ARTIFICIAL INTELLIGENCE

RECAP

Problem Solving Agent

- Uninformed Search
- Informed Search
- Local Search

All search schemes we covered so far were designed for single agent

- decisions of the agents determine the outcome

ADVERSARIAL SEARCH

Multiagent Environment: Competitive, Cooperative

Competitive environments -> adversarial search problems also called games.

Assumption(s):

- Environments is fully observable and deterministic.
- Two agents act alternately in which the utility values at the end of the game are always equal and opposite e.g., chess.

SEARCH VERSUS GAMES

- Search: no adversary
  - Solution is (heuristic) method for finding goal
  - Evaluation function: estimate cost from start to goal through a given node
  - Examples: path planning, scheduling activities, …

- Games: adversary
  - Solution is a strategy
    - Specifies move for every possible opponent reply
  - Time limits force an approximate solution
  - Evaluation function: evaluate "goodness" of game position
  - Examples: chess, checkers, Othello, backgammon

GAME FORMULATION

Consider two players, MAX and MIN

S_0 : Initial state of the game

Player(s): Which player has to move in a state, s

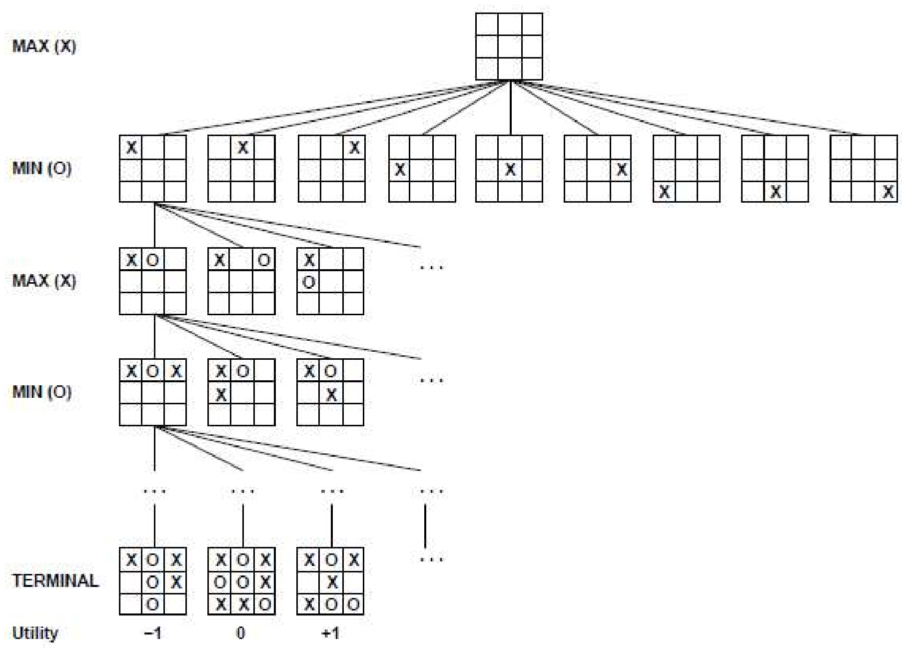
Action(s): Returns the set of legal moves in a state, s

RESULT(s, a): The **transition model**, which defines the result of a move, i.e., the resulting state when action a is applied on a state s.

TERMINAL-TEST(s): A **terminal test**, which is true when the game is over and false otherwise.

UTILITY(s, p): A **utility function** (objective function/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 0.5 .

GAME TREE (2-PLAYER, DETERMINISTIC, TURNS)



GAMES AS SEARCH

- Two players, “MAX” and “MIN”
- MAX moves first, & take turns until game is over
  - Winner gets reward, loser gets penalty
  - “Zero sum”: sum of reward and penalty is constant
- MAX uses search tree to determine “best” next move

- Formal definition as a search problem:
  - Initial state: set-up defined by rules, e.g., initial board for chess
  - Player(s): which player has the move in state s
  - Actions(s): set of legal moves in a state
  - Results(s,a): transition model defines result of a move
  - Terminal-Test(s): true if the game is finished; false otherwise
  - Utility(s,p): the numerical value of terminal state s for player p
    - E.g., win (+1), lose (-1), and draw (0) in tic-tac-toe
    - E.g., win (+1), lose (0), and draw (1/2) in chess

