Previously…

Use Heuristics to Guide Search

- Greedy best-first search
- A^* search

A^* Search Heuristic

- If $h(n)$ is admissible, then A^* w. Tree-Search is optimal
- If $h(n)$ is consistent, then A^* w. Graph-Search is optimal
- A heuristic that dominates another incurs lower search cost

LOCAL SEARCH

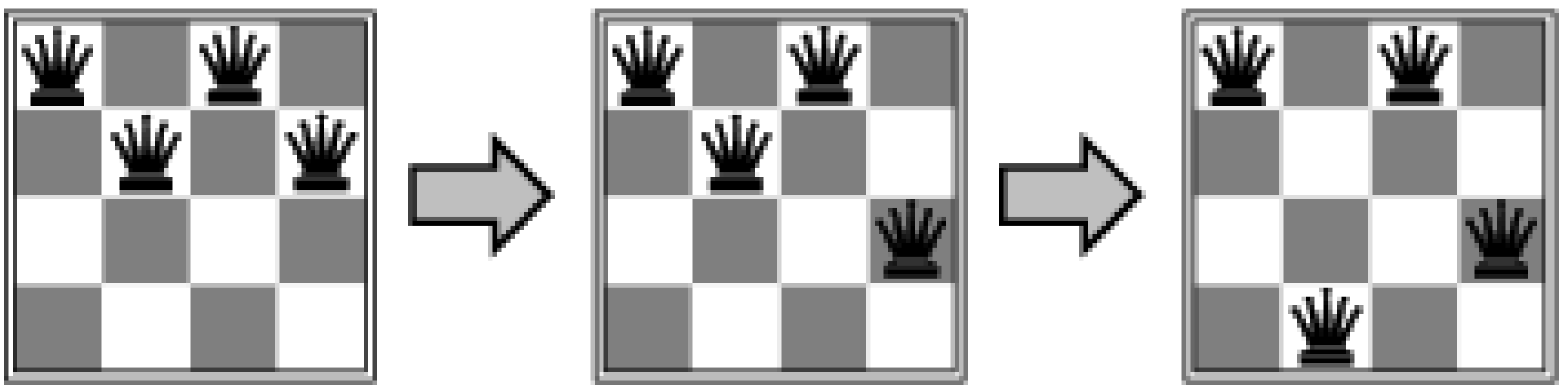
AIMA Chapter 4.1

Local Search Algorithms

- The path to goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
- Find final configuration satisfying constraints, e.g., n-queens
- Local search algorithms: maintain single "current best" state and try to improve it
- Advantages:
  - very little/constant memory
  - find reasonable solutions in large state space

Example: n-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal



Hill-Climbing Search

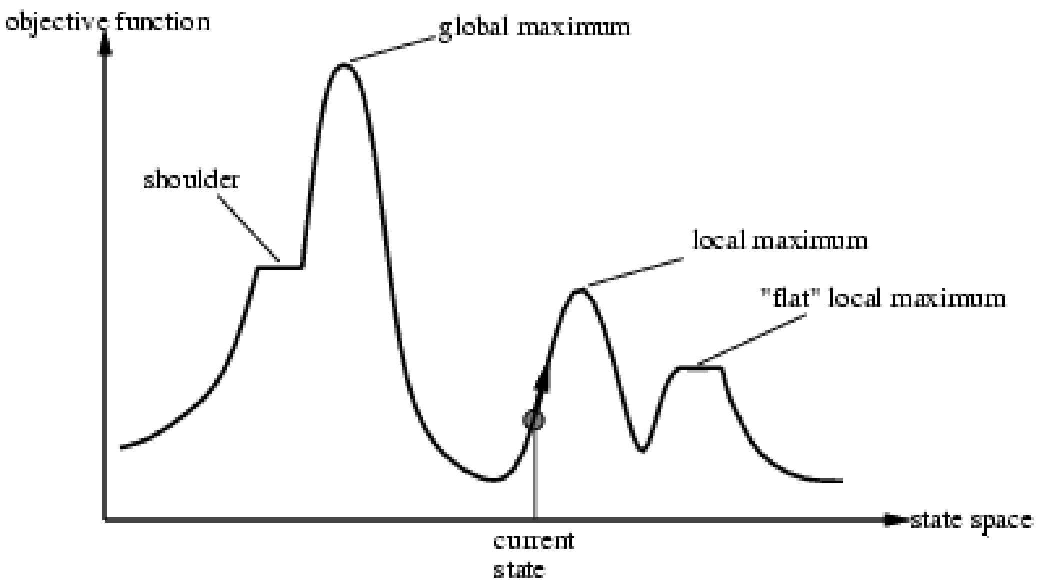
```
function HILL-CLIMBING(problem) returns a state that is a local maximum

  current ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
      neighbor ← a highest-valued successor of current
      if neighbor.VALUE ≤ current.VALUE then return current.STATE
      current ← neighbor
```

"Like climbing Mt. Everest in thick fog with amnesia"

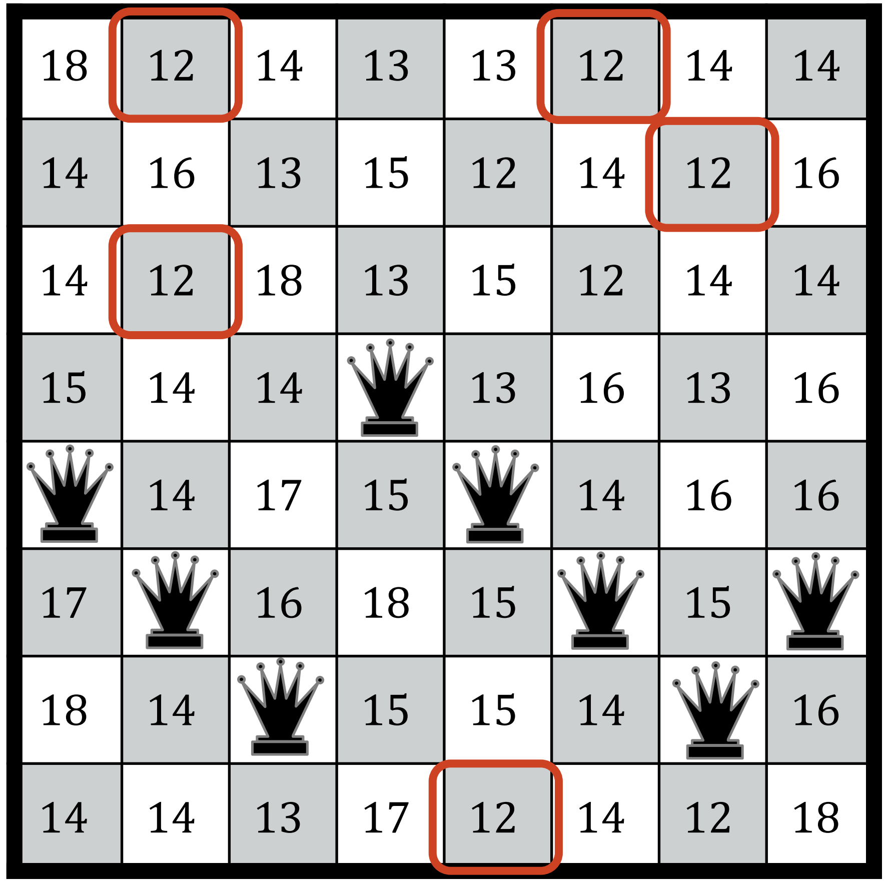
Hill-Climbing Search

- Problem: depending on initial state, can get stuck in local maxima



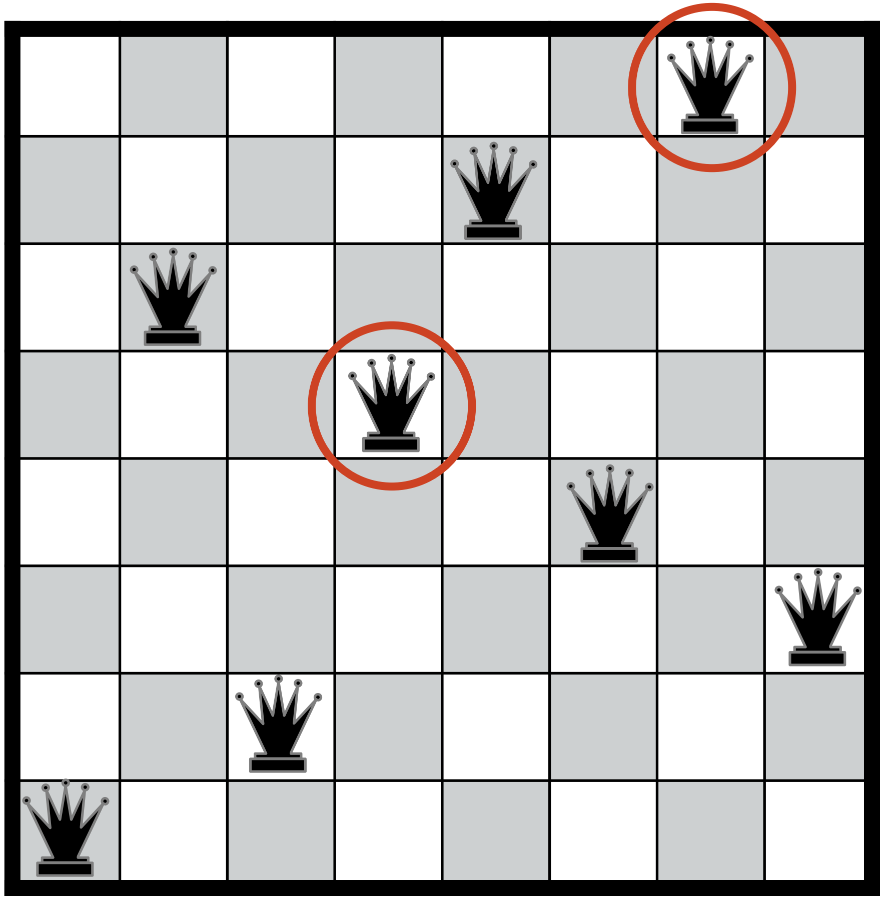
- Non-guaranteed fixes: sideway moves, random restarts

Hill-Climbing Search: 8-Queens



- $h =$ number of pairs of queens that are attacking each other, either directly or indirectly
- $h = 17$ for the above state

Hill-Climbing Search: 8-Queens



Local Minimum with $h = 1$

Local search strategies

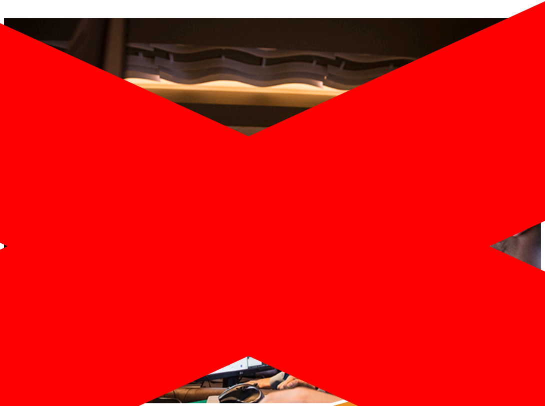
- Hill-climbing search: use of heuristic function to improve "current" state

Adversarial Search

AIMA Chapter 5.1 – 5.5

AI vs. Human Players: the State of the Art

Kaspa
B



World Poker Champs vs
Libratus (2017)

DeepMind's
o (2016)

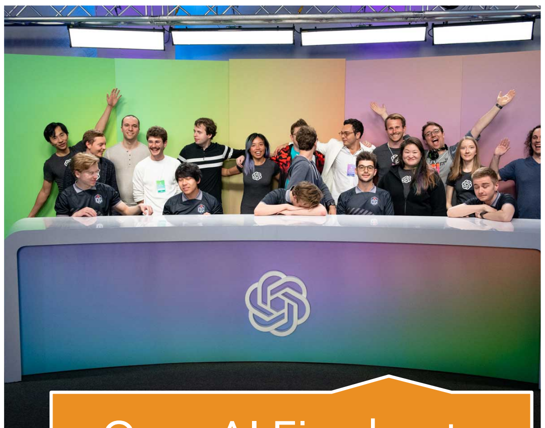
AI vs. Human Players: the State of the Art

World
Lib

Lee Sedol Vs. DeepMind
AlphaGo (2016)

n solve any
m game!

