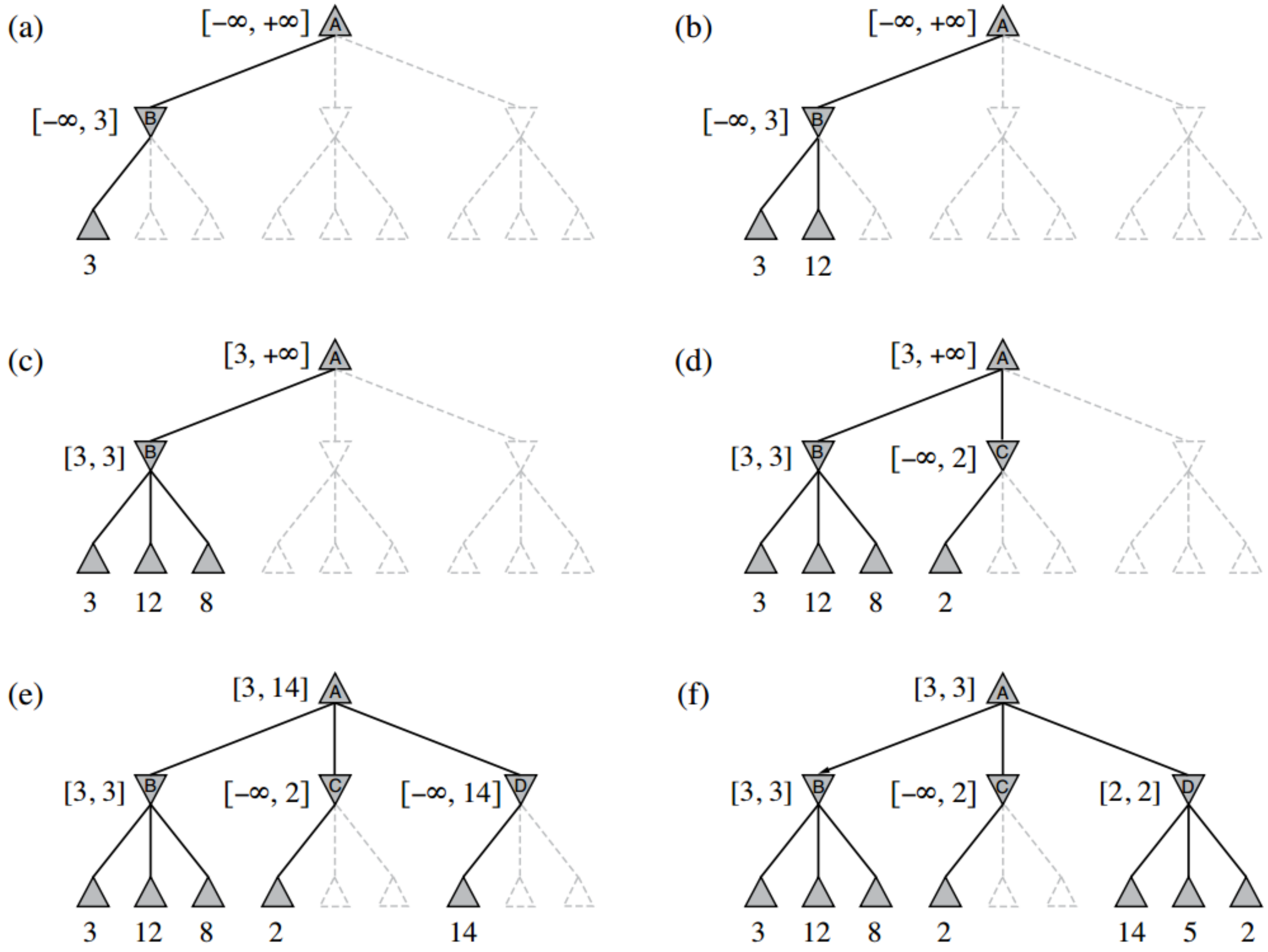
- Alpha: maximum lower bound of possible solutions
- Beta: minimum upper bound of possible solutions