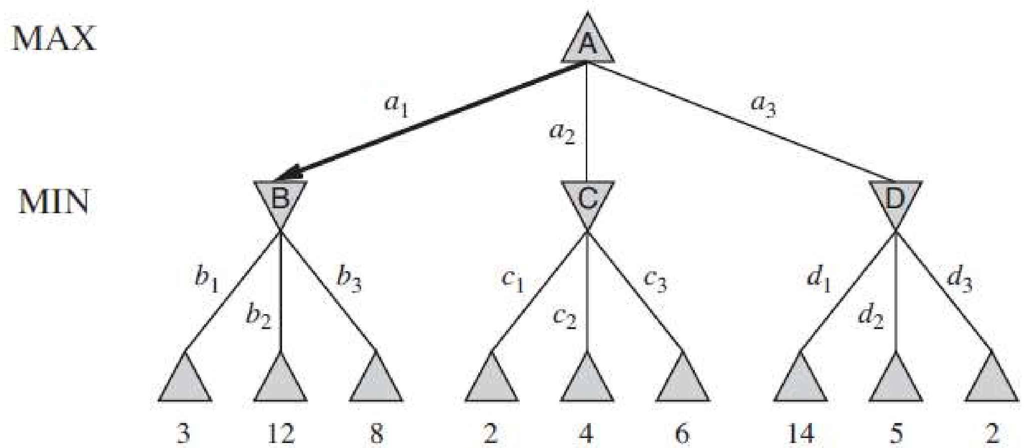
MIN-MAX: AN OPTIMAL PROCEDURE

- Designed to find the optimal strategy & best move for MAX:

  1. Generate the whole game tree to leaves
  2. Apply utility (payoff) function to leaves
  3. Back-up values from leaves toward the root:
     - a Max node computes the max of its child values
     - a Min node computes the min of its child values
  4. At root: choose move leading to the child of highest value

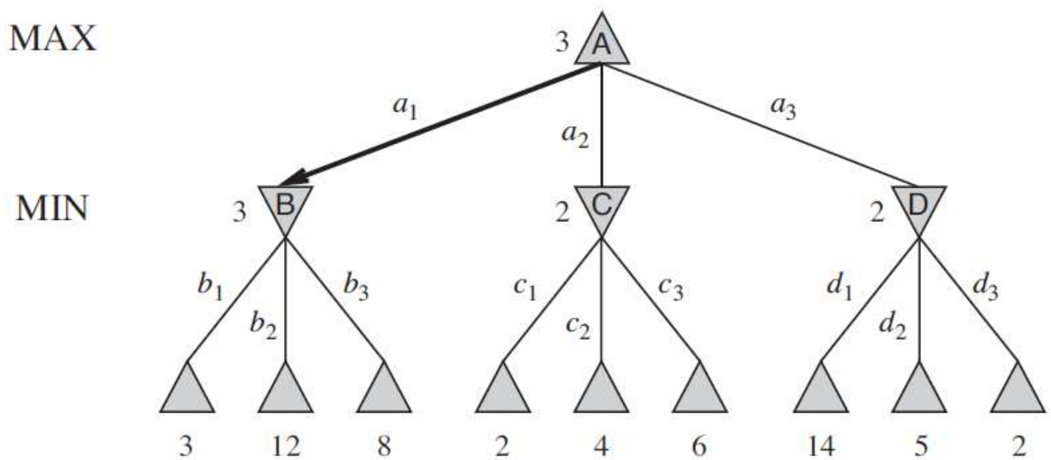
OPTIMAL SOLUTION IN GAMES?

We will devise a strategy from Max's perspective, assuming that Min will play optimally!