AI vs. Human Players: the State of the Art



OpenAI Five beats pro DOTA 2 players.

Pluribus beats multi-player poker pros

AlphaStar beats top Starcraft II players

TO BE UPDATED NEXT YEAR!

Deterministic Games in Practice

- Chinook (Checkers, 1994). Precomputed endgame database ⇒ perfect play for all positions with ≤ 8 pieces on the board (total of 444 billion positions).
- Deep Blue (Chess, 1997). Searches 200 million positions/sec + evaluation functions + secret sauce.
- Logistello (Othello, 1997). Human champions refuse to play against AI.
- AlphaGo + Alphazero (Go/everything above, 2016-2017). Learning for evaluation functions + database and efficient search + secret sauce.

Outline

- Adversarial search problems (aka games)
- Optimal (i.e., Minimax) decisions
- α-β pruning
- Imperfect, real-time decisions

Games vs. Search Problems

Utility maximizing opponent

- solution is a strategy specifying a move for every possible opponent response.

Time limit

- unlikely to find goal, must approximate

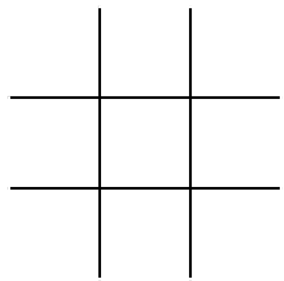
Let's Play!

- Two players in a **zero-sum game**:
  - Winner gets paid and loser pays.
- Easy to think in terms of a **max player** and **min player**
  - Player 1 wants to maximize value (MAX player)
  - Player 2 wants to minimize value (MIN player)

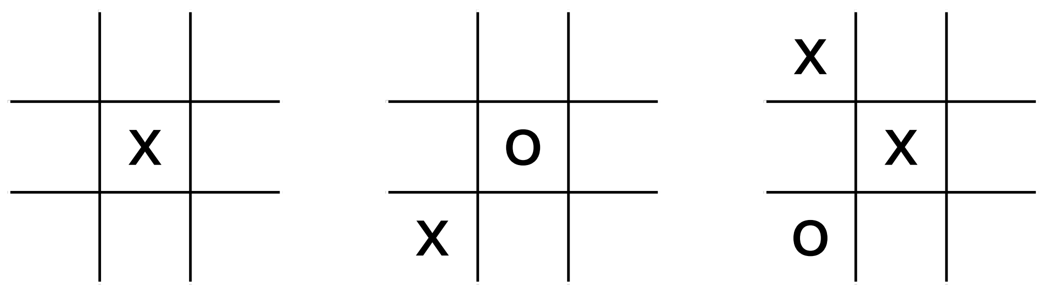
Game: Problem Formulation

A game is defined by 7 components:

1. Initial state s_0



2. States



3. Players: $PLAYER(s)$ defines which player has the move in state s.

Game: Problem Formulation

A game is defined by 7 components:

4. Actions : $ACTIONS(s)$ returns set of legal moves in state s

X	O	
	X	
O	X	O

$a = (\mathbf{X}, \langle 1,3 \rangle)$
$a = (\mathbf{X}, \langle 2,1 \rangle)$
$a = (\mathbf{X}, \langle 2,3 \rangle)$

5. Transition model : $RESULT(s, a)$ returns state that results from the move a in state s.

X		
	X	
O		

$a = (\mathbf{O}, \langle 2,3 \rangle)$

X		
	X	O
O		

Game: Problem Formulation

A game is defined by 7 components:

6. Terminal test $TERMINAL(s) = true$ iff game end

X	O	X
O	X	X
O	X	O

X		O
O	X	
X	O	X

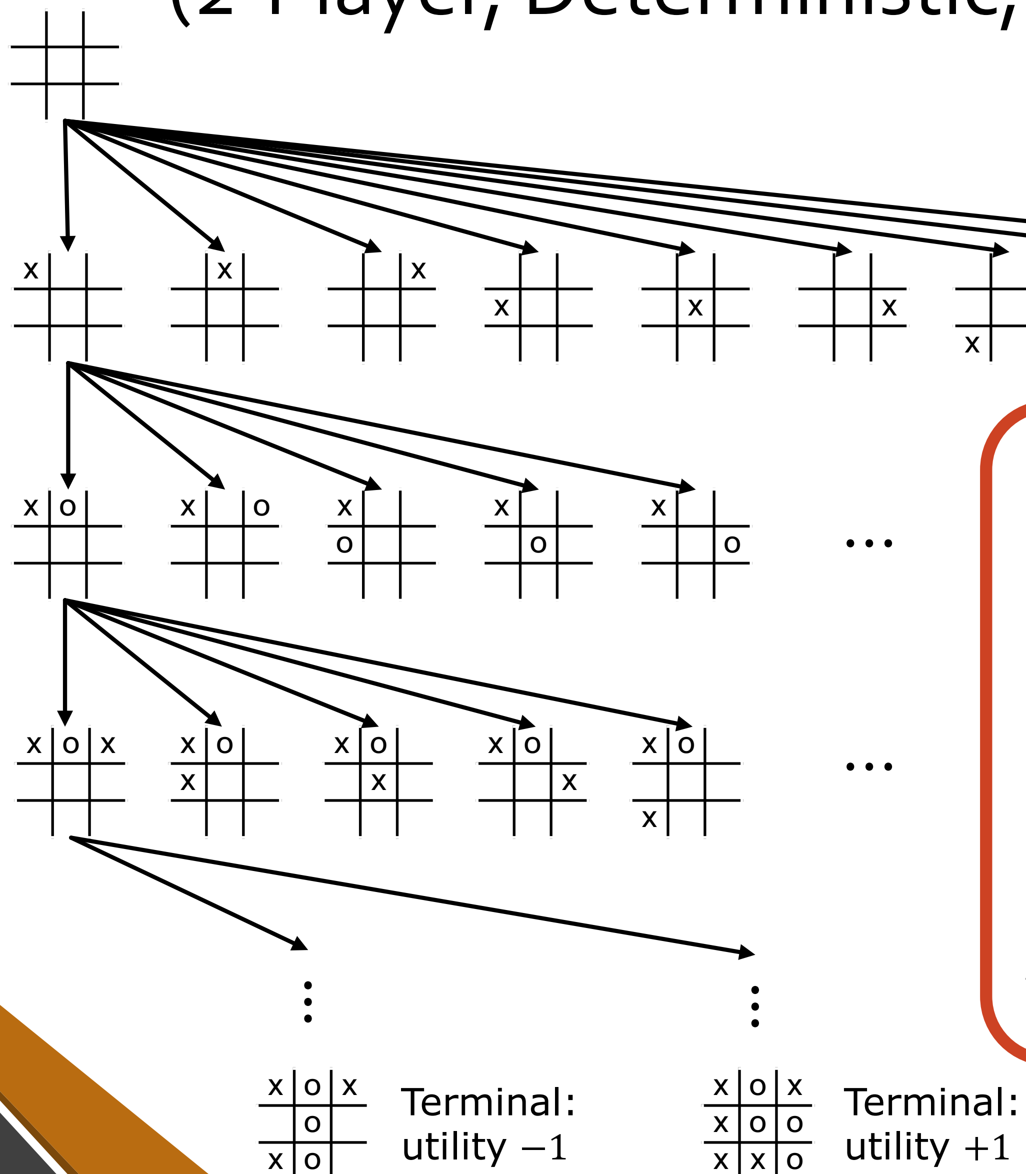
X	X	O
	O	O
X	X	O

O	X	O
X	X	O
X		O

7. Utility function $UTILITY(s, p)$: final numeric value for a game that ends in terminal state s for a player p

- Tic-Tac-Toe: X wins $+1$; O wins -1; draw 0.

Game Tree
(2-Player, Deterministic, Turn-Taking)



Zero-sum games

- A loser for every winner
- Agent utilities sum to zero
- Aka constant-sum game
- Completely adversarial game

Non Zero-Sum Games?

Example : Game of NIM

Several piles of sticks are given. We represent the configuration of the piles by a monotone sequence of integers, such as $(1,3,5)$. A player may remove, in one turn, any number of sticks from one pile. Thus, $(1,3,5)$ would become $(1,1,3)$ if the player were to remove 4 sticks from the last pile. The player who takes the last stick loses.

- Represent the NIM game $(1,2,2)$ as a game tree.

Player Strategies

A strategy s for player i: what will player i do at every node of the tree that they make a move in?

Need to specify behavior in states that will never be reached!

Winning Strategy

A strategy s_1^* for player 1 is called **winning** if for any strategy s_2 by player 2, the game ends with player 1 as the winner.

A strategy t_1^* for player 1 is called **non-losing** if for any strategy s_2 by player 2, the game ends in either a tie or a win for player 1.