$$\alpha \leq N \leq \beta$$

Example



Alpha-beta pruning



Alpha-beta pruning

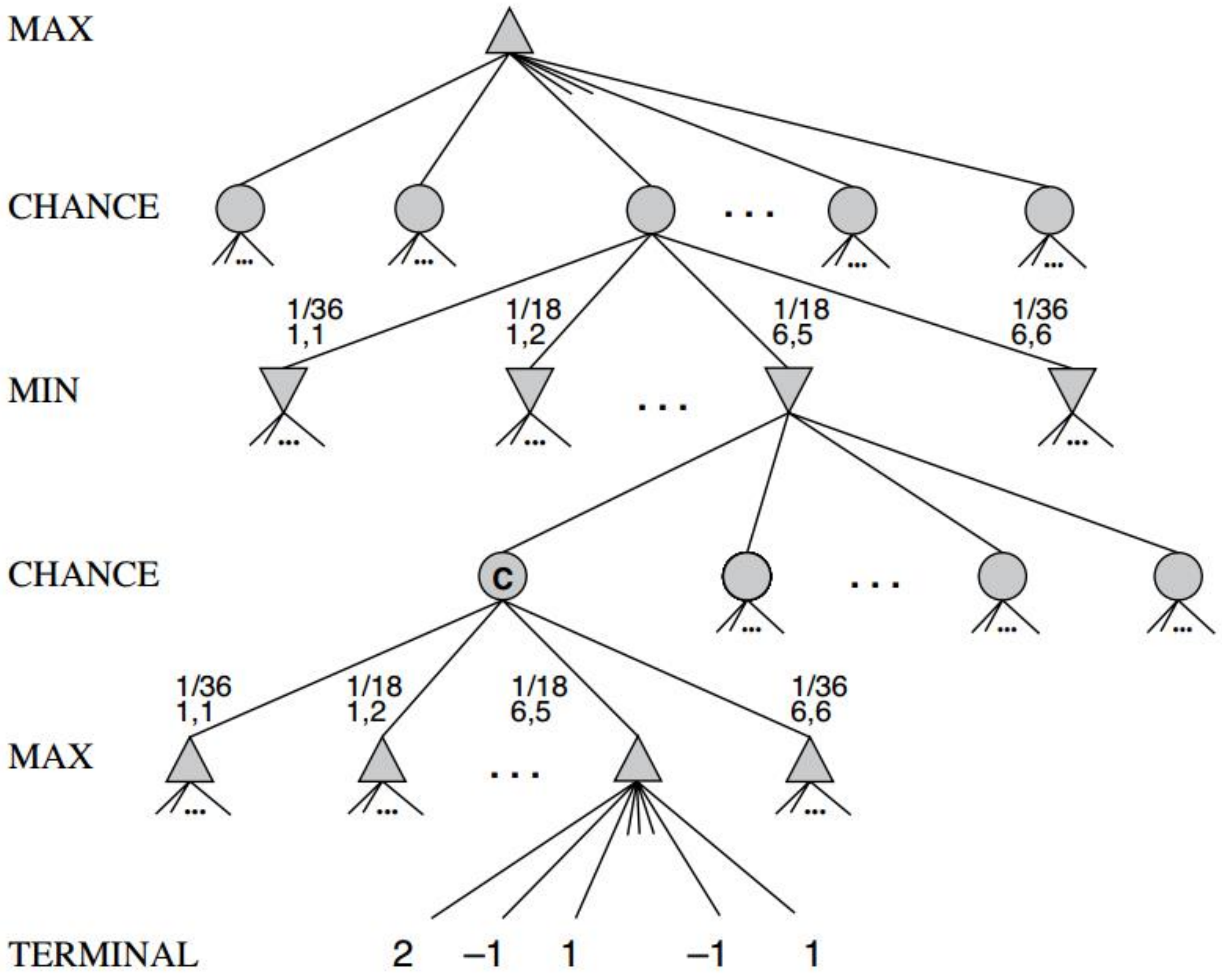
Online Demo:

http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html

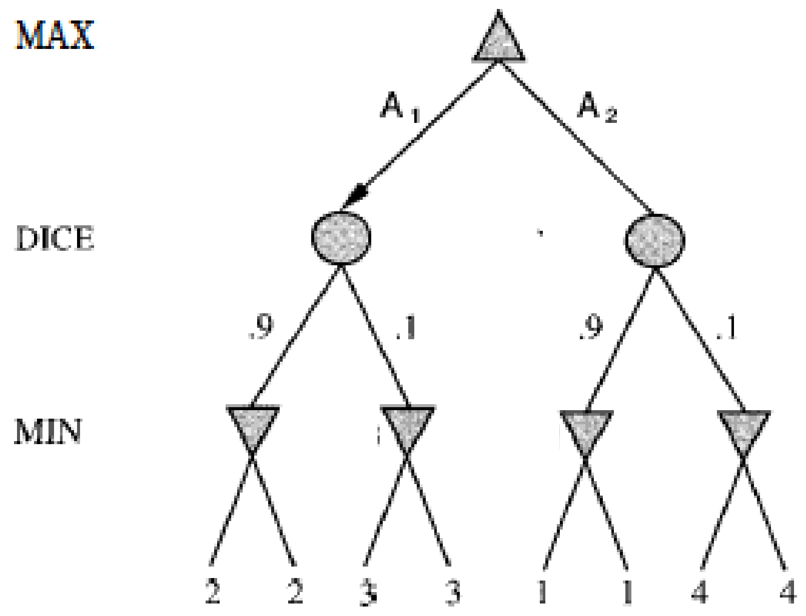
EXPECT MINIMAX

$$
\textsc{Expectiminimax}(s) =
\begin{cases}
\textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\
\max_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\
\min_a \textsc{Expectiminimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \\
\sum_r P(r)\textsc{Expectiminimax}(\textsc{Result}(s,r)) & \text{if } \textsc{Player}(s) = \textsc{chance}
\end{cases}
$$