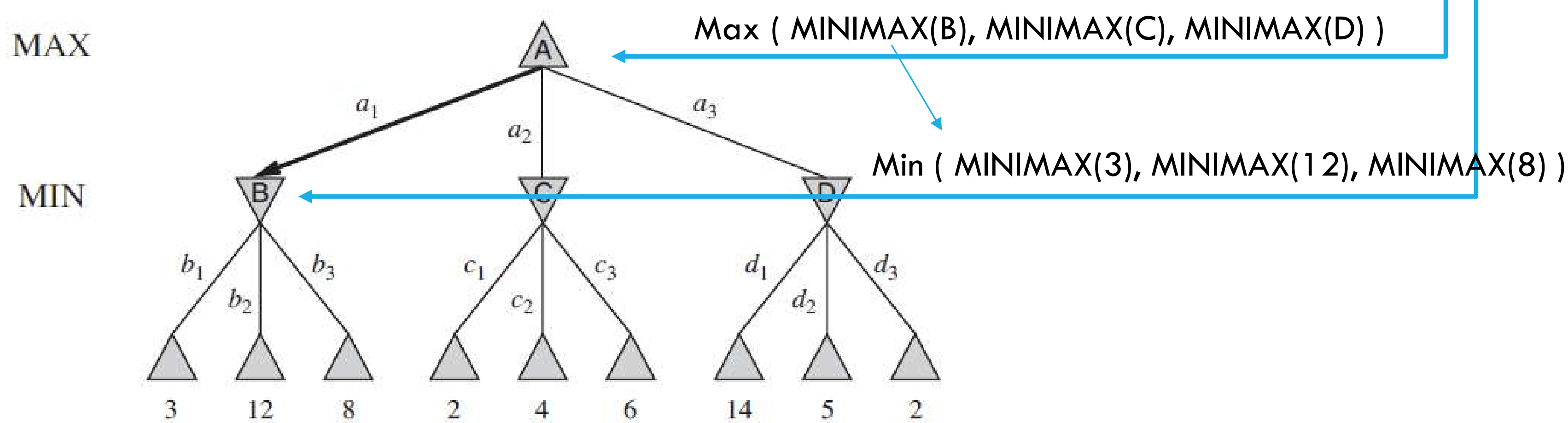
OPTIMAL SOLUTION IN GAMES?

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

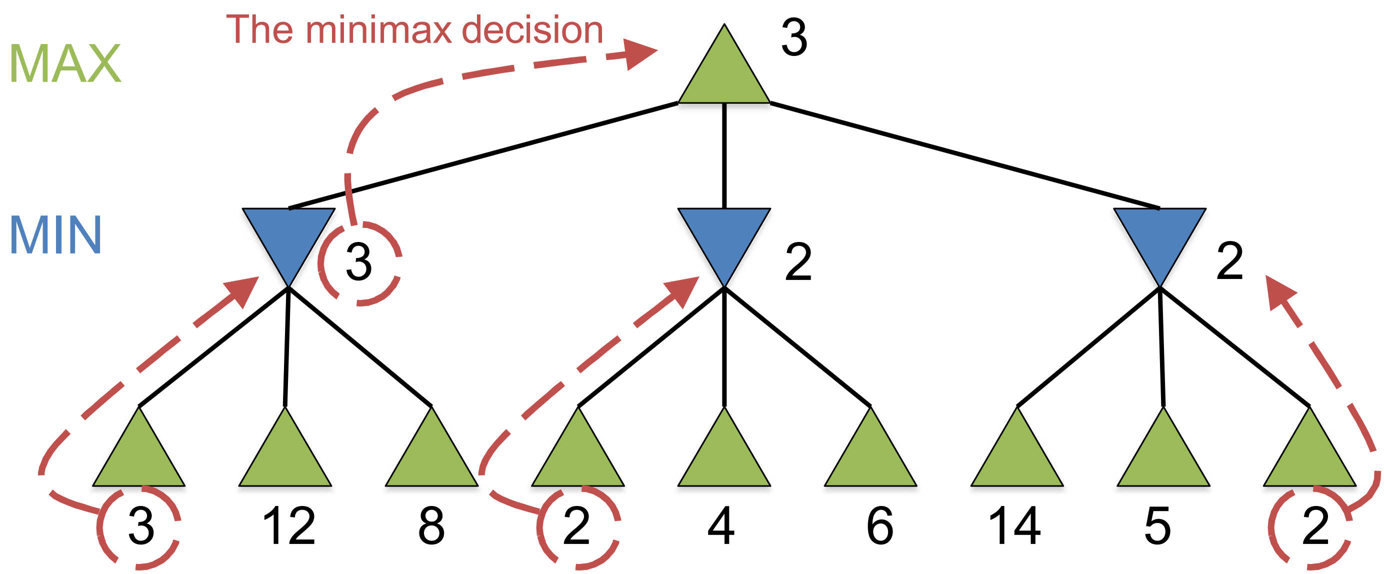


OPTIMAL SOLUTION IN GAMES?

$$
\text{MINIMAX}(s) =
\begin{cases}
\text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\
\max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\
\min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN}
\end{cases}
$$



TWO-PLY GAME TREE



Minimax maximizes the utility of the worst-case outcome for MAX

RECURSIVE MIN-MAX SEARCH

```
minMaxSearch(state)
  return argmax( [ minValue( apply(state,a) ) for each action a ] )
```

Simple stub to call recursion f'ns

```
maxValue(state)
  if (terminal(state)) return utility(state);
  v = -infty
  for each action a:
    v = max( v, minValue( apply(state,a) ) )
 return v
```

If recursion limit reached, eval position

Otherwise, find our best child:

```
minValue(state)
  if (terminal(state)) return utility(state);
  v = infty
  for each action a:
    v = min( v, maxValue( apply(state,a) ) )
 return v
```

If recursion limit reached, eval position

Otherwise, find the worst child:

GAMES ARE HARD TO SOLVE!