Theorem (Von Neumann):
in the game of chess, only one of the following is true:

1. White has a winning strategy s_w^*
2. Black has a winning strategy s_b^*
3. Each player has a non-losing strategy.

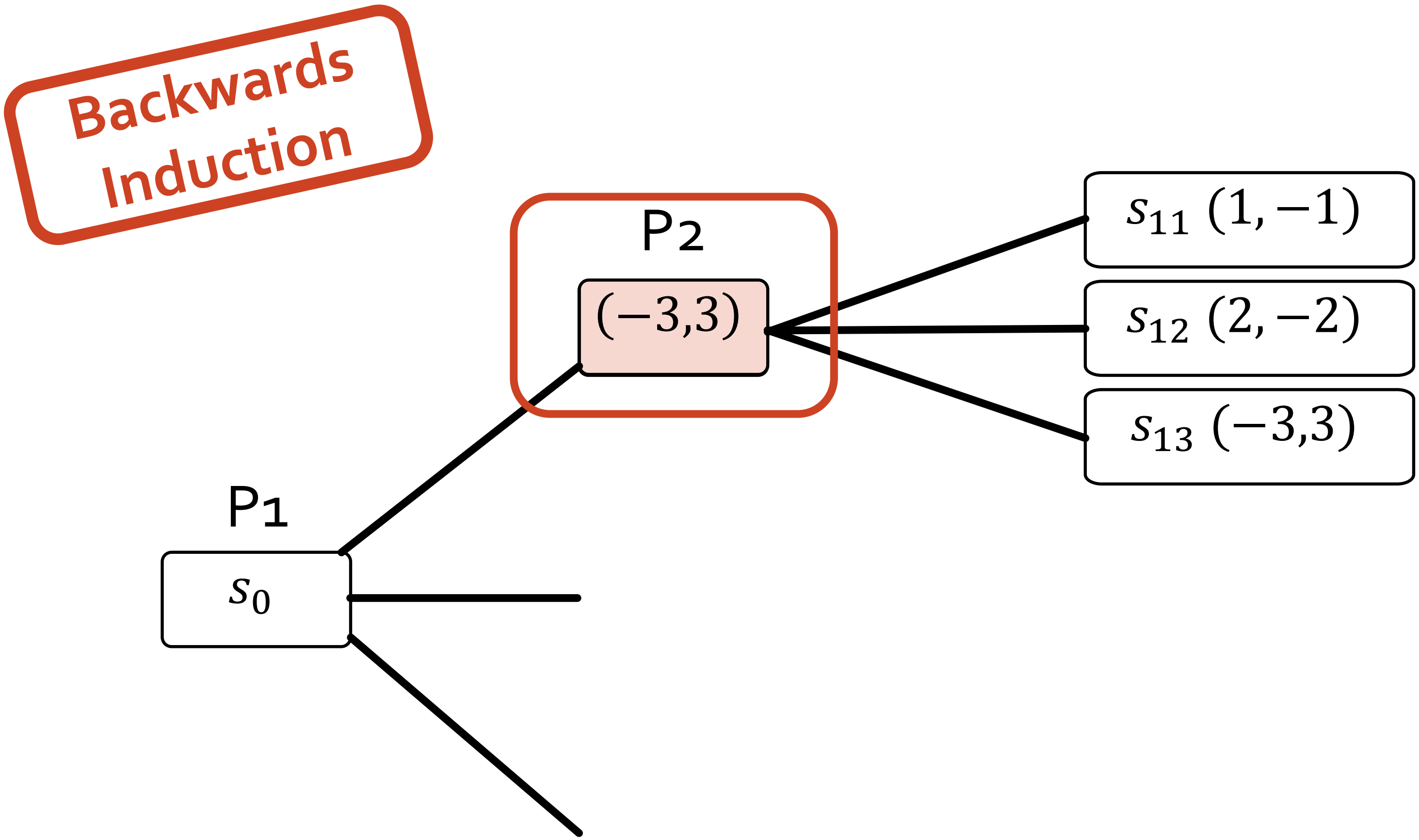
Optimal Strategy at Node - Minimax

$$Minimax(s) = \begin{cases} Utility(s) \text{ if TerminalTest}(s) \\ \max_{a \in \text{Actions}(s)} \text{Minimax}\big(\text{Result}(s,a)\big) \text{ if Player}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{Minimax}\big(\text{Result}(s,a)\big) \text{ if Player}(s) = \text{MIN} \end{cases}$$

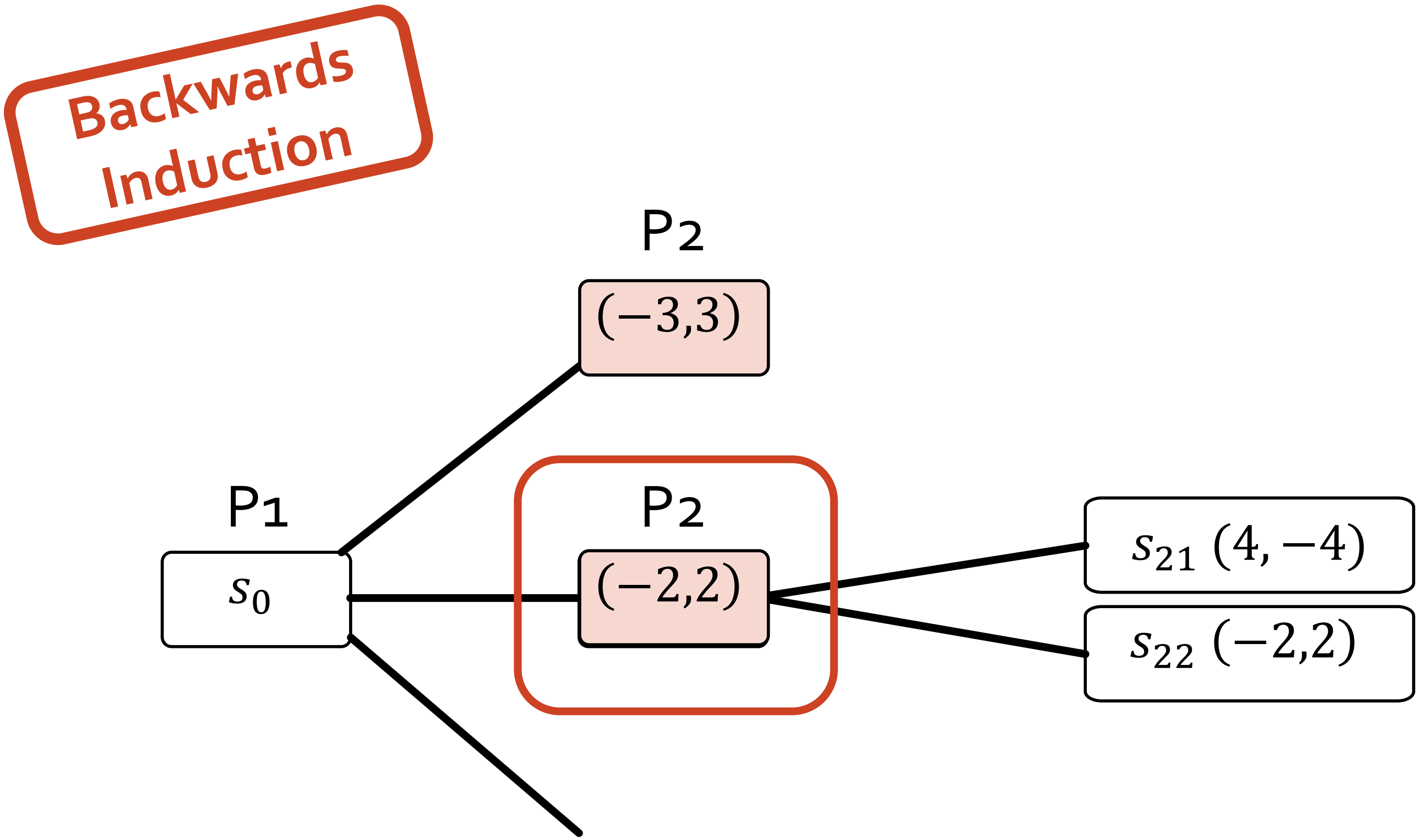
Intuitively,

- MAX chooses move to maximize the minimum payoff
- MIN chooses move to minimize the maximum payoff

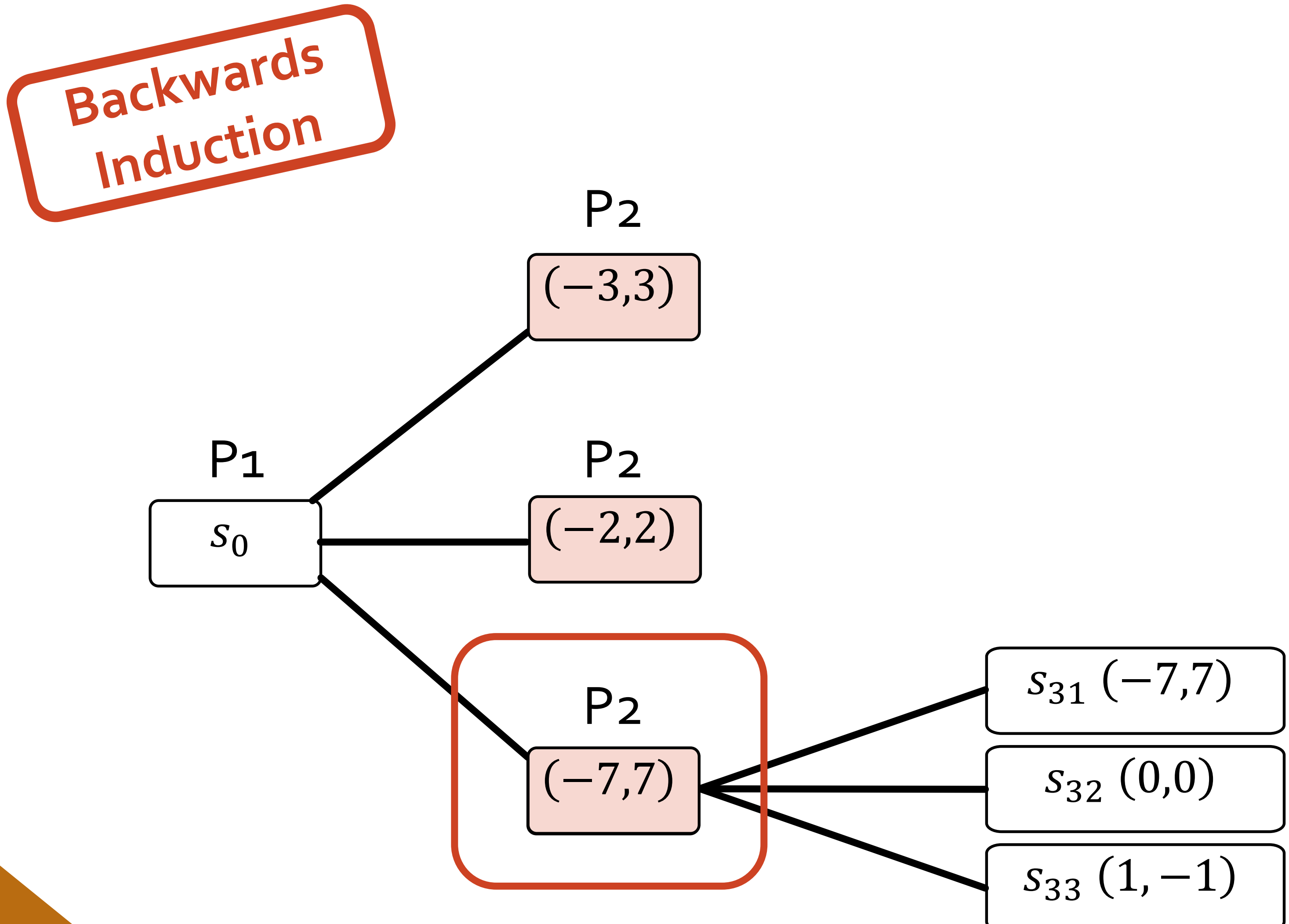
Minimax Play (Subperfect Nash Equilibrium)



Minimax Play
(Subperfect Nash Equilibrium)