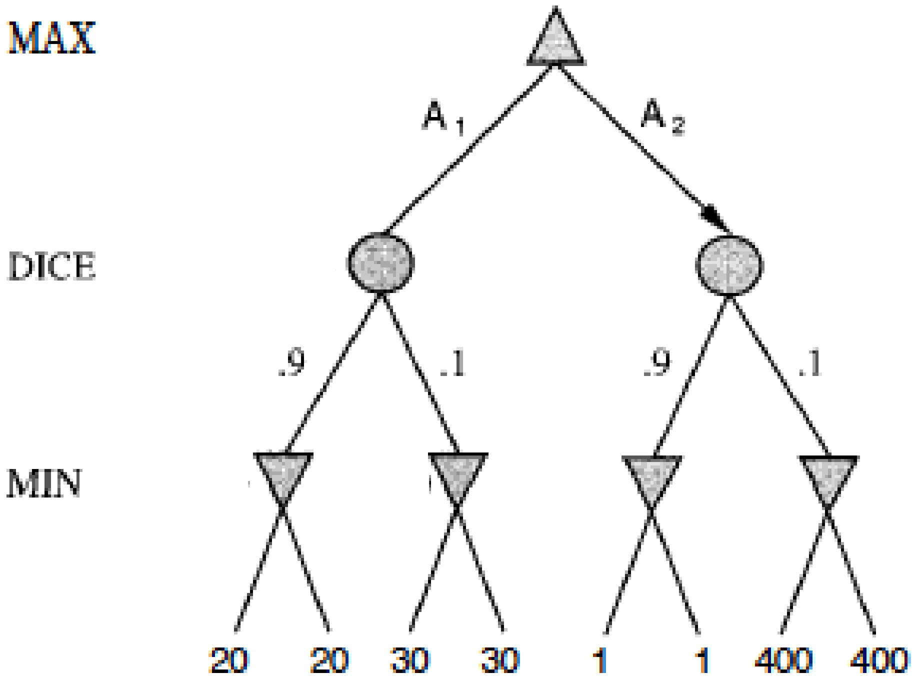
EXPECT MINIMAX



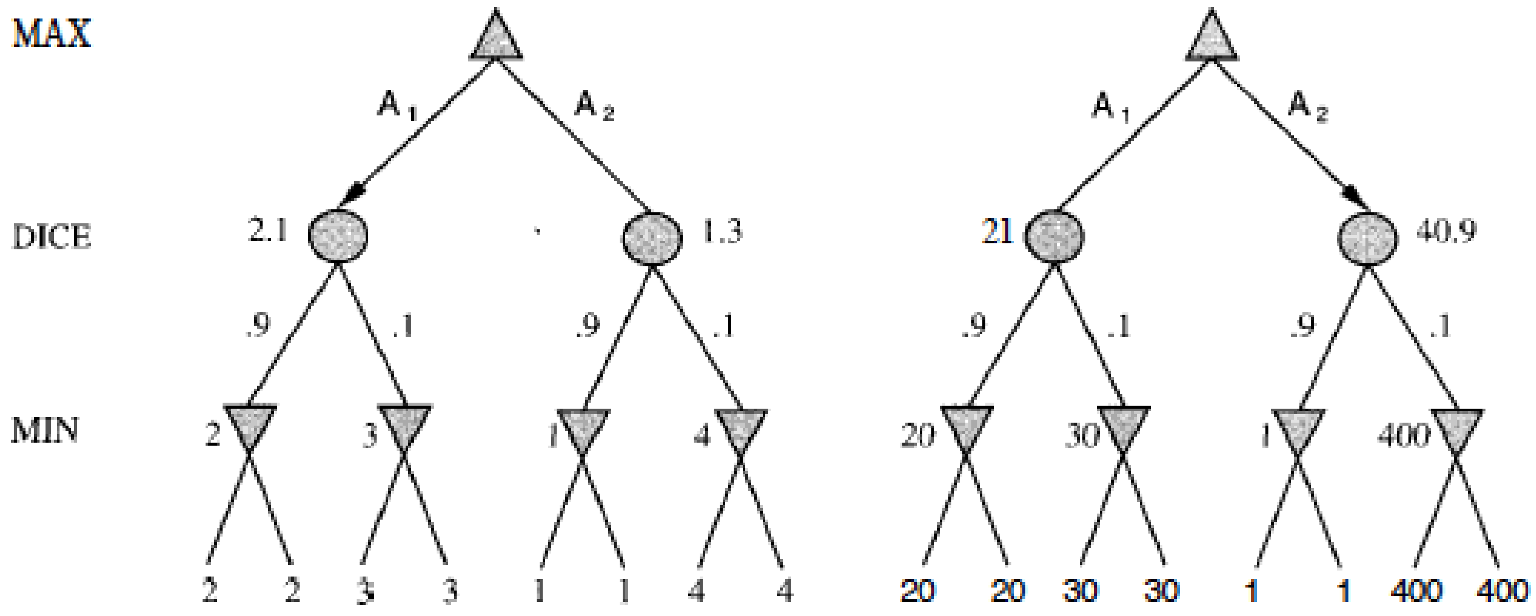
Example



Example



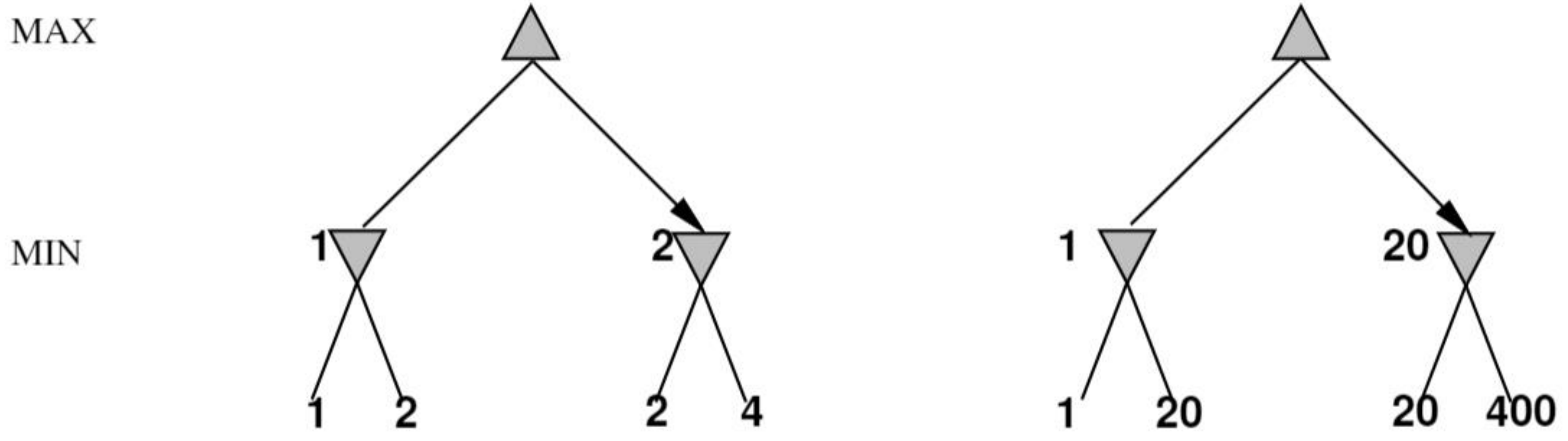
