Adversarial Search: Playing Games

CS3243: Introduction to Artificial Intelligence – Lecture 7

6 March 2023

Contents

Reference: AIMA 4th Edition, Section 6.1-6.3

Administrative Matters

Upcoming…

- Deadlines
  - None this week!

  - TA6 (released today)
    - *Due in your Week 6 tutorial session*
    - *Submit the a physical copy (more instructions on the Tutorial Worksheet)*
  - Prepare for the tutorial!
    - Participation marks = 5%

  - Midterm Marks Appeal
    - *Due next week (i.e., Week 9) on the day of your assigned tutorial session, 2359 hrs*
  - Project 2
    - *Due next week, Sunday (19 March), 2359 hrs*

Reviewing Search Problems

So Far...

- Path search (path planning)
  - Search for a path from start to goal
    - Complete: finds a solution or says when there isn't one
    - Optimal: path cost of path found is minimal
  - Uninformed
    - Systematically search all paths via general search problem formulation
  - Informed
    - Uses a heuristic to search less of the search space
- Goal search
  - Focus on goal and ignore path
    - Completeness consideration only
  - Local search
    - Uses heuristic to guide search to goal (uses restarts; many variants)
  - Constraint satisfaction problem
    - Uses specific search problem formulation and shrinks search space via inference

Games

Games and Search

- Can we solve games using existing methods?
  - In our searching thus far, we control all actions
    - All actions taken are determined by our agent
  - With games, your opponent decides actions too…
    - Multi-agent problem
    - Conventional planning ⇒ wasted computation since opponent can spoil your plans

Games and Search

- What is a game anyway?
  - Assume two players
  - Zero-sum game
    - Winner gets paid, and loser pays
  - We define
    - MAX player – player 1, who wants to maximise value (agent)
    - MIN player – player 2, who want to minimise value (opponent who wants agent to lose)
- General idea behind the search problem
  - Simulate play against utility maximising opponent
  - Find a strategy – i.e., define a move for every possible opponent response

Search Problem Formulation for Games

Formulating Games

- State representation
  - As per general formulation

- TO-MOVE(s)
  - Returns p, the player to move in state s
- ACTIONS(s)
  - Legal moves in state s
- RESULT(s, a)
  - The transition model; returns resultant state when taking action a at state s
- IS-TERMINAL(s)
  - Returns TRUE when game is over and FALSE otherwise
    - States where game has ended are called terminal states
- UTILITY(s, p)
  - Defines the final numeric value to player p when the game ends in terminal state s

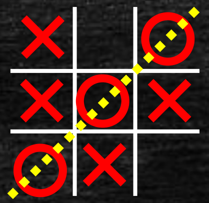
Note on Utility

- Given zero-sum games
  - At terminal state s
    - UTILITY(MAX,s) + UTILITY(MIN,s) = 0
- Tic-Tac-Toe example
  - X (agent) wins
    - UTILITY(s_i, MAX) = 1
    - UTILITY(s_i, MIN) = -1

s_i :

  - O (opponent) wins
    - UTILITY(s_j, MAX) = -1
    - UTILITY(s_j, MIN) = 1

s_j :



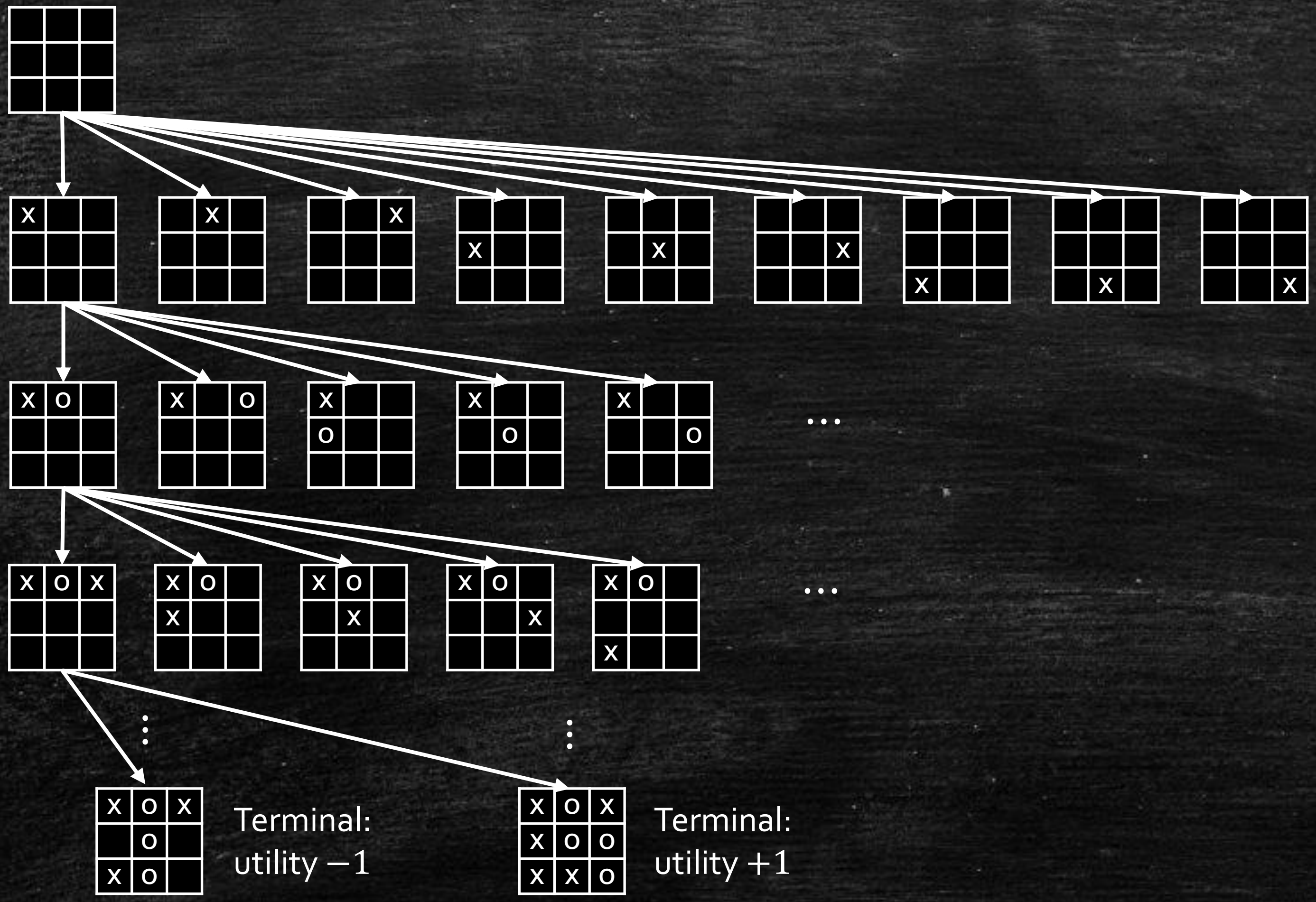
  - Draw
    - UTILITY(s_k, MAX) = 0
    - UTILITY(s_k, MIN) = 0

s_k :

Notice that here the utilities are relative to the player – however, we will standardise the scores such that they reflect only the MAX player (i.e., agent) scores later