Chess has average branching factor b = 35

A single game requires an average of 50 moves per player leading to 35^{100} states.

Need to do some action even when calculating the *optimal* decision is infeasible!

OPTIMAL MOVE?

How to choose a good move when time is limited?

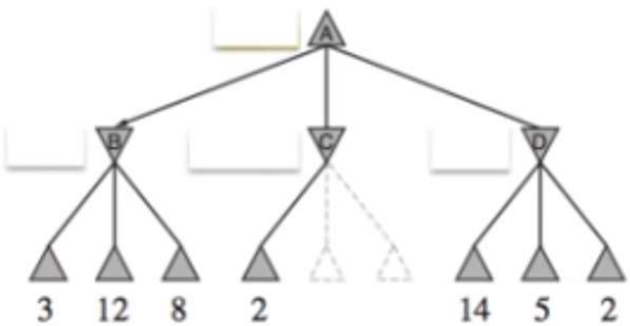
Pruning allows us to ignore portions of the search tree that make no difference to the final choice

Heuristic **evaluation functions** allow us to approximate the true utility of a state without doing a complete search

ALPHA-BETA PRUNING

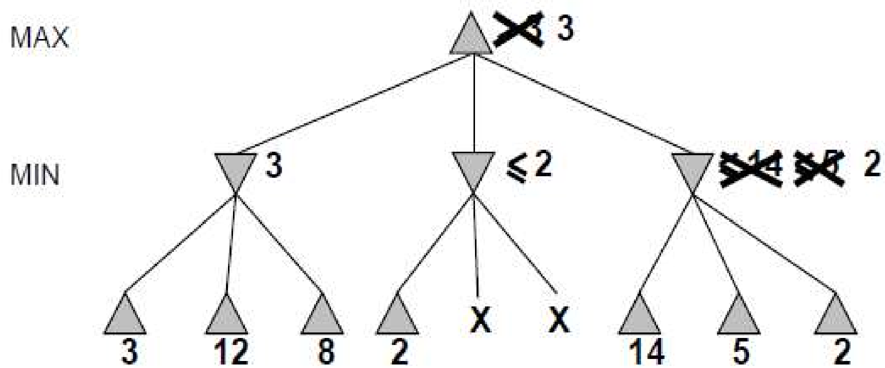
- Exploit the “fact” of an adversary
- If a position is provably bad
  - It’s no use searching to find out just how bad
- If the adversary can force a bad position
  - It’s no use searching to find the good positions the adversary won’t let you achieve
- Bad = not better than we can get elsewhere

ALPHA BETA PRUNING

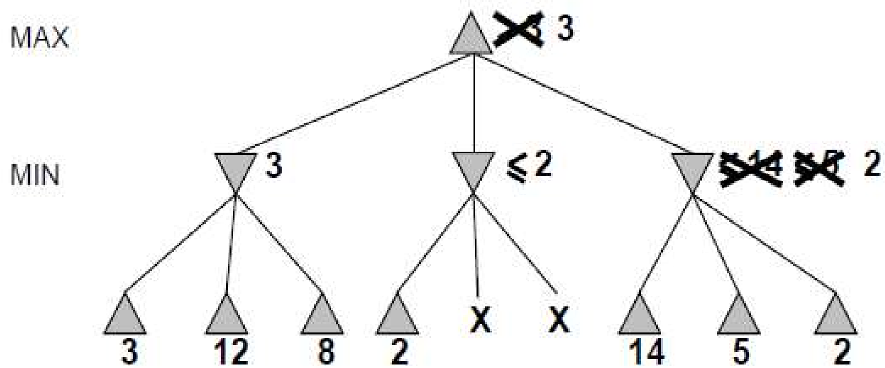


Do we need to expand all nodes?