Example



MAX
DICE
MIN
A1
A2
2.1
1.3
.9
.1
.9
.1
2
3
1
4
2 2 3 3 1 1 4 4
A1
A2
21
40.9
.9
.1
.9
.1
20
30
1
400
20 20 30 30 1 1 400 400

Transformation

In deterministic games, exact values do not matter.



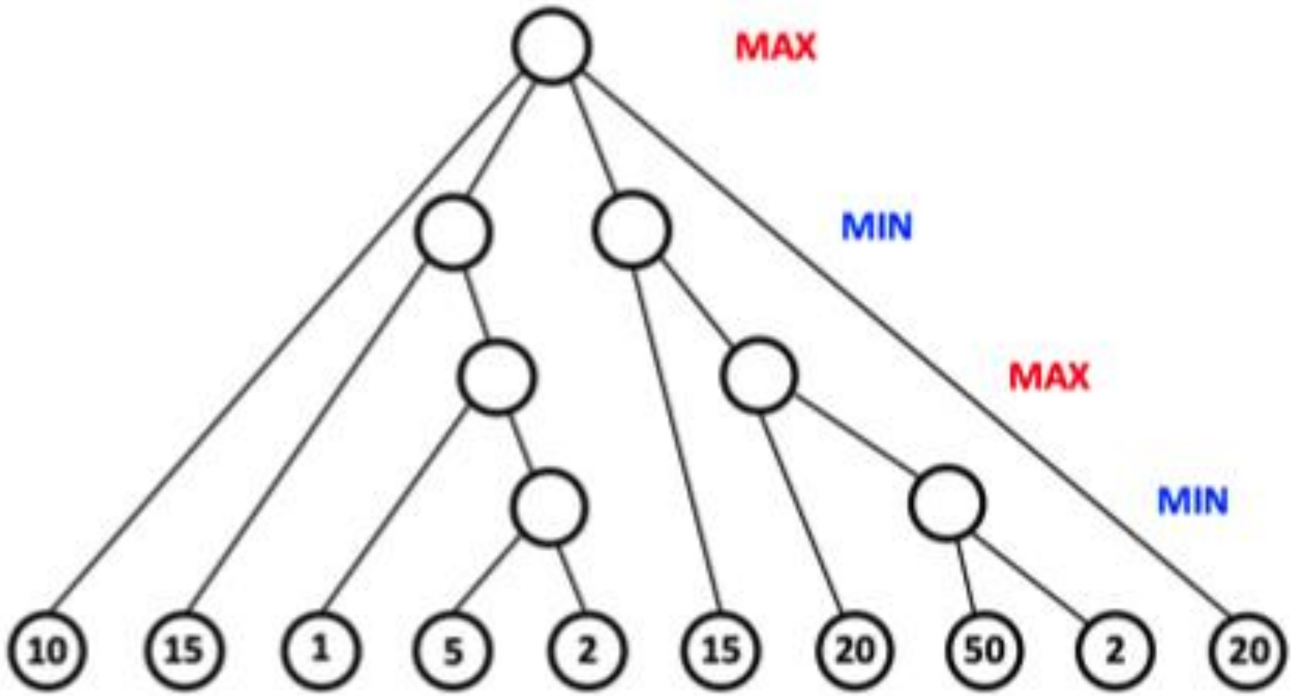
Behaviour is preserved under any **monotonic** transformation of the original EVAL