Game Trees

Example Game Tree

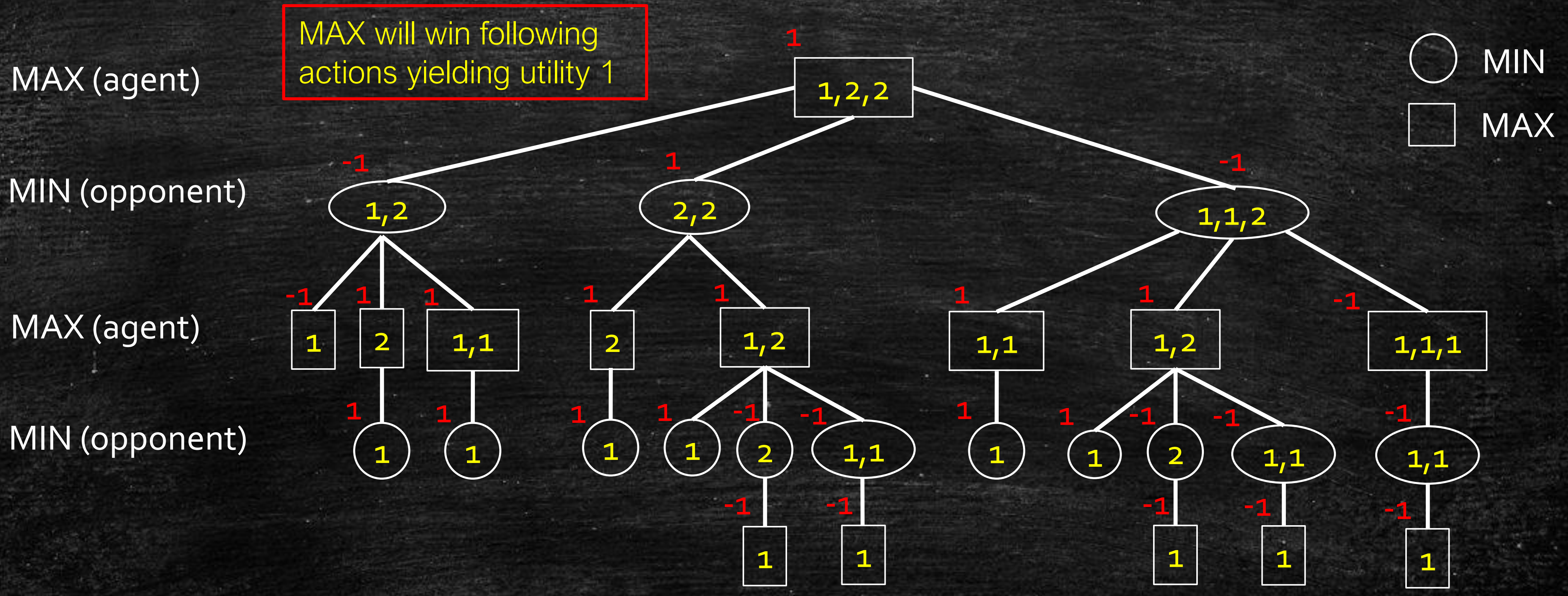


- More on environment characteristics
  - 2-player
  - Deterministic
  - Turn-taking
- Zero-sum implications
  - Loser for every winner
  - Agent utilises sum to zero
  - May also consider constant-sum game
  - Completely adversarial game

Another Example: Game of NIM

- Several piles of sticks are given
  - Represent the configuration of piles by a monotone sequence of integers
    - Example: (1,3,5)
  - With each turn, a player may remove any number of sticks from ONE pile
    - Example:
      - Remove 4 sticks from last pile (of 5 sticks) ⇒ (1,3,5) becomes (1,1,3)
  - The player who takes the last stick loses
- Let’s try…
  - Represent the NIM game (1,2,2) as a game tree

Game of NIM: (1,2,2) Game Tree



DFS Traversal with Backwards induction on utility

Strategies

Player Strategies

- A strategy **s** for ***player i*** :
  - What will *player i* do at every node of the game tree that they make a move in?
    - Need to specify behaviour in states that may 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.

- Non-losing strategy

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.

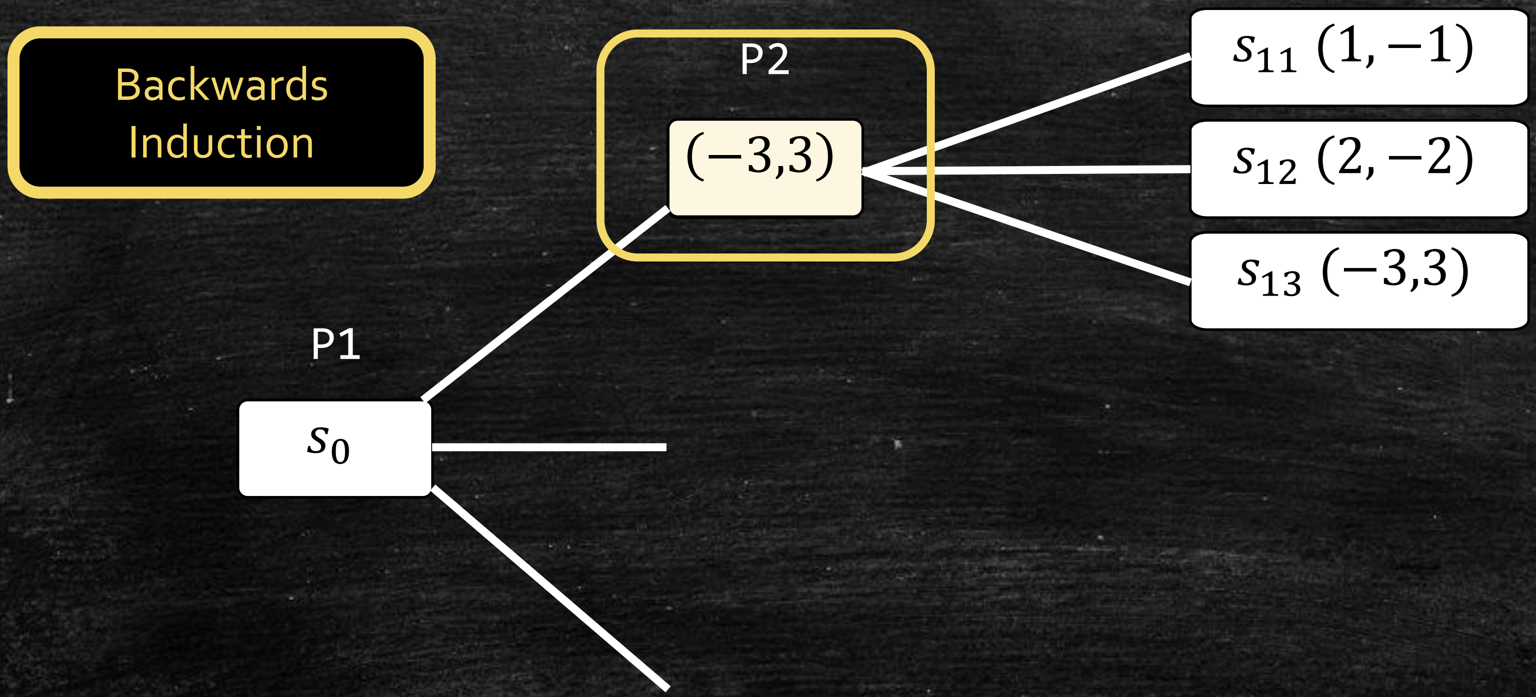
Optimal Strategy at Node - Minimax

$$Minimax(s) = \begin{cases} Utility(s, \text{MAX}) \text{ if Is}-\text{Terminal}(s) \\ \max_{a \in \text{Actions}(s)} \text{Minimax}\big(\text{Result}(s, a)\big) \text{ if To}-\text{Move}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{Minimax}\big(\text{Result}(s, a)\big) \text{ if To}-\text{Move}(s) = \text{MIN} \end{cases}$$