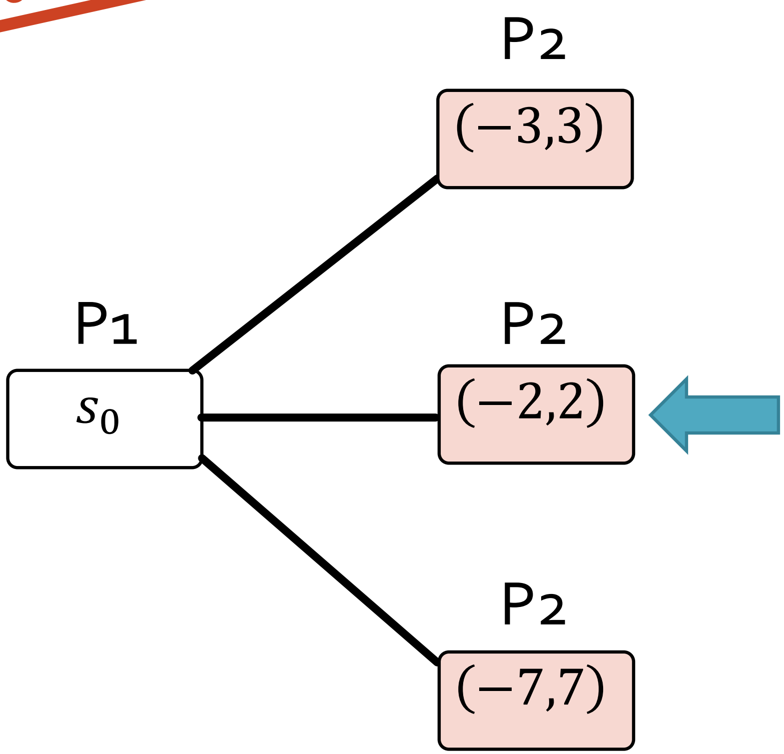


Minimax Play
(Subperfect Nash Equilibrium)



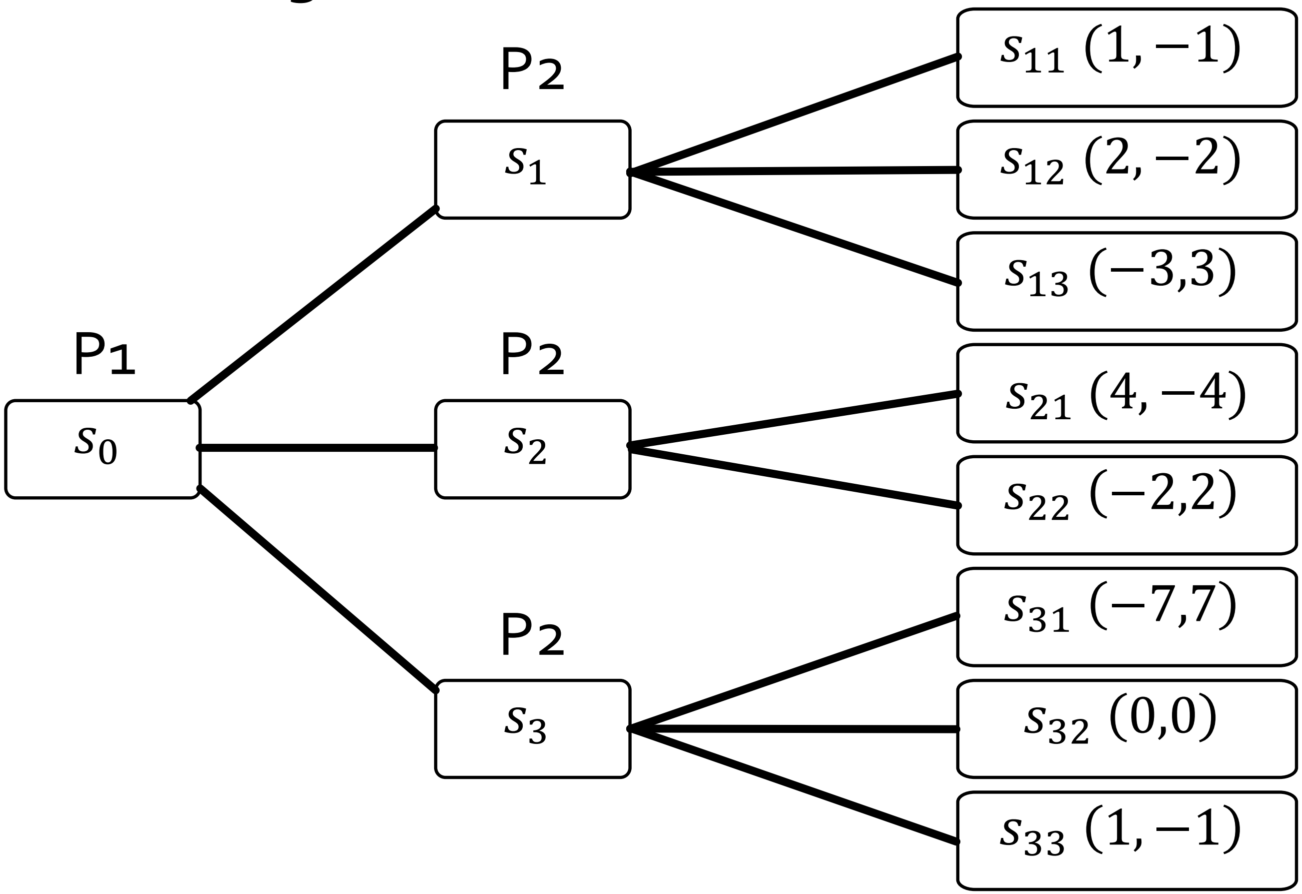
Minimax Play
(Subperfect Nash Equilibrium)

Backwards Induction



Minimax Play
(Subperfect Nash Equilibrium)

What are the optimal strategies in this game?



Properties of Minimax

Property	
Complete?	Yes (if game tree is finite)
Optimal	Yes (optimal gameplay)
Time	$\mathcal{O}(b^m)$
Space	Like DFS: $\mathcal{O}(bm)$

Minimax Algorithm

- Runs in time polynomial in tree size
- Returns a sub-perfect Nash equilibrium: the best action at every choice node.

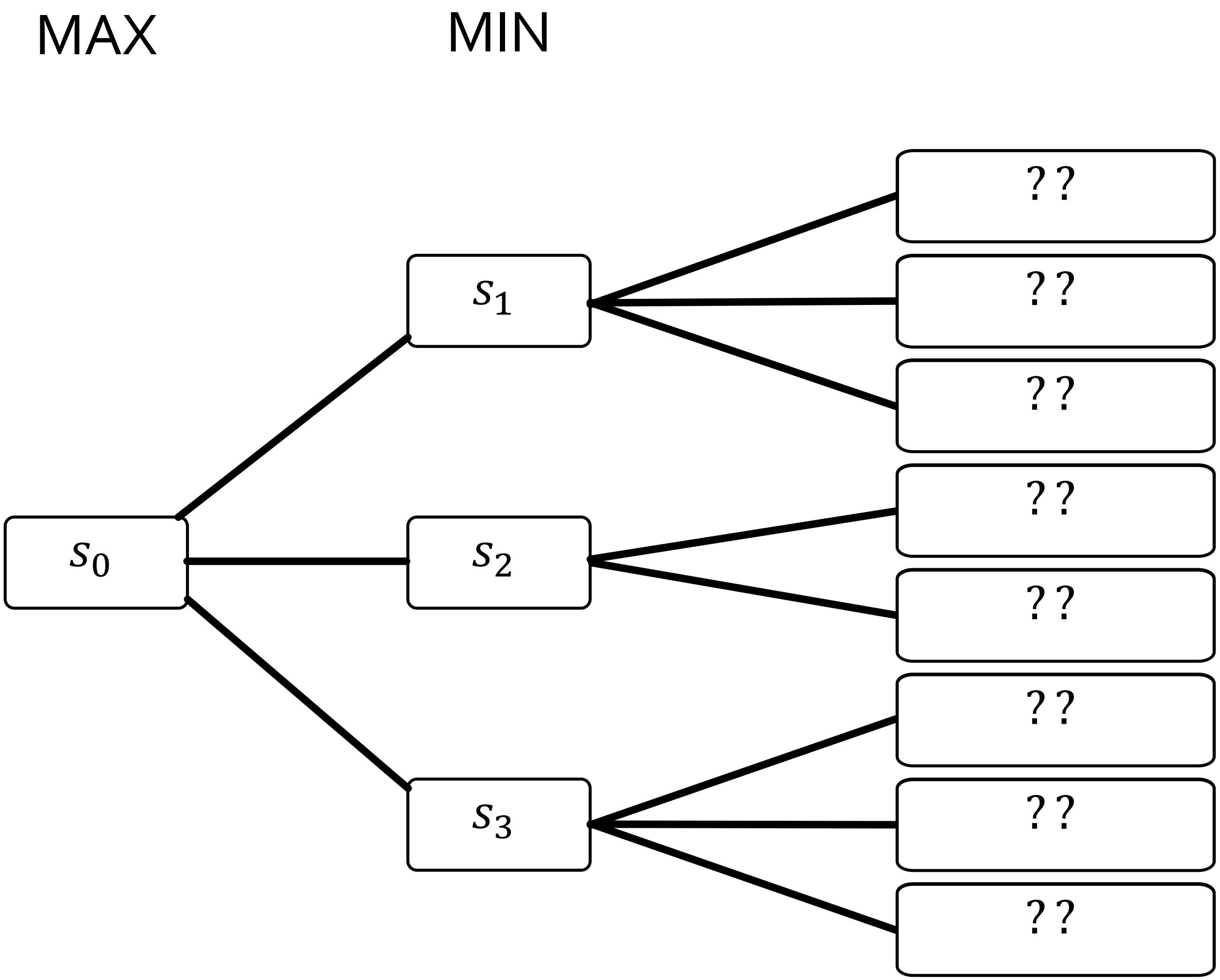
Are we done here?