In non-deterministic games, exact values matter!!!

Complexity of expect-minimax

- b: number of legal moves at each point
- m: depth of the tree
- n: number of distinct rolls
- complexity: $O(b^m n^m)$

Local Search



Classical Search

- Deterministic
  - A state has fixed successor
- Designed to explore the search space systematically
- Solution: a sequence of actions
  - path from initial state to goal state

However

- Sometimes the path to the goal is not our focus. Only final configuration matters
  - n-queens, circuit design, etc.
- Sometimes the state space is continuous
  - $x^2 + y^2 = 10, x \in \mathbb{R}, y \in \mathbb{R}$

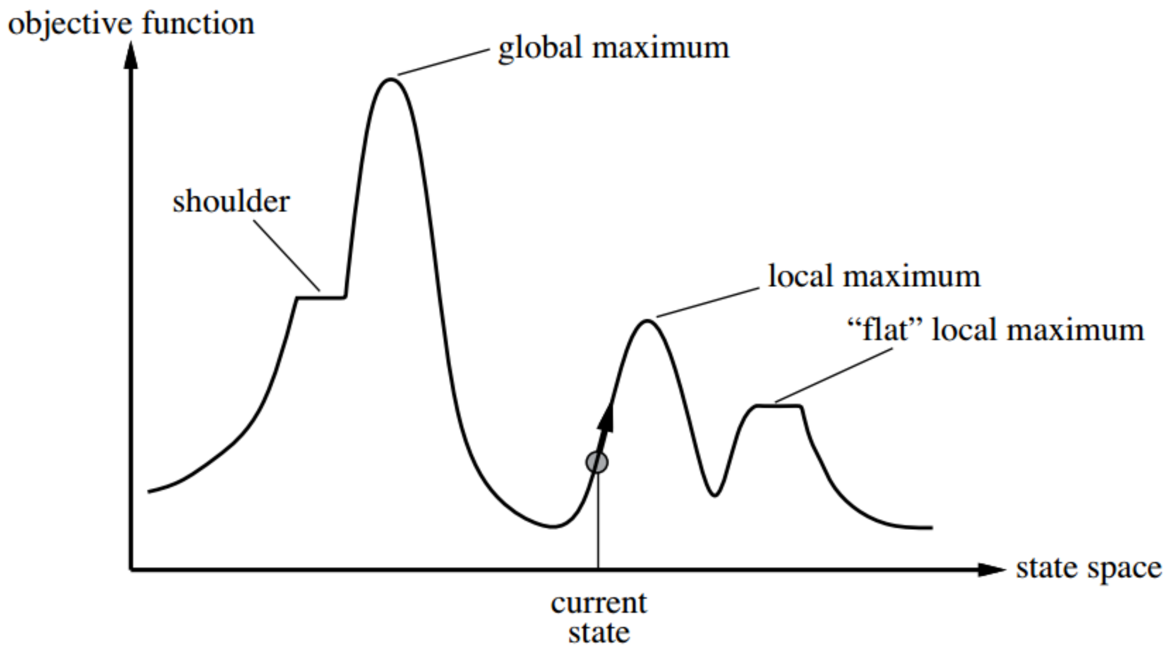
Local Search

- Keep track of a single node (current state)
- Move to neighbors
- No need to keep paths

Advantages:

- Little memory
- Find reasonable solution in large or infinite state spaces
  - Good for **optimization problems** (Find best state according to an **objective function**)

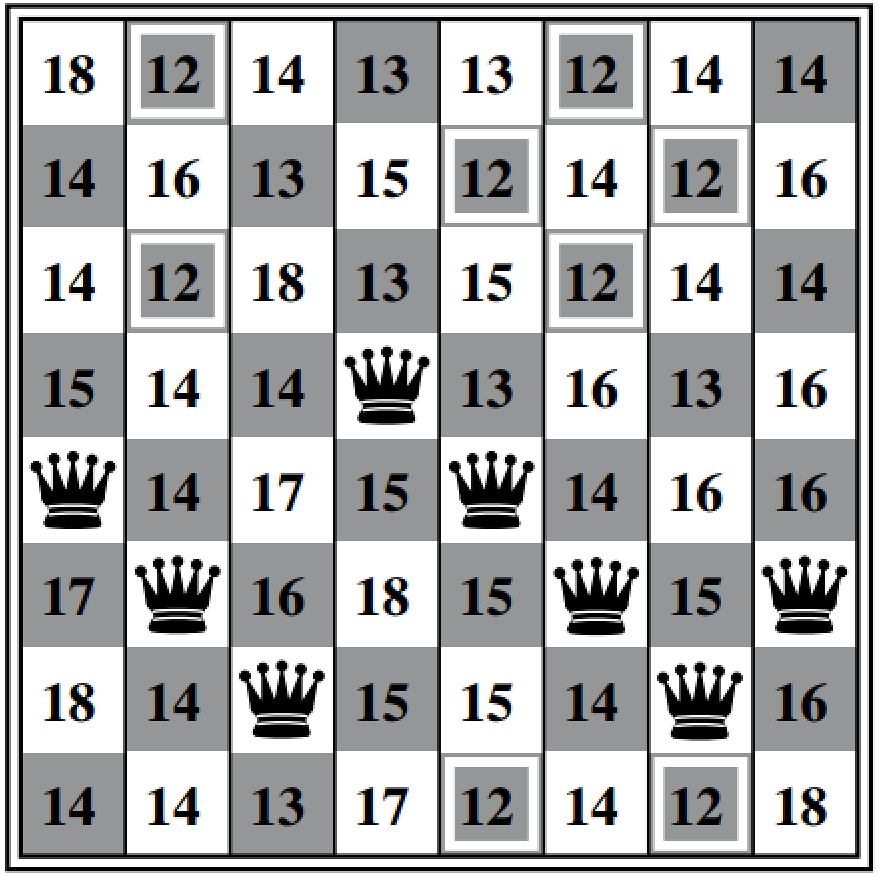
Example - State-Space Landscape



Hill-climbing search

- Greedy local search
  - Grabs a good neighbor state without thinking ahead about where to go next.
- Check all neighbors of current state
- choose the one with the highest value (lowest cost)
- Terminate when no neighbor has a higher value