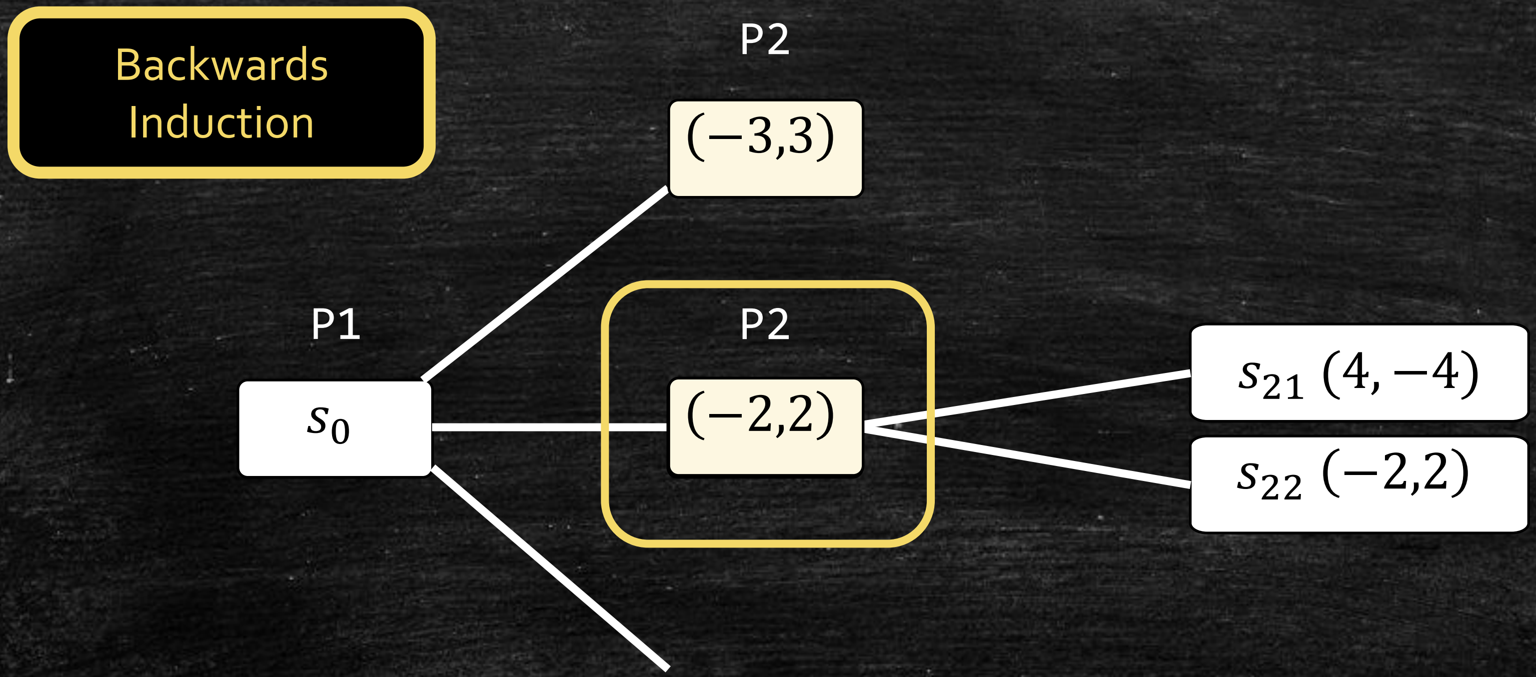
- Intuitively
  - MAX chooses move to maximise the minimum payoff
    - MIN chooses at successors
  - MIN chooses move to minimise the maximum payoff
    - MAX chooses at successors

Remember that Result(s, a) outputs the utility in terms of MAX player's utility (i.e., MIN wants to minimise this value, while MAX wants to maximise it)

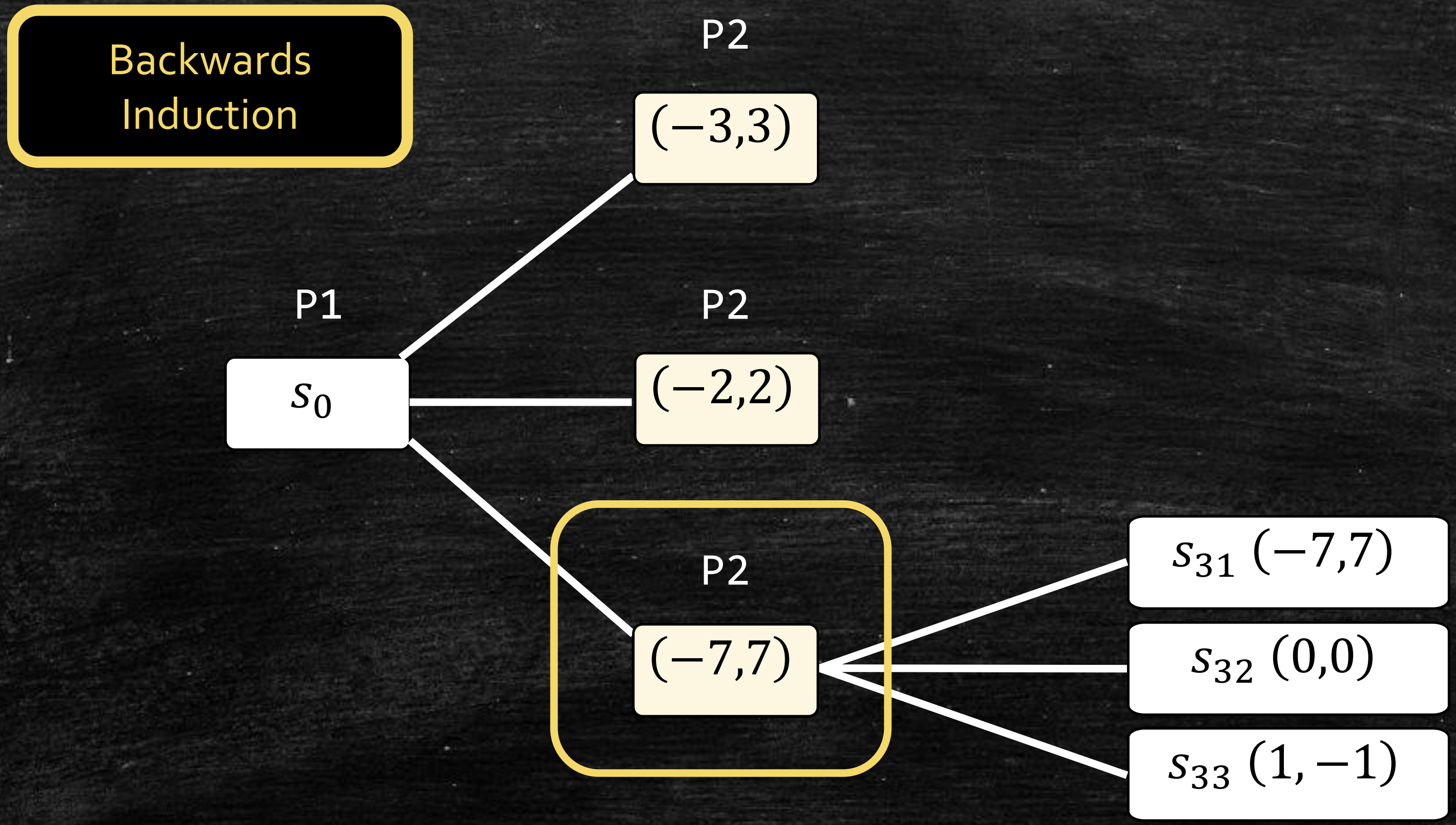
Minimax Play – Subgame Perfect Nash Equilibrium