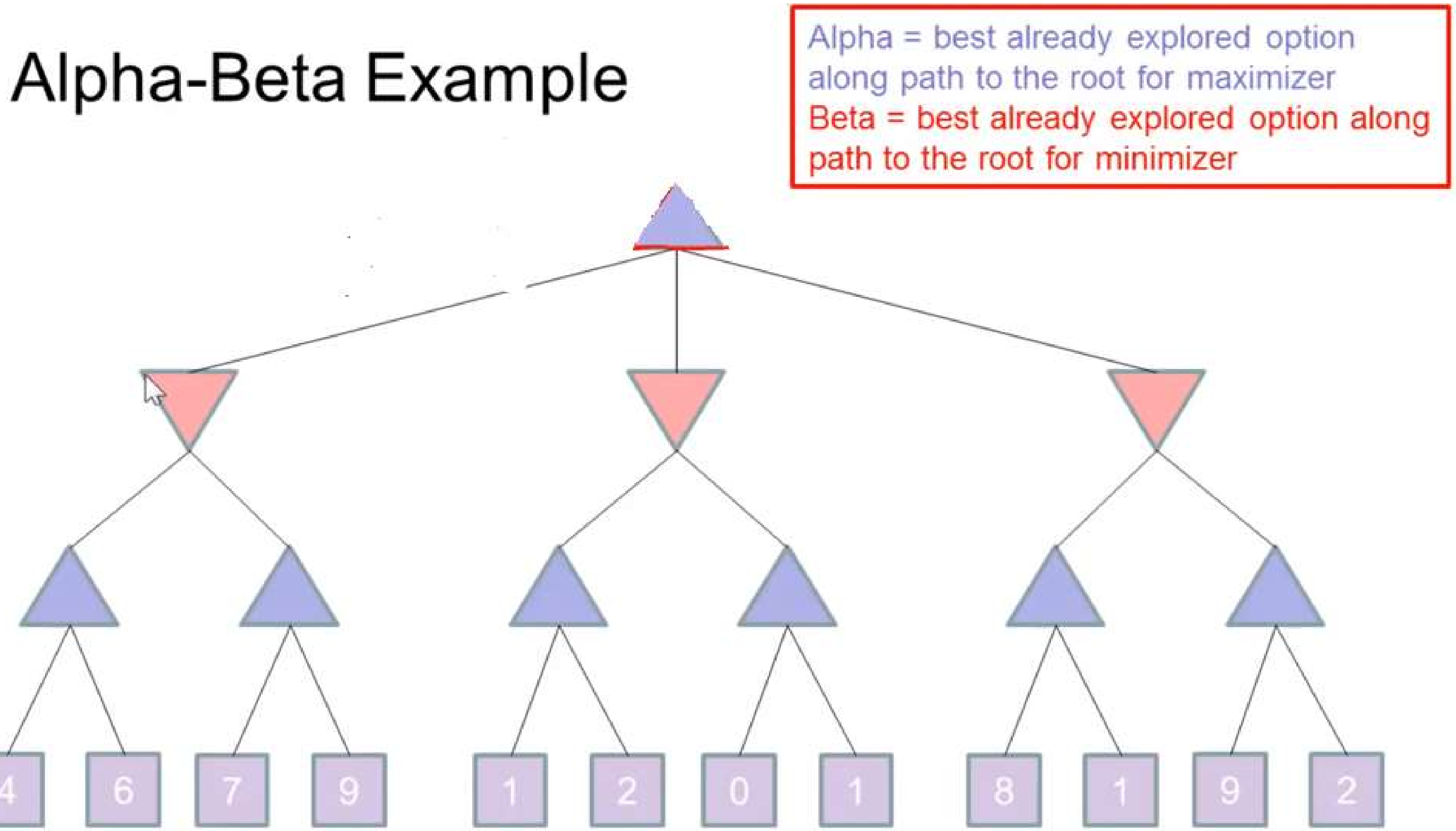
$$\begin{aligned} minimax(root) &= max(min(3,12,8), min(2,x,y), min(14,5,2)) \\ &= max(3, min(2,x,y), 2) \\ &= max(3, z, 2) \\ &= 3 \end{aligned}$$


MAX
MIN
3
≤ 2
2
3
3
12
8
2
X
X
14
5
2

ALPHA BETA PRUNING



EXERCISE



Alpha-Beta Example
Alpha = best already explored option along path to the root for maximizer
Beta = best already explored option along path to the root for minimizer
4
6
7
9
1
2
0
1
8
1
9
2