Example – 8-queens

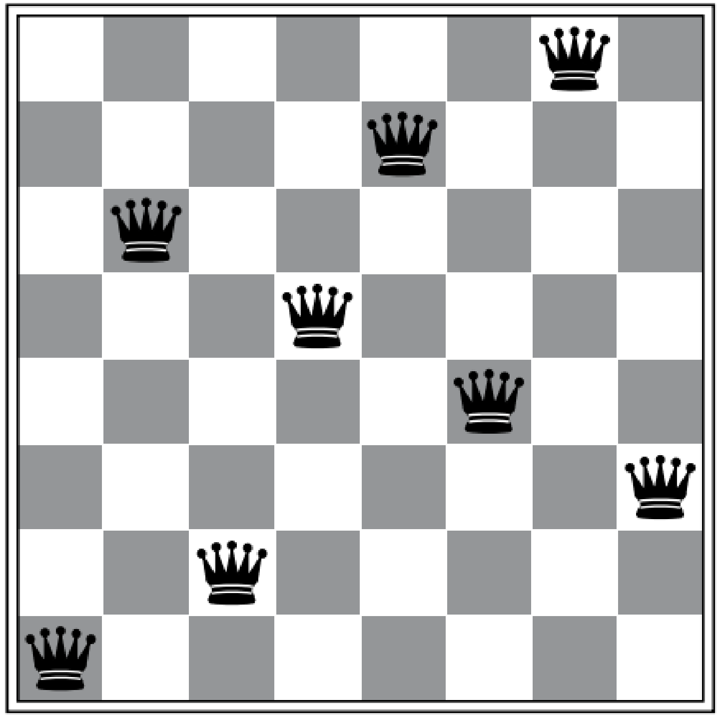


Hill-climbing search

- Advantage
  - Easy to improve a bad state (rapid progress)
- Disadvantage
  - Get stuck in
    - local optimal
    - Ridges
    - Plateau

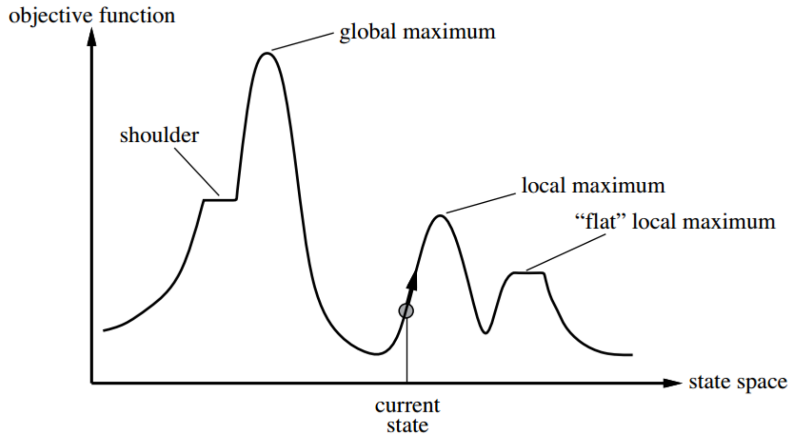
Example – 8-queens

The state has h = 1 but every successor has a higher cost.



Hill-climbing search

- Disadvantage
  - Get stuck in
    - local optimal
    - Ridges
    - Plateau



The success of hill climbing depends very much on the shape of the state-space landscape!

NP-hard problems typically have an exponential number of local maxima to get stuck on.

Simulated Annealing

- Hill-climbing algorithms never move towards state with lower value
  - May result in local optimal

Simulated Annealing:

- Randomly select candidate successor
- Go there if better
- Else go there with probability as function of “energy” and “temperature” (Start with high temperature and low temperature in the end, i.e. accept bad moves more likely in beginning)

Local Beam Search and Stochastic Beam Search

Local Beam Search

- Start with k states
- Keep the best k successors

- Advantage: useful information is passed among the parallel search
- Disadvantage: lack of diversity
  - What if all the k successors get stuck at the same local optimal?

Stochastic Beam Search

(Introducing randomness)

- Randomly select k successors
- Better successor has a higher chance to be selected

Genetic Algorithms