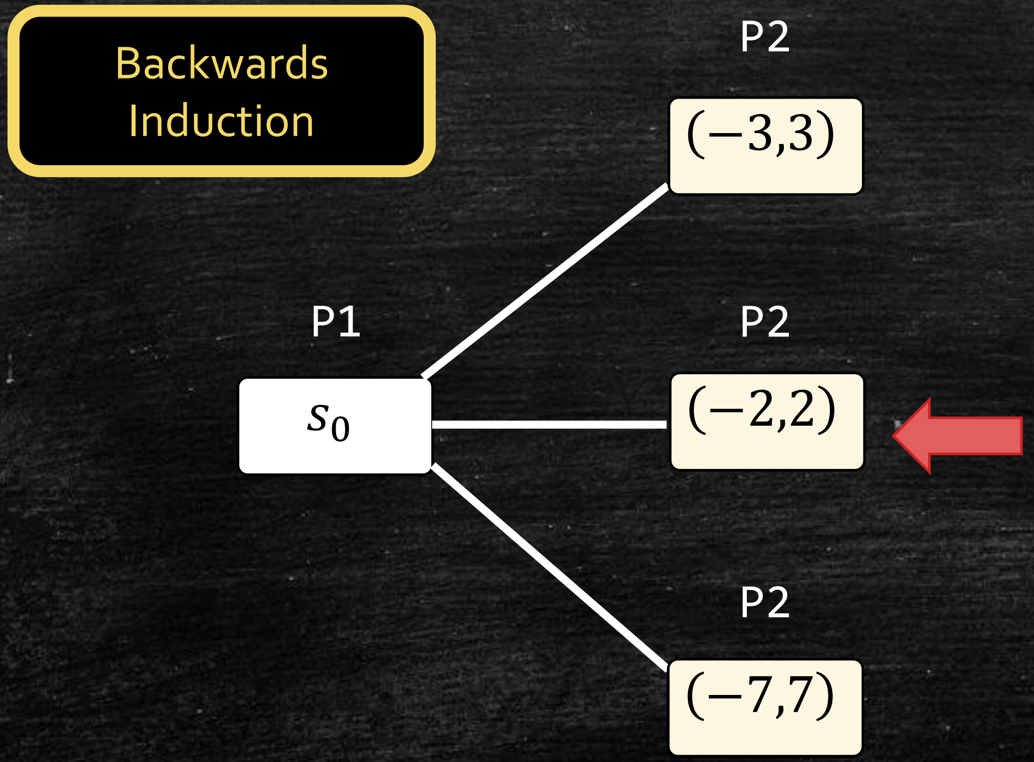
Minimax Play – Subgame Perfect Nash Equilibrium



Minimax Play – Subgame Perfect Nash Equilibrium

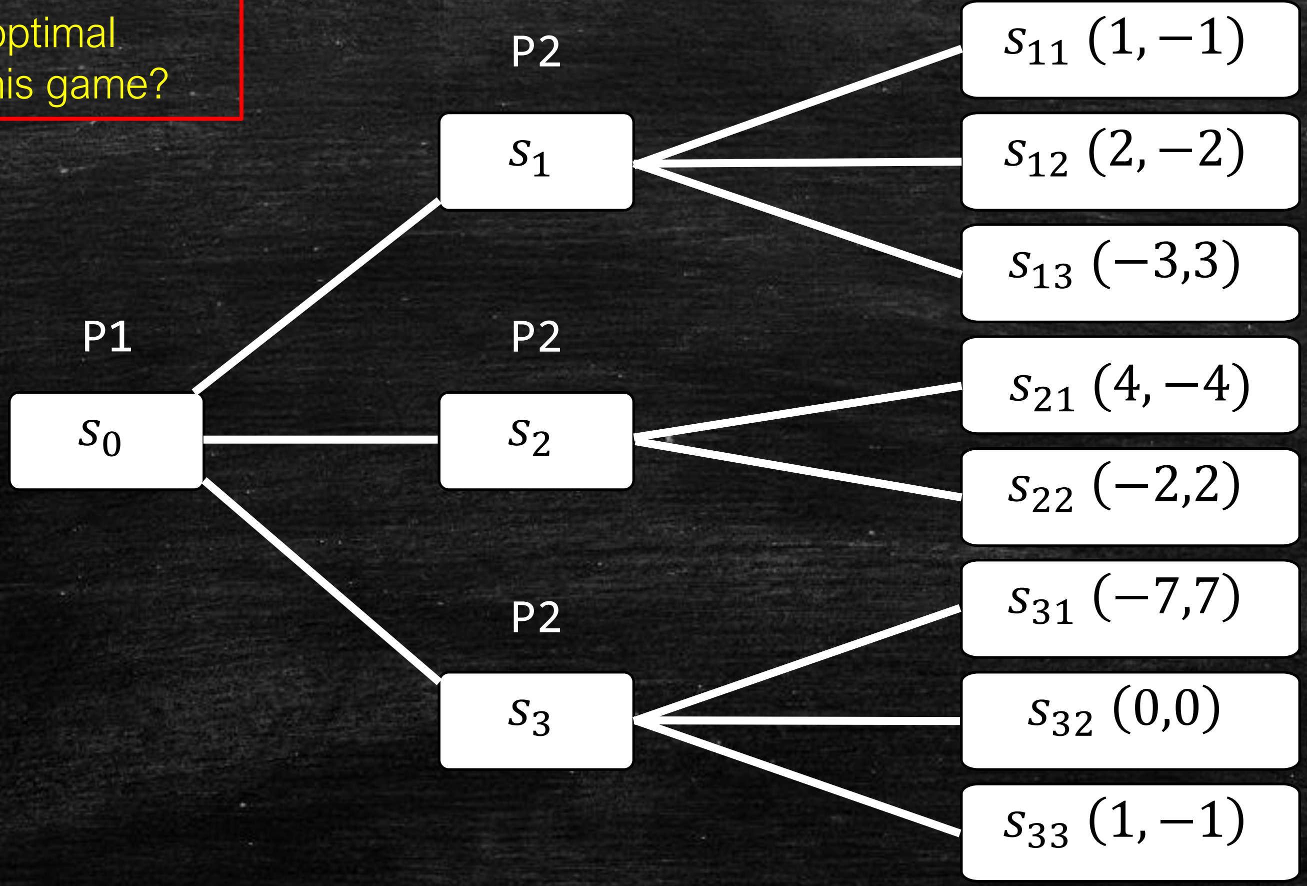


Minimax Play – Subgame Perfect Nash Equilibrium



Minimax Play – Subgame Perfect Nash Equilibrium

What are the optimal strategies in this game?



Minimax Algorithm Properties

- Complete?
  - Yes (if game tree is finite)
- Optimal?
  - Yes (optimal gameplay)
- Time
  - $O(b^m)$
- Space
  - $O(bm)$

- Minimax runs in time polynomial in tree size
- Returns a subgame perfect Nash Equilibrium
  - i.e., the best action at every node

Are we done?

Backwards Induction

- Game trees are massive
  - Chess has a massive game tree
    - 10^{123} nodes
  - In comparison, planet Earth has about 10^{50} atoms …
- Impossible to expand the entire tree
- Have to find ways to shrink the search tree
  - We've seen this before
  - Common theme in search

Questions about the Lecture?

- Was anything unclear?
- Do you need to clarify anything?