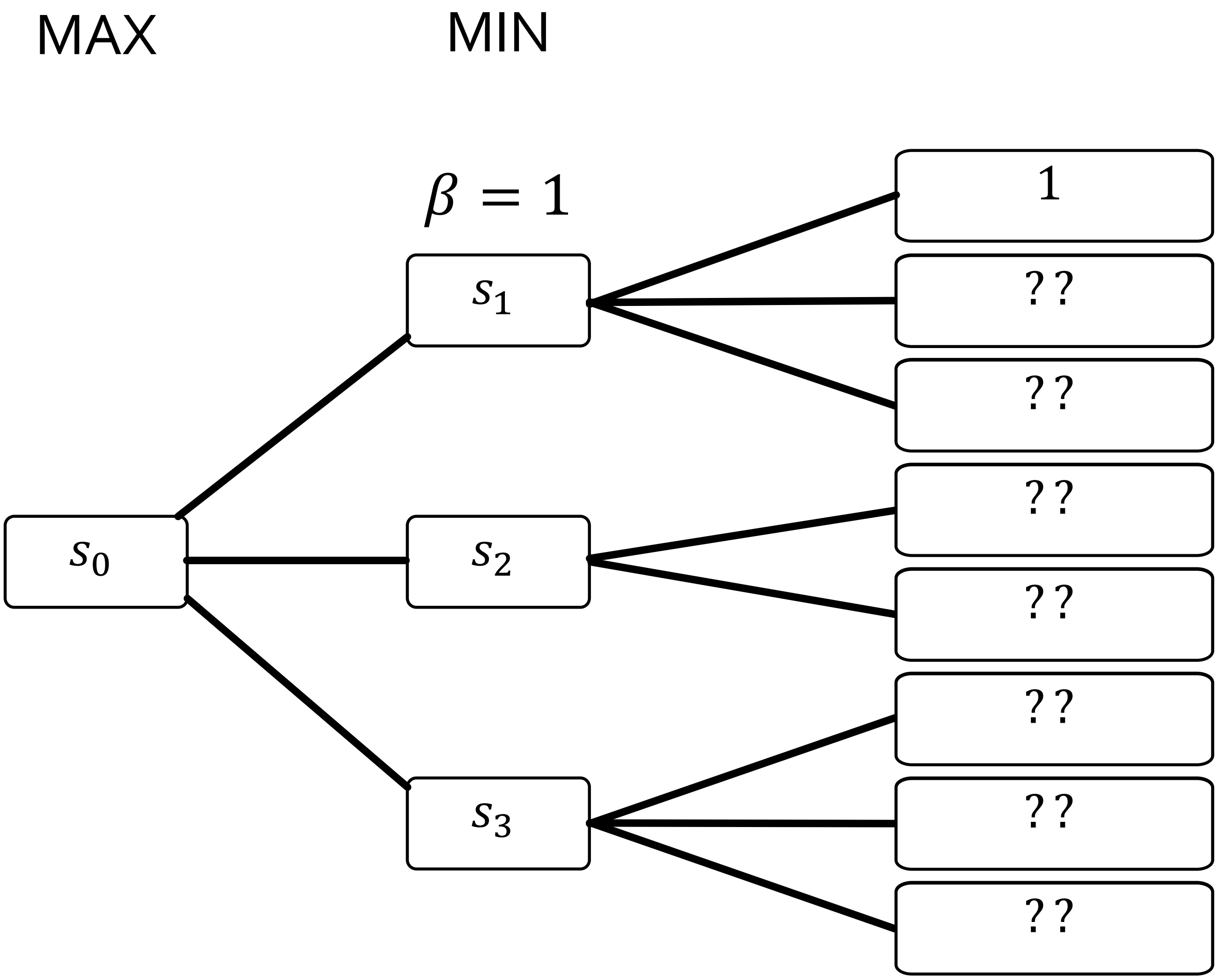
Backwards Induction

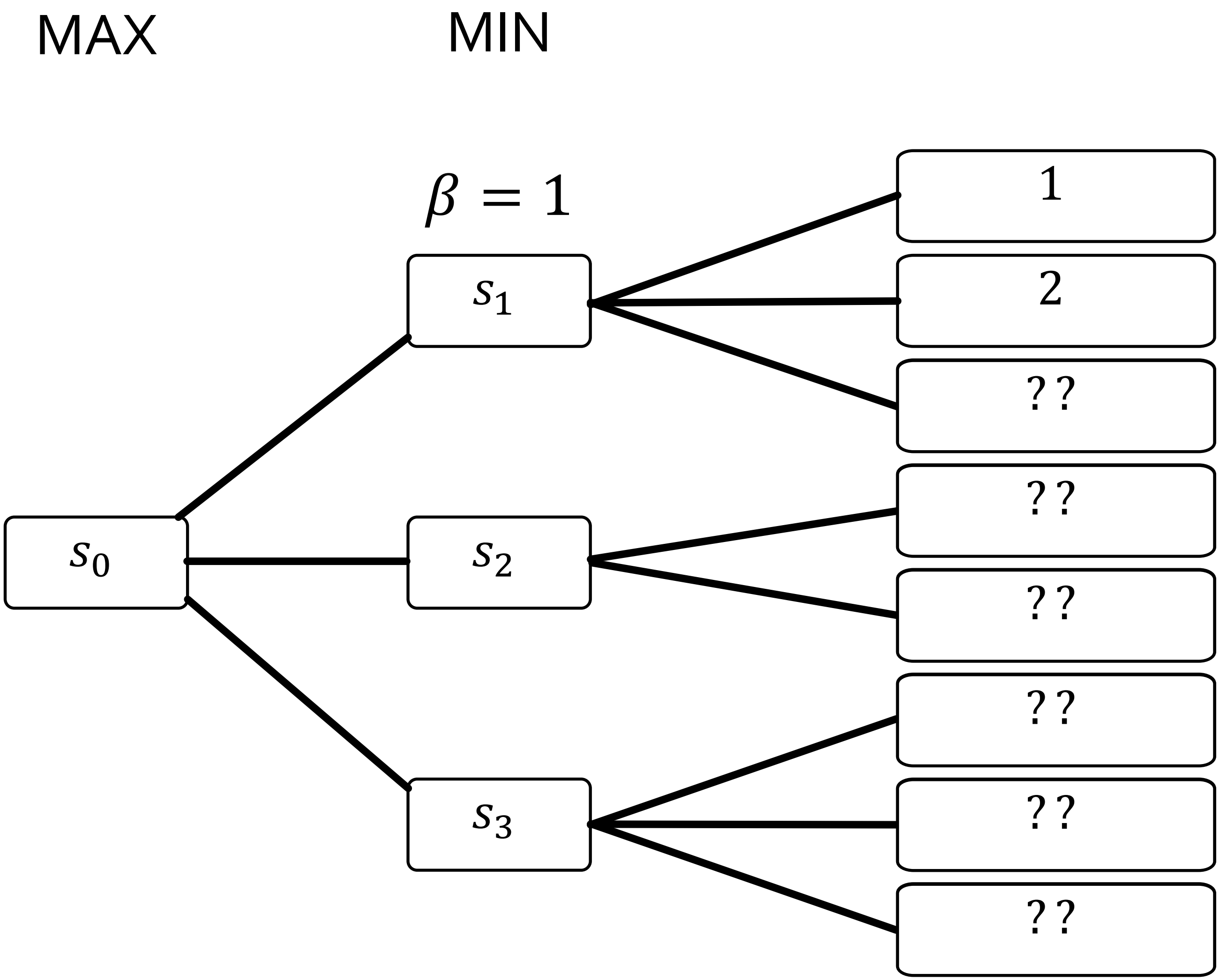
- Game trees are huge: chess has game tree with $\sim 10^{123}$ nodes (planet Earth has $\sim 10^{50}$ atoms)
- Impossible to expand the entire tree

α-β Pruning

- **Basic idea:** *"If you have an idea that is surely bad, don't take the time to see how truly awful it is."* -- Pat Winston
- Maintain a **lower bound $\boldsymbol{\alpha}$** and **upper bound $\boldsymbol{\beta}$** of the values of, respectively, MAX's and MIN's nodes seen thus far.
- Prune subtrees that will never affect minimax decision.

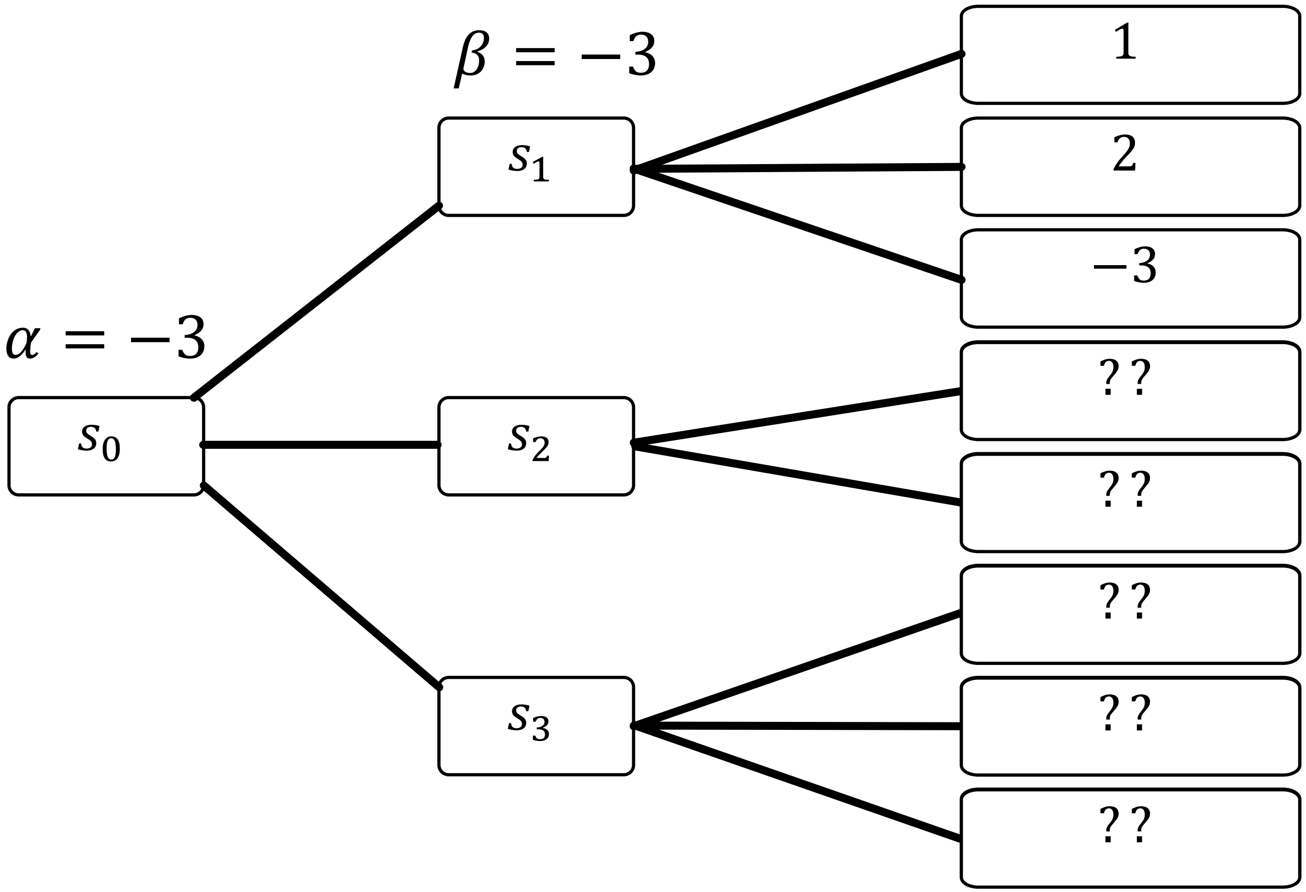

MAX
MIN
s_0
s_1
s_2
s_3
??
??
??
??
??
??
??
??