- Ask on Archipelago
  - Specify a question
  - Upvote someone else's question

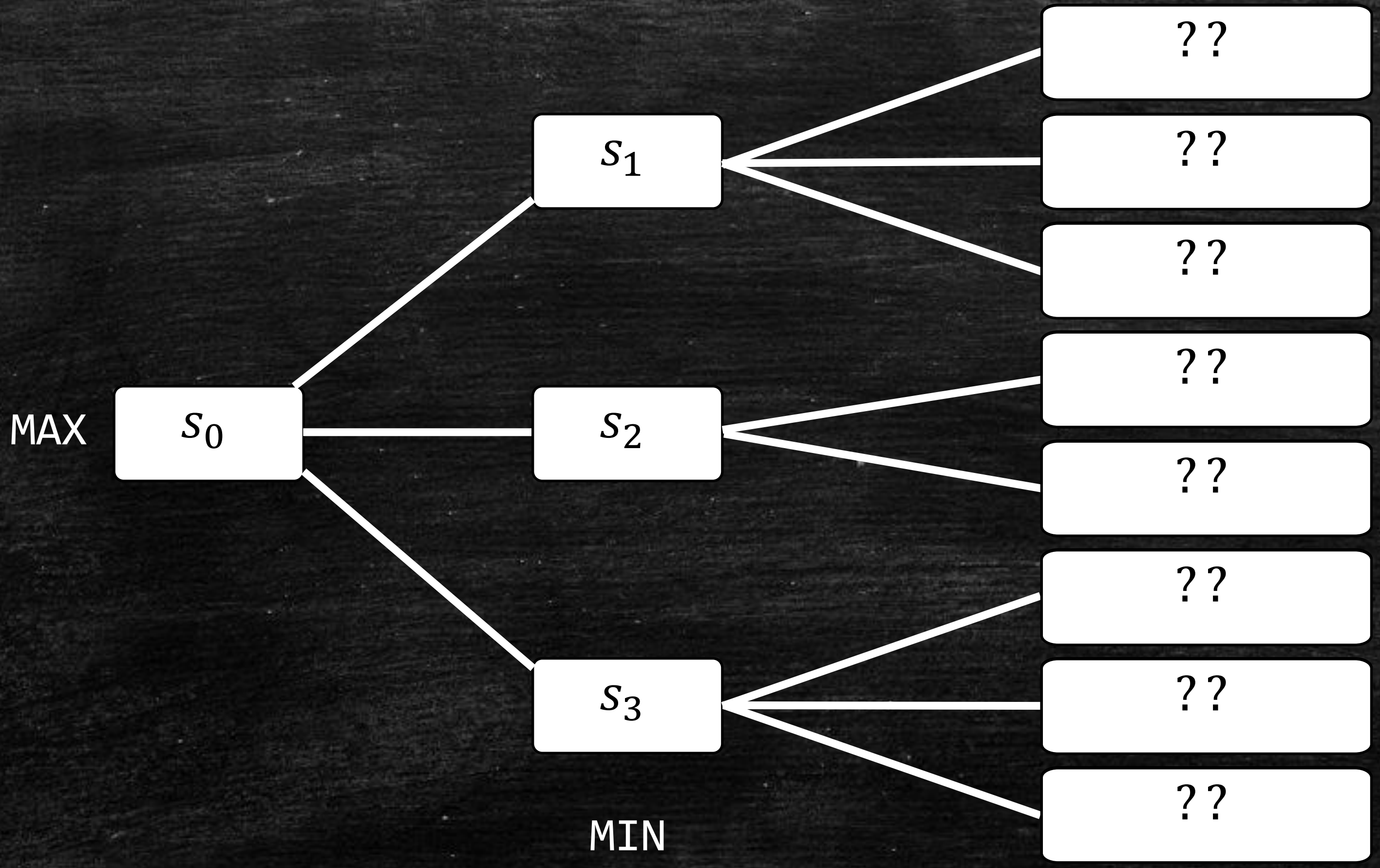
Invitation Link (Use NUS Email --- starts with E)
https://archipelago.rocks/app/resend-invite/64238273017

α-β Pruning

α-β Pruning - General Idea

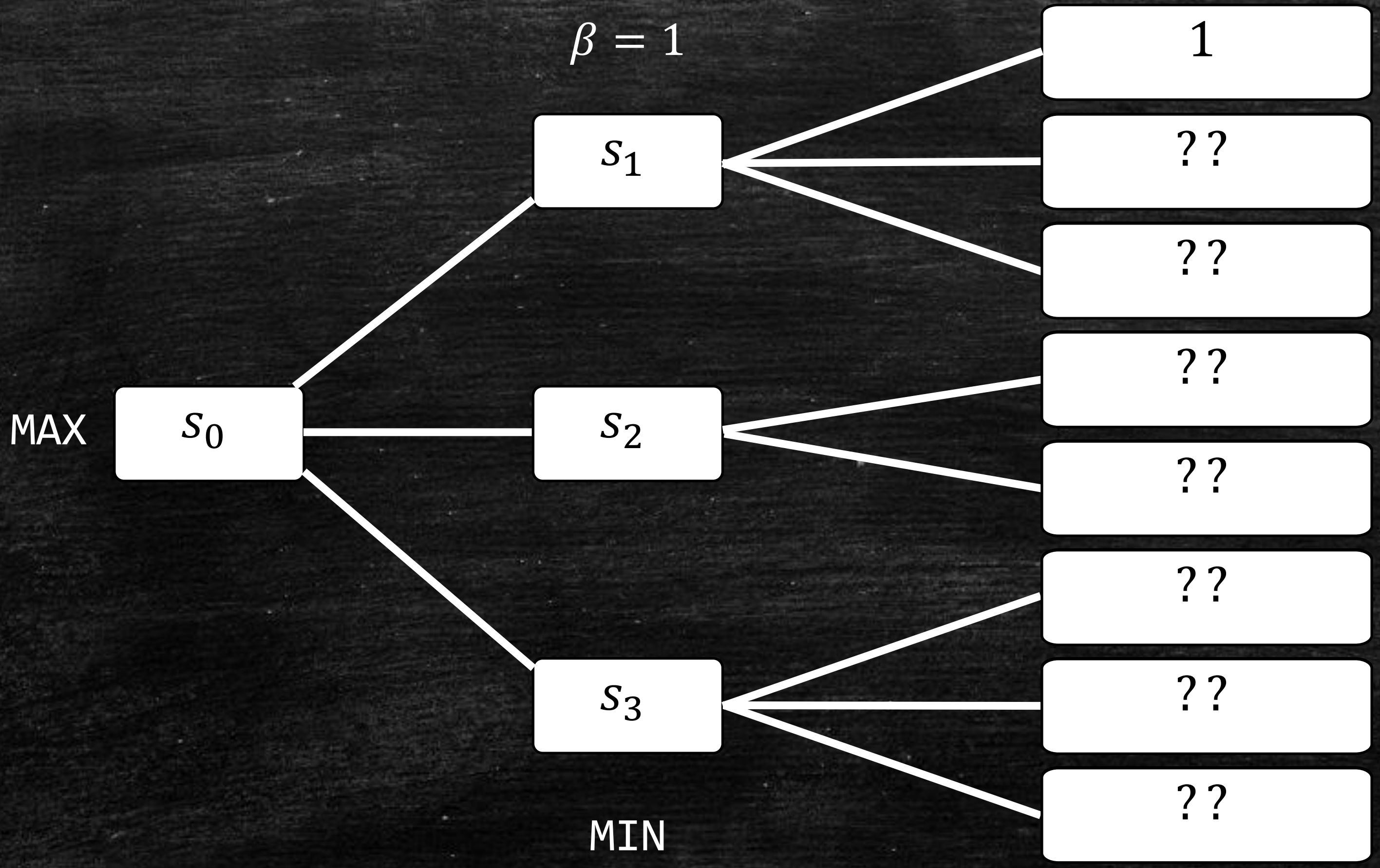
- Basic idea
  - Don't explore moves that would never be considered
- Maintain bounds on values seen thus far while searching
  - α bounds MAX's values
  - β bounds MIN's values
- Prune subtrees that will never affect Minimax decision