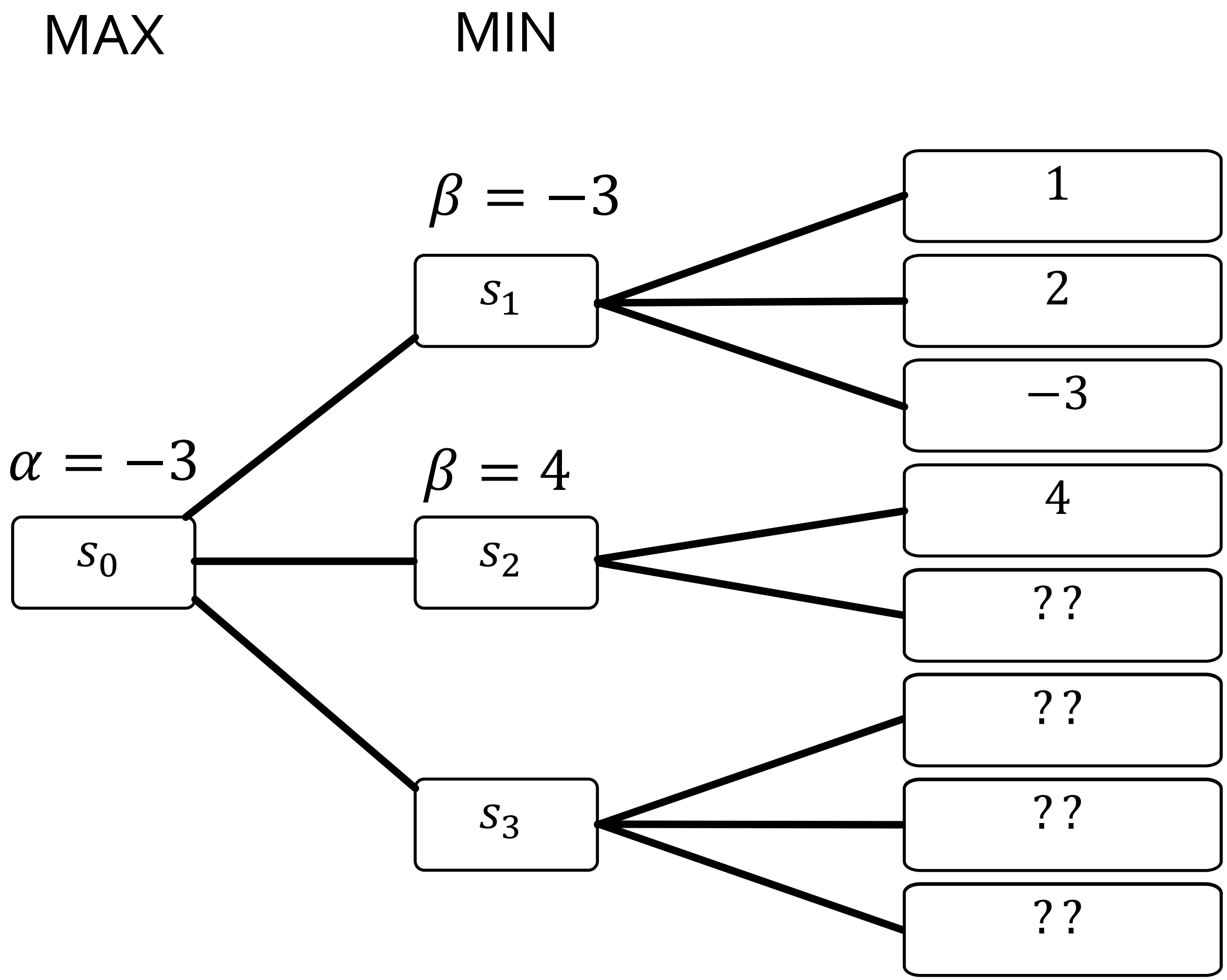

MAX
MIN
$\beta = 1$
s_1
s_0
s_2
s_3
1
??
??
??
??
??
??
??

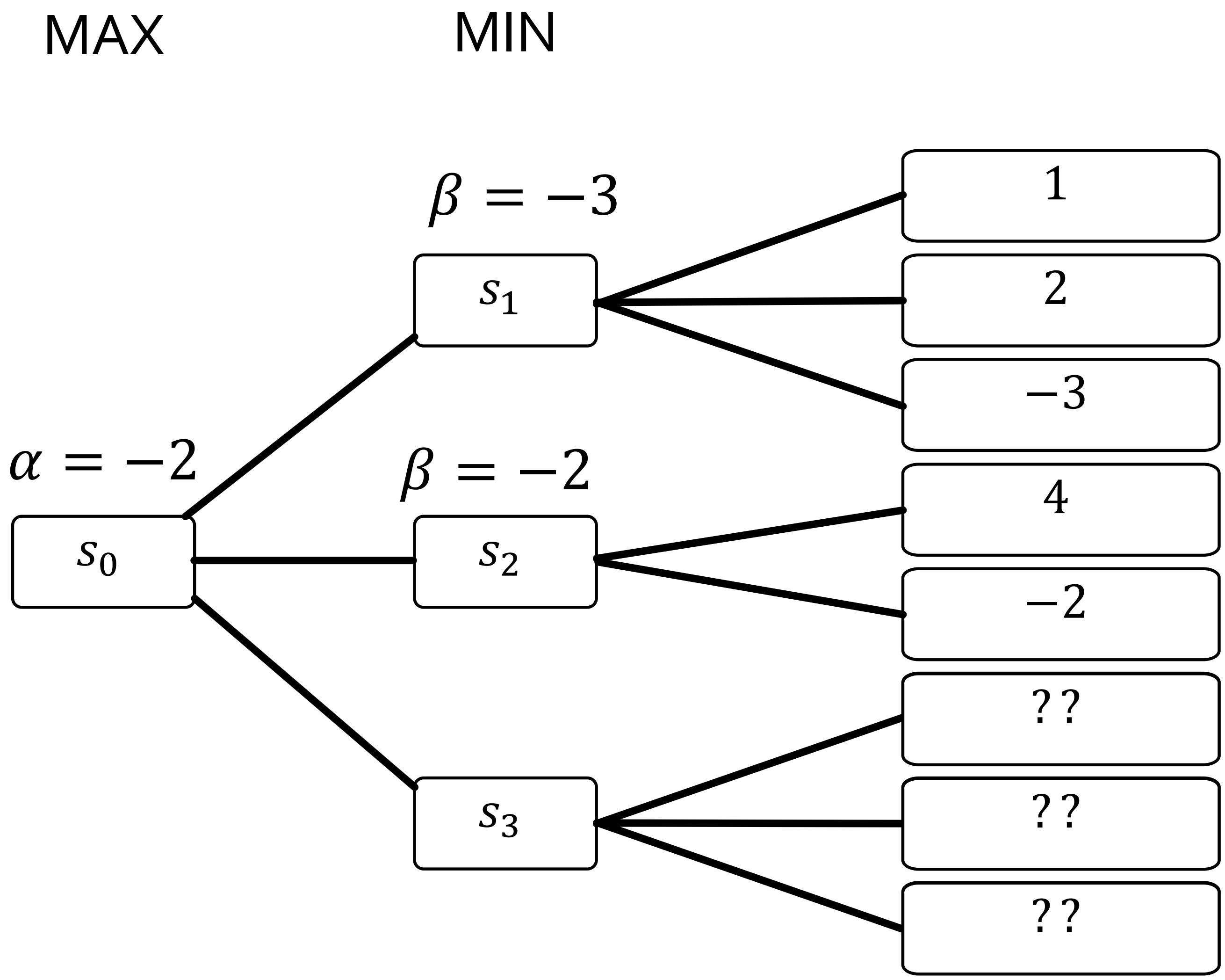

MAX
MIN
$\beta = 1$
s_0
s_1
s_2
s_3
1
2
??
??
??
??
??
??