α-β Pruning – Example Trace



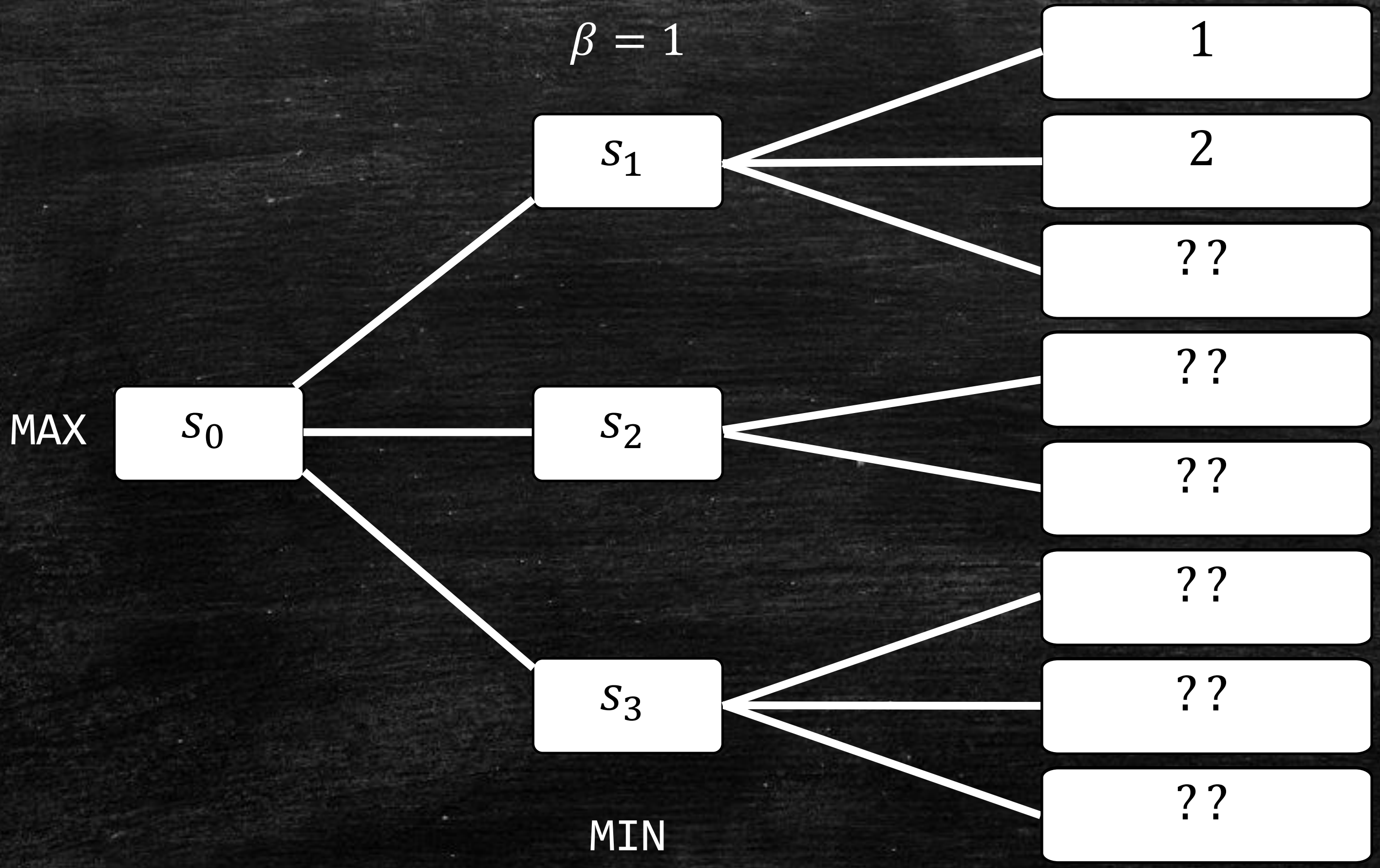
Note that from this point, all utility values will be specified in terms of the MAX player.

α-β Pruning – Example Trace



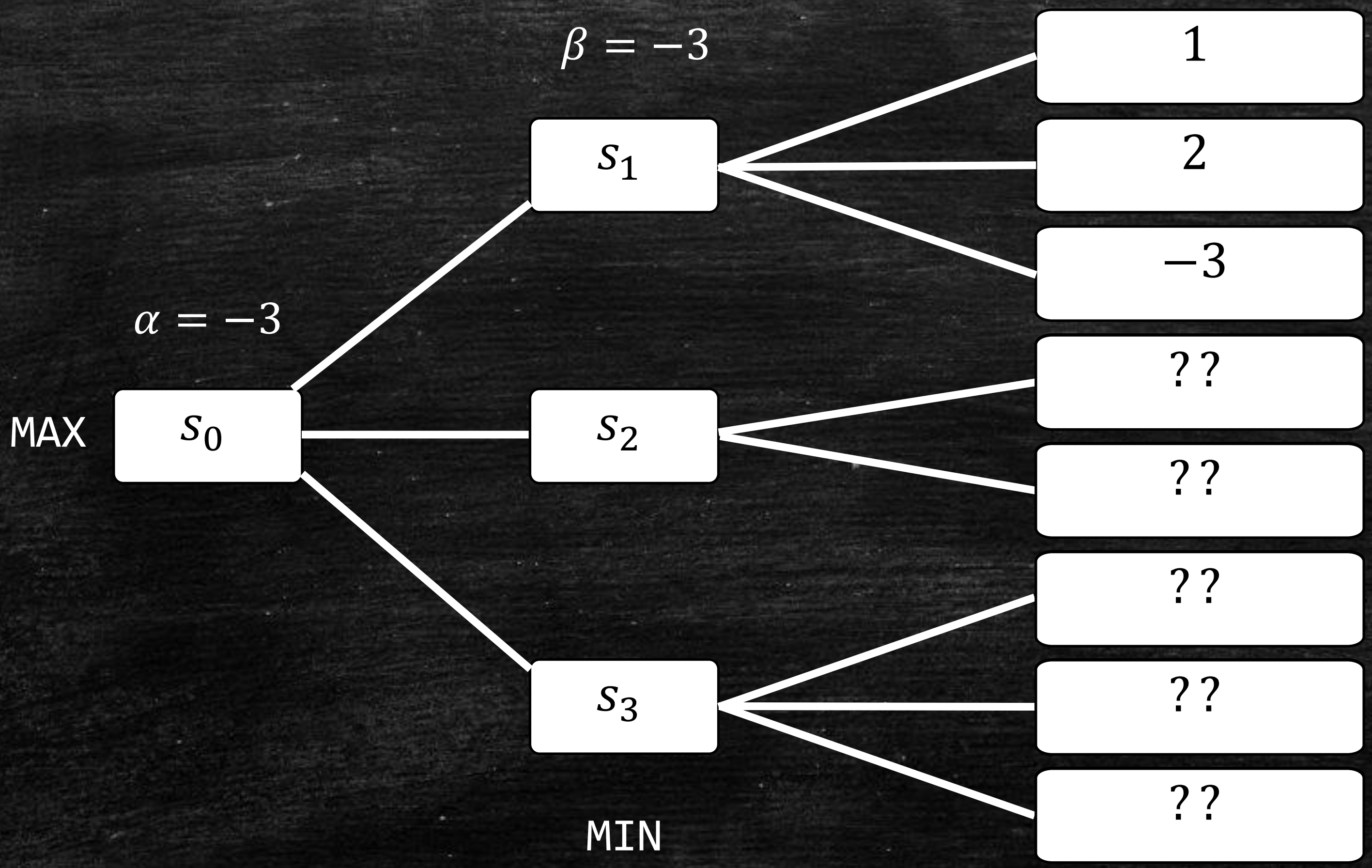
Note that from this point, all utility values will be specified in terms of the MAX player.