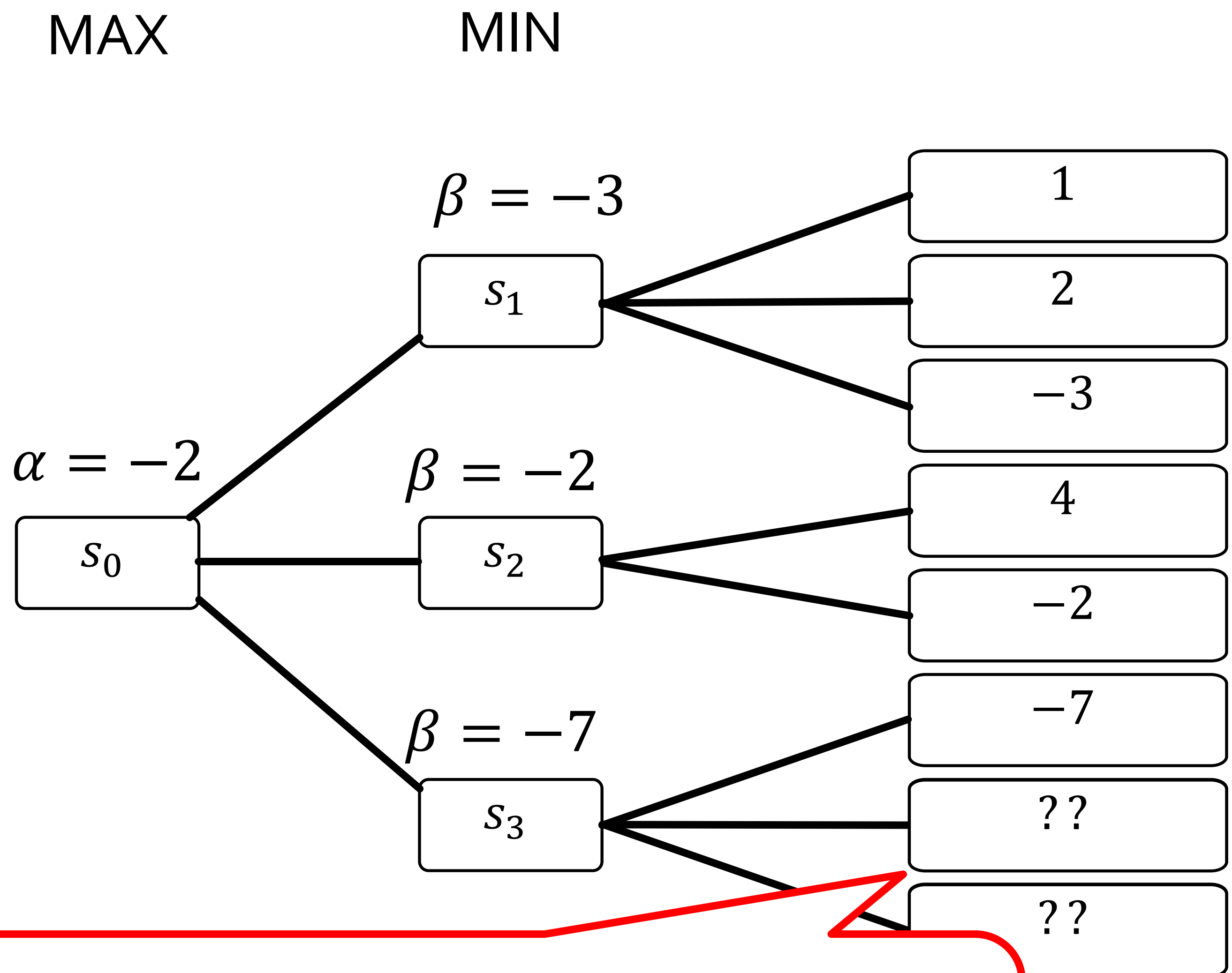
MAX

MIN

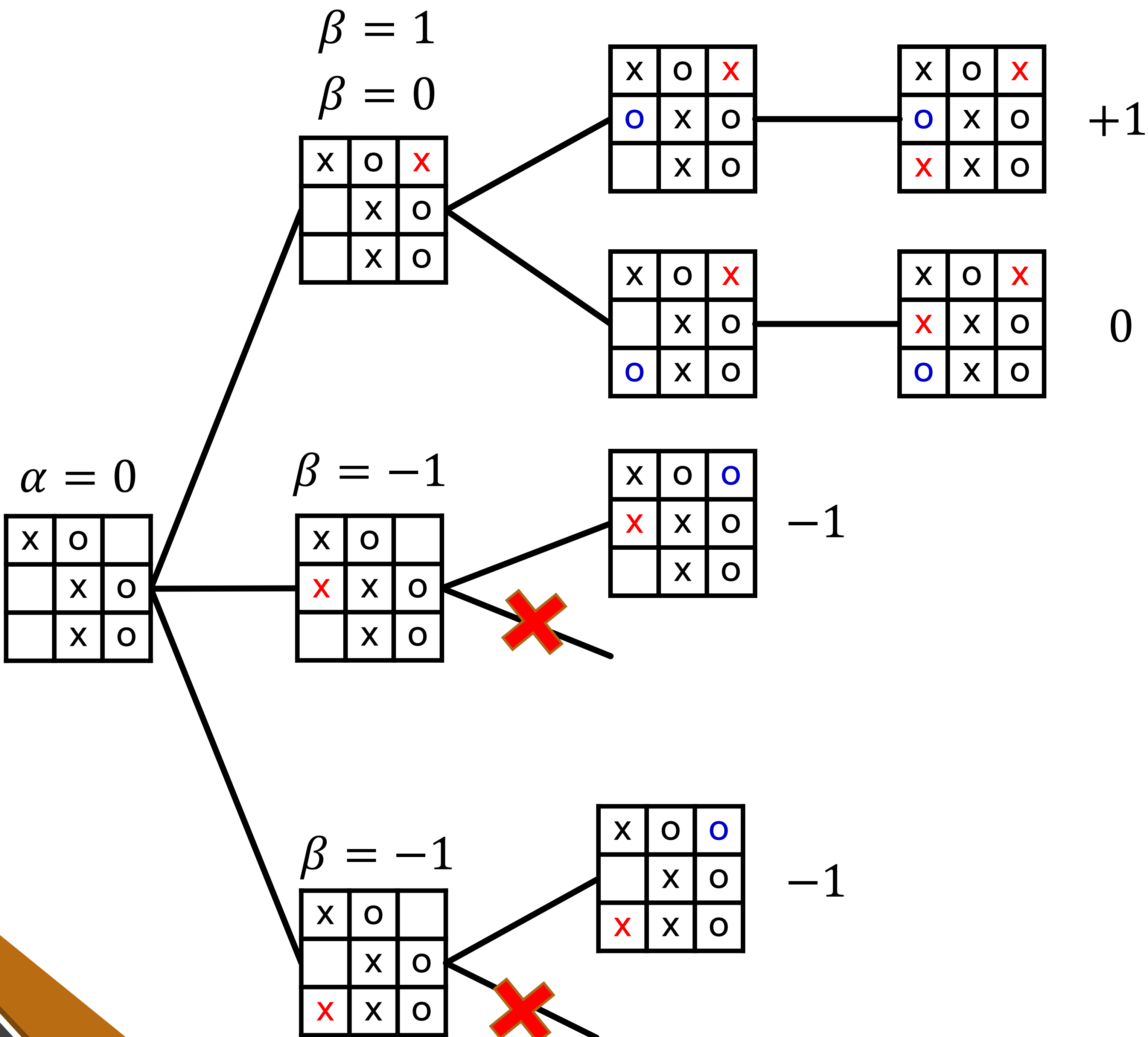



MAX
MIN
$\beta = -3$
$\alpha = -3$
$\beta = 4$
s_0
s_1
s_2
s_3
1
2
-3
4
??
??
??
??


MAX
MIN
$\beta = -3$
$\alpha = -2$
$\beta = -2$
s_0
s_1
s_2
s_3
1
2
-3
4
-2
??
??
??



No point exploring the last two nodes!
Choosing s_3 results in a loss of at least 7…



β = 1
β = 0
+1
0
α = 0
β = −1
−1
β = −1
−1

α-β Pruning

- MAX node n: $\alpha(n) =$ highest **observed** value found on path from n; initially $\alpha(n) = -\infty$
- MIN node n: $\beta(n)$ is the lowest observed value found on path from n; initially $\beta(n) = +\infty$
- Pruning:
  - given a MIN node n, stop searching below n if there is some MAX ancestor i of n with $\alpha(i) \geq \beta(n)$
  - given a MAX node n, stop searching below n if there is some MIN ancestor i of n with $\beta(i) \leq \alpha(n)$

Analysis of α-β Pruning

- When we prune a branch, it **never** affects final outcome.
- Good move ordering improves effectiveness of pruning
- “Perfect” ordering: time complexity =$\mathcal{O}\left(b^{\frac{m}{2}}\right)$

  → Good pruning strategies allow us to search twice as deep!

  - Chess: simple ordering (checks, then take pieces, then forward moves, then backwards moves) gets you close to best-case result.
  - It makes sense to have good expansion order heuristics.
- Random ordering: complexity = $\mathcal{O}\left(b^{\frac{3m}{4}}\right)$ for $b < 1000$

Summary: α-β Pruning Algorithm

- Initially, $\alpha(n) = -\infty, \beta(n) = +\infty$
- $\alpha(n)$ is max along search path containing n
- $\beta(n)$ is min along search path containing n
- If a MIN node has value $v \leq \alpha(n)$, no need to explore further.
- If a MAX node has value $v \geq \beta(n)$, no need to explore further.

Time Limit

- **Problem:** very large search space in typical games
- **Solution:** α-β pruning removes large parts of search space
- Unresolved problem: Maximum depth of tree
- Standard solutions:
  - evaluation function = estimated expected utility of state
  - cutoff test: e.g., depth limit

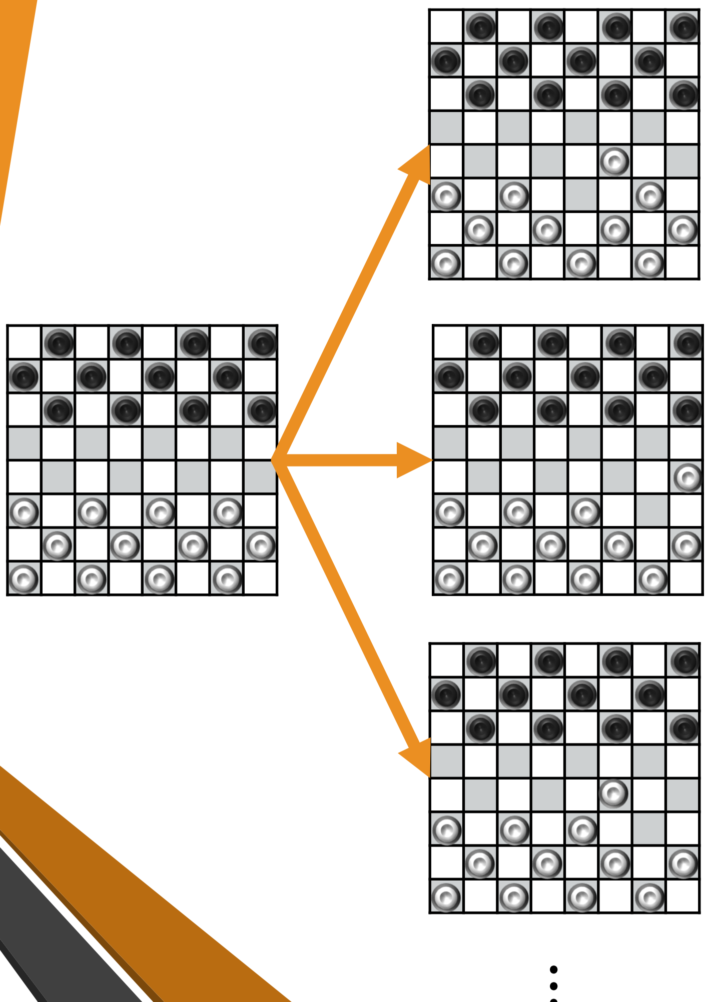
Heuristic Minimax Value

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

$$
\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \boxed{\text{if CUTOFF-TEST}(s,d)} \\ \max_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN} \end{cases}
$$

Run minimax until depth d; then start using the evaluation function to choose nodes.

Evaluation Functions



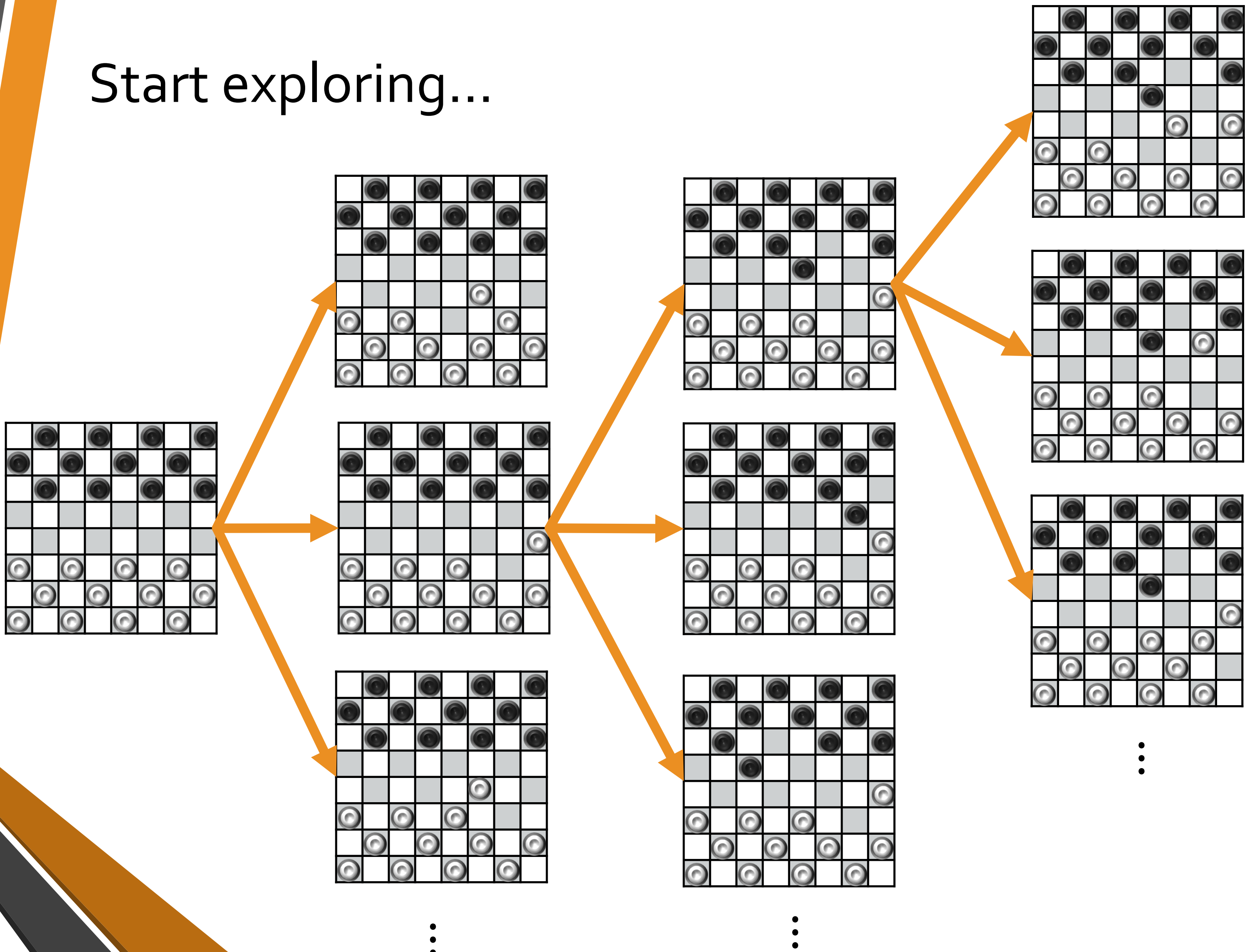
How good is this move?