α-β Pruning – Example Trace



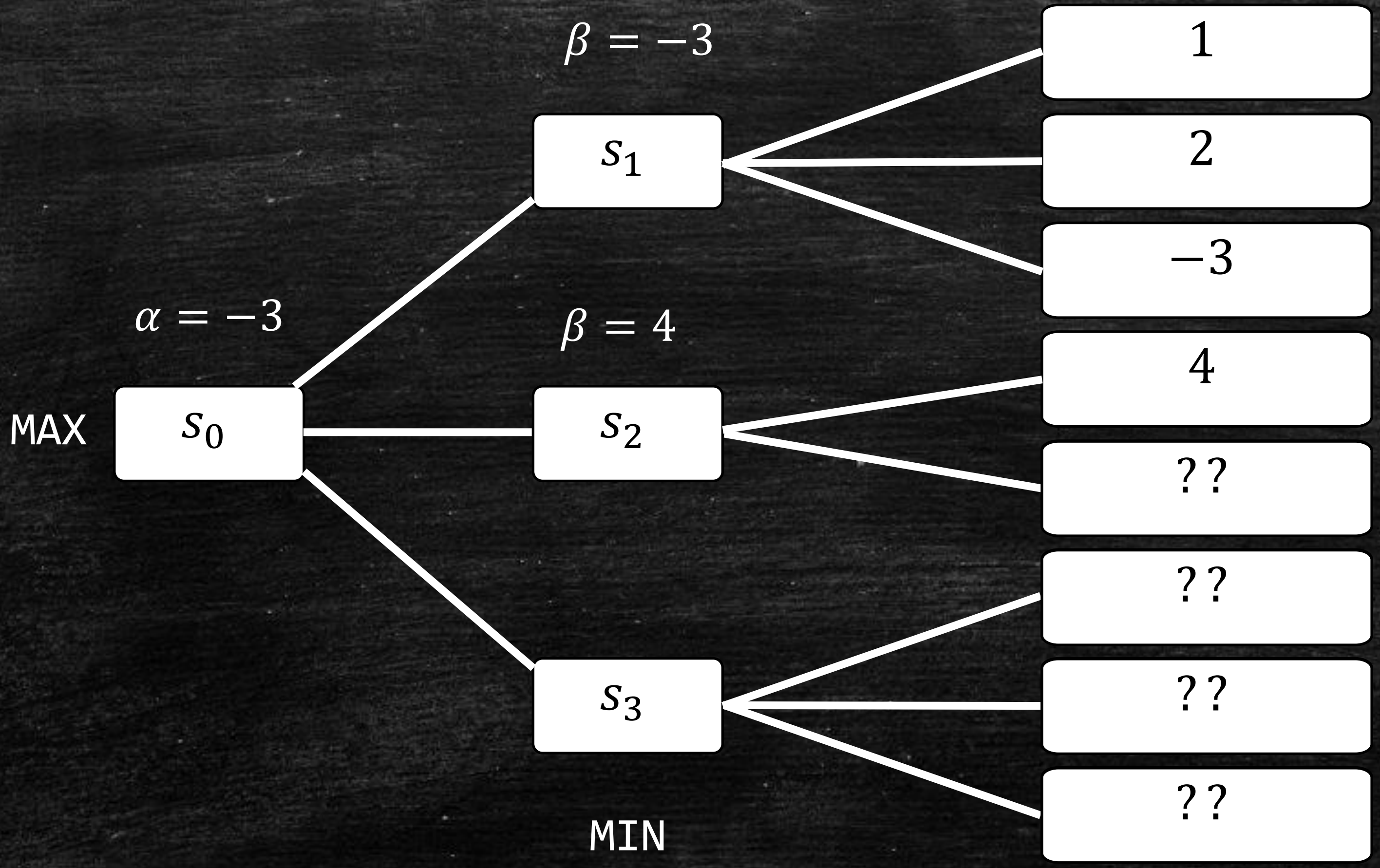
Note that from this point, all utility values will be specified in terms of the MAX player.

α-β Pruning – Example Trace



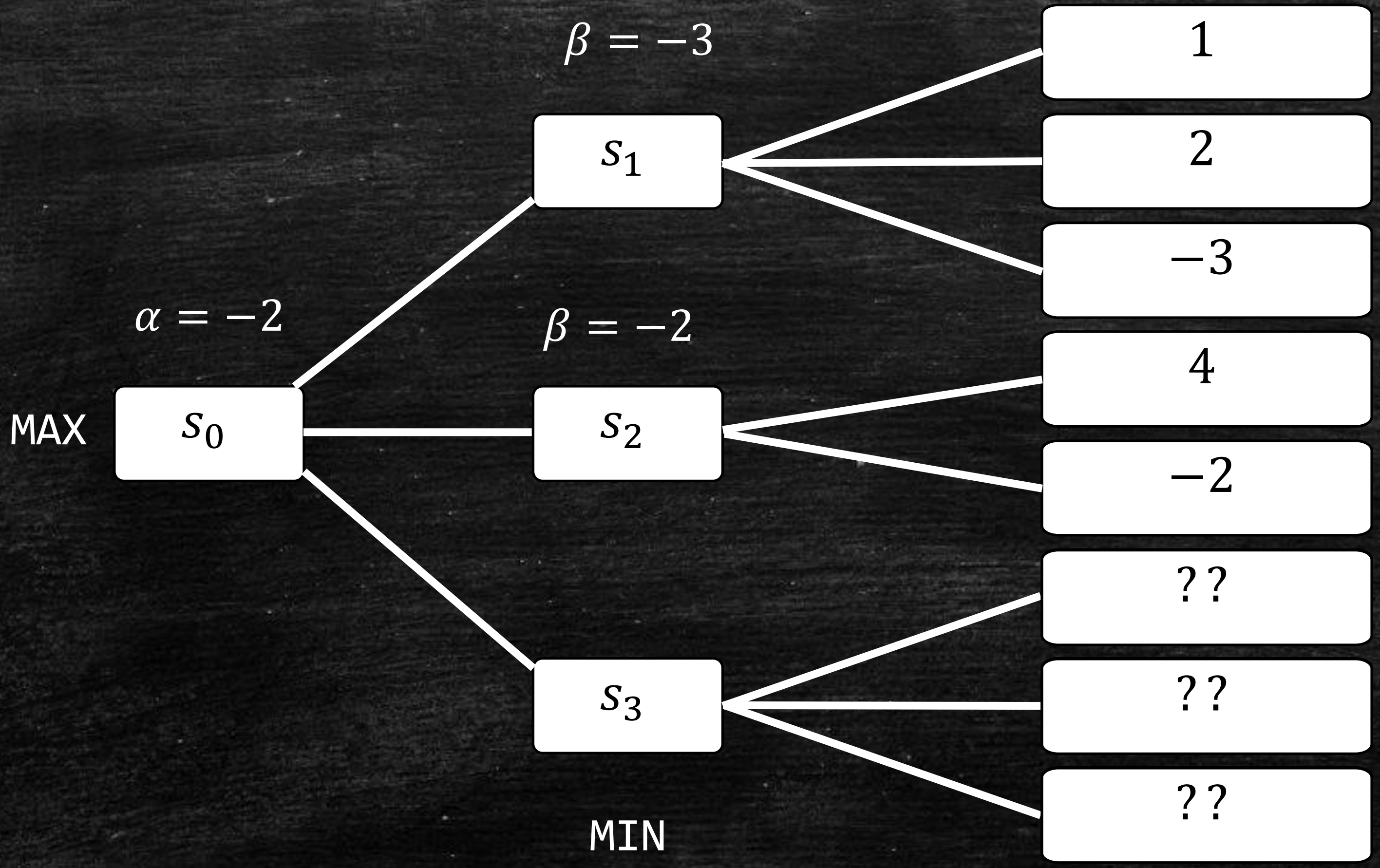
Note that from this point, all utility values will be specified in terms of the MAX player.