To know for sure, we need to explore entire subtree up to terminal states.

Unrealistic (even for ‘simple’ game of checkers)

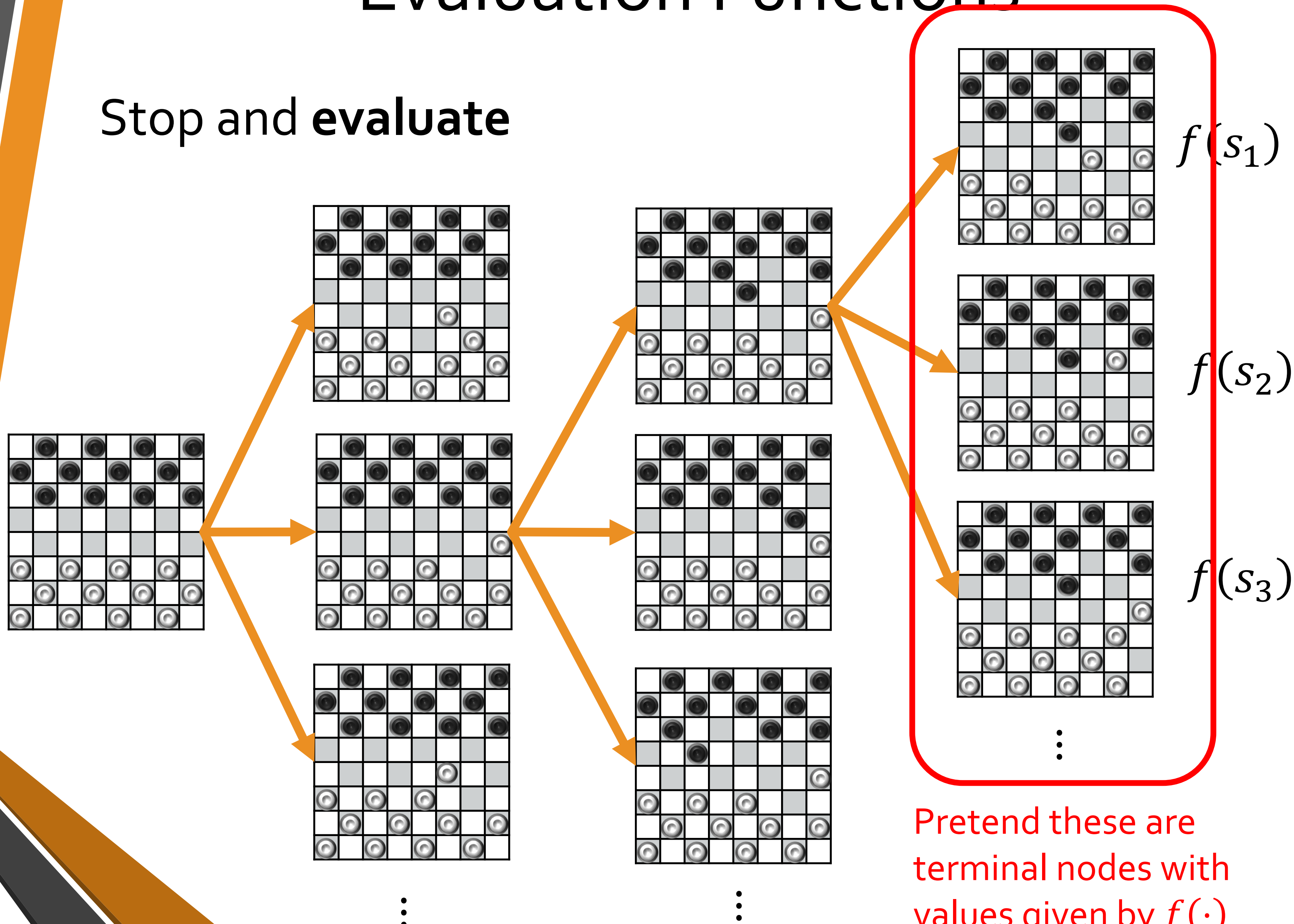
Evaluation Functions

Start exploring…



Evaluation Functions

Stop and **evaluate**



Pretend these are terminal nodes with values given by $f(\cdot)$

Evaluation Functions

- An evaluation function is a mapping from game states to real values: $f: \mathcal{S} \rightarrow \mathbb{R}$
  - So far:

$$f(s) = \begin{cases} UTILITY(s) \text{ if } TERMINAL(s) \\ 0 \ otherwise \end{cases}$$

No information on quality of non-terminal nodes

- For non-terminal states, must be strongly correlated with actual chances of winning

Evaluation Functions

Important Features

- # of pieces
- # of queens
- # of controlled squares
- # of threatened opponent pieces
- …

$$f(n) = w_1 \times (NPcs) + w_2 \times (NQns) + w_3 \times (CtlSqs) + w_4 \times (ThrPcs)$$

$$\boxed{\boldsymbol{w_1, \ldots, w_4} = ???}$$

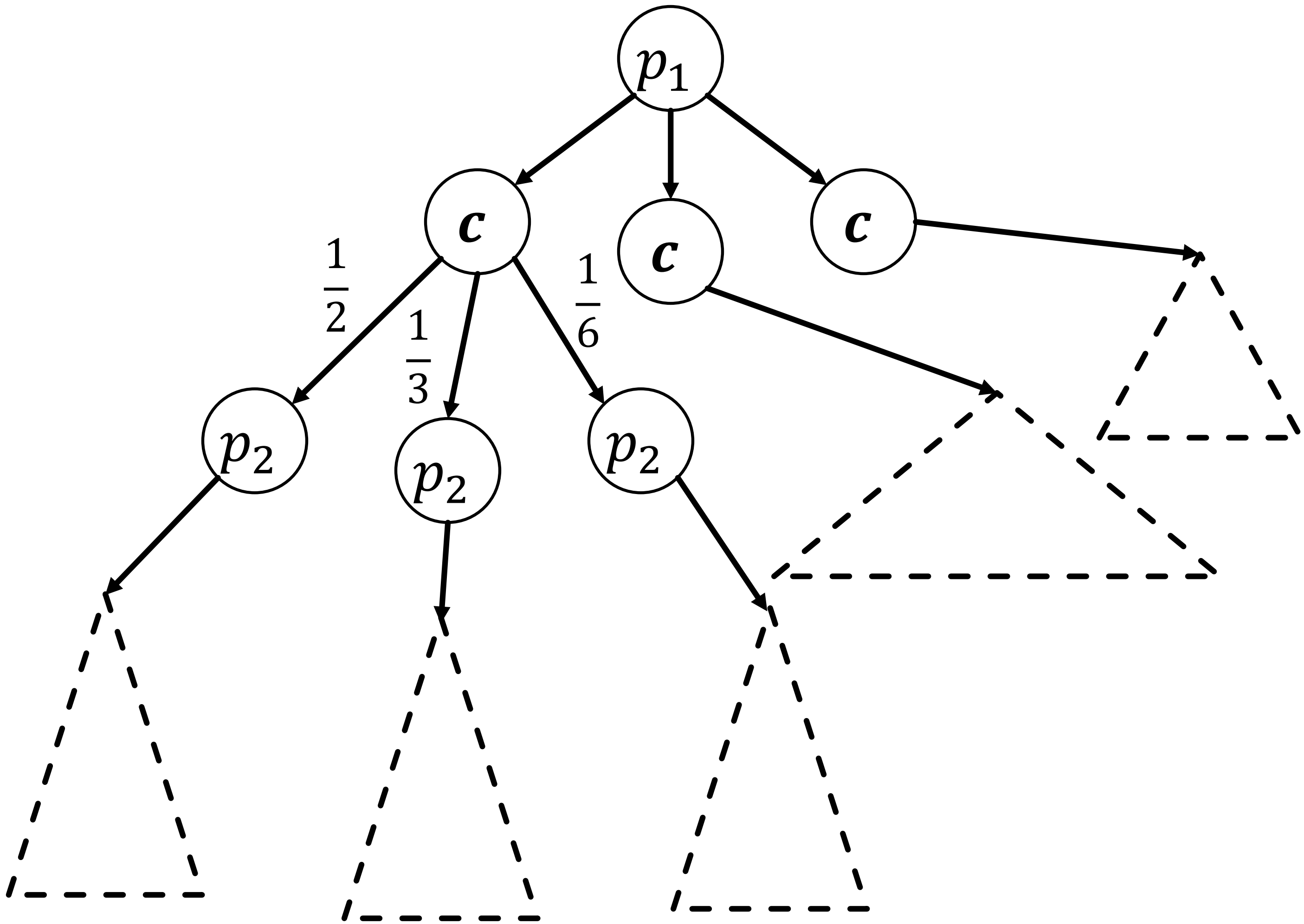
Cutting Off Search

- Modify minimax or α-β pruning algorithms by replacing
  - $\text{TERMINALTEST}(s)$ with $\text{CUTOFFTEST}(s, d)$
  - $\text{UTILITY}(s)$ is replaced by $\text{EVAL}(s)$
- Can also be combined with iterative deepening

Stochastic Games

- Many games have randomization:
  - Backgammon
  - Settlers of Catan
  - Poker
- How do we deal with uncertainty?
- Can we still use minimax? Yes, but search space is much bigger

Adding Chance Layers



Calculate the expected value of a state (MUCH harder than deterministic games)