α-β Pruning – Example Trace



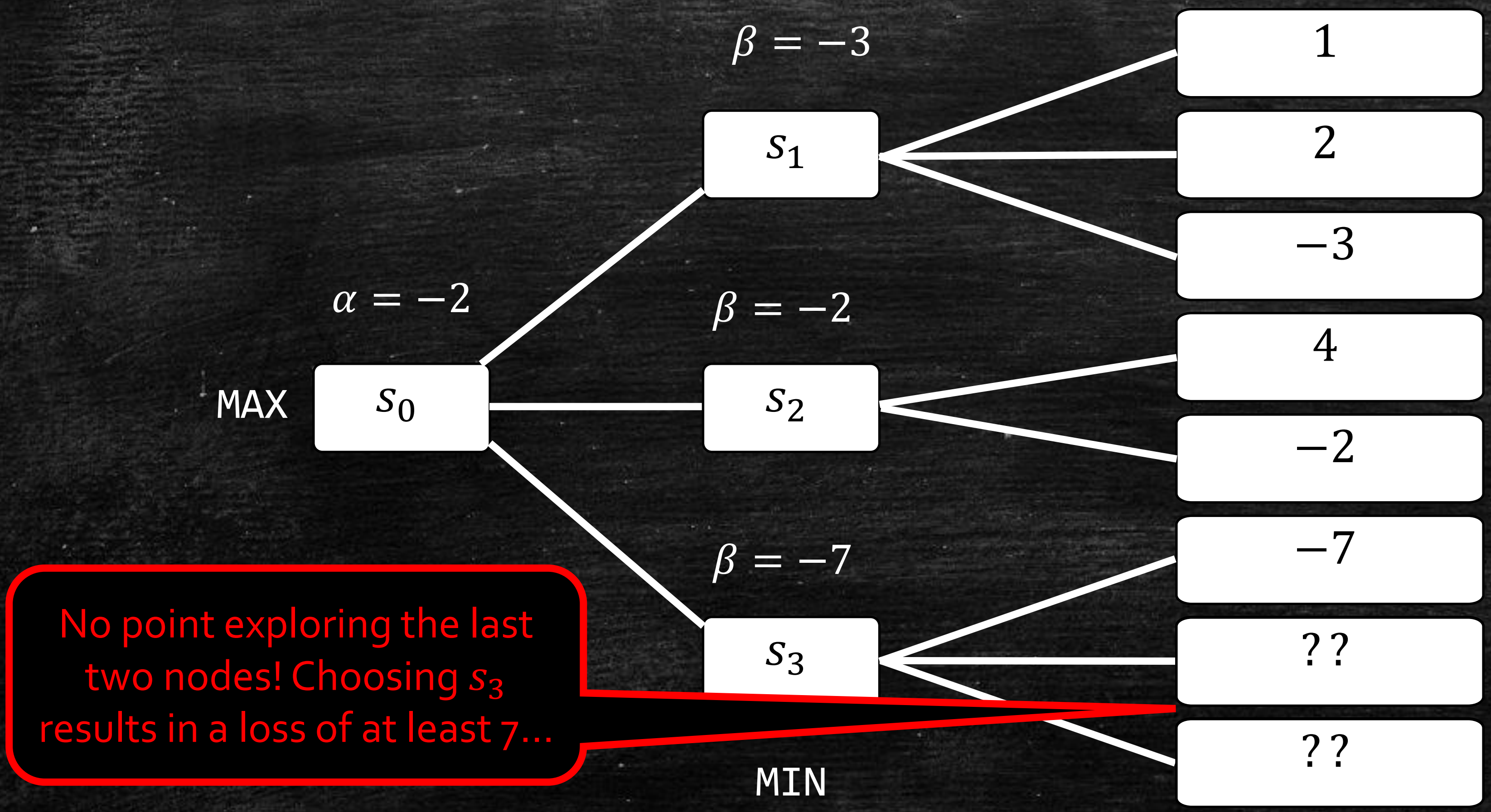
Note that from this point, all utility values will be specified in terms of the MAX player.

α-β Pruning – Example Trace



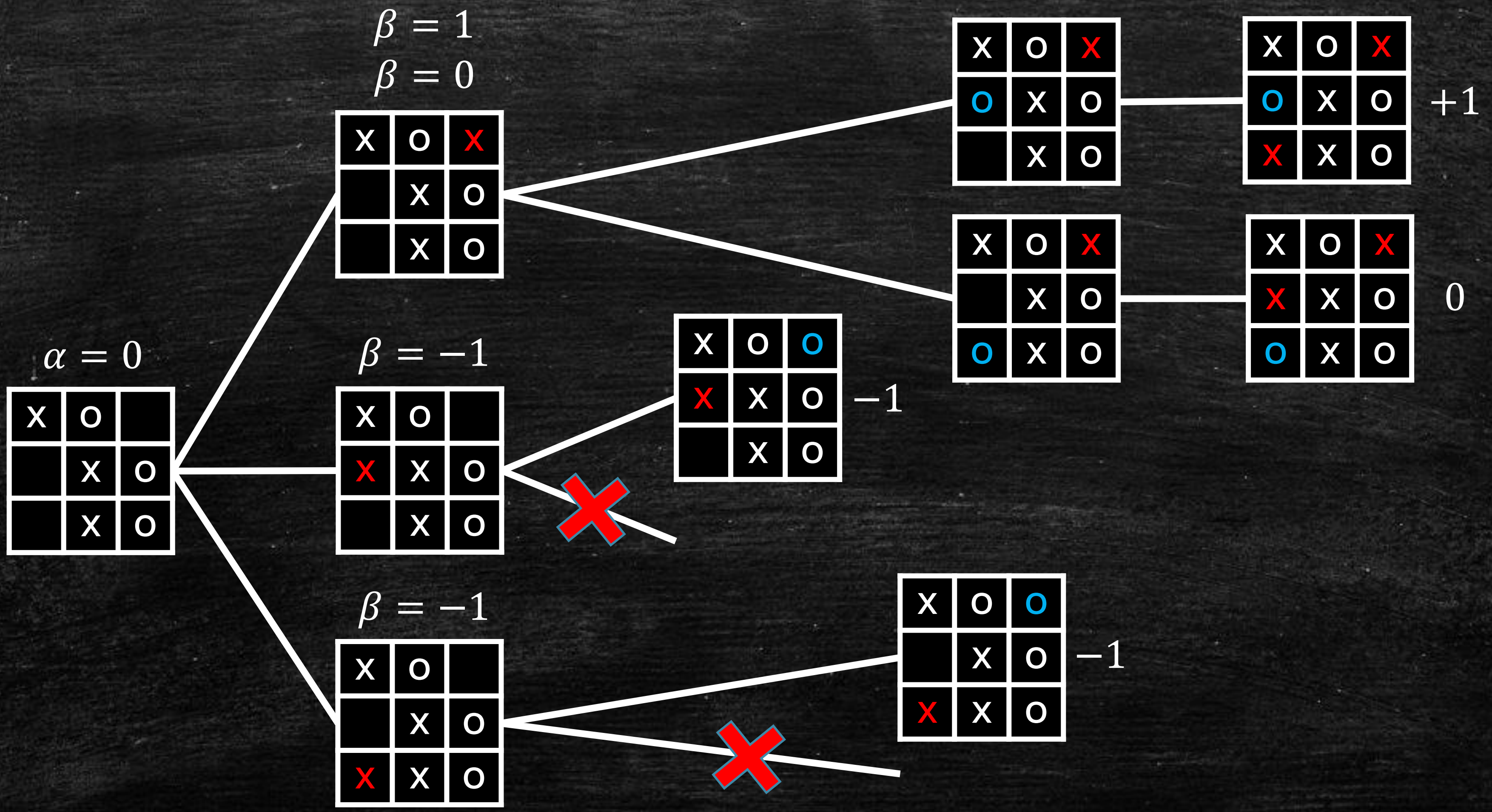
Note that from this point, all utility values will be specified in terms of the MAX player.

α-β Pruning – Example Trace



Note that from this point, all utility values will be specified in terms of the MAX player.

α-β Pruning – Tic-Tac-Toe Example



α-β Pruning

- MAX node n
  - α(n) = highest observed value found on path from n
  - Initially α(n) = -∞
- MIN node n
  - β(n) = lowest observed value found on path from n
  - Initially β(n) = +∞
- Pruning rules
  - Given a MIN node n, stop searching below n if
    - Some MAX ancestor i (of n) with $\alpha(i) \geq \beta(n)$
  - Given a MAX node n, stop searching below n if
    - Some MIN ancestor i (of n) with $\beta(i) \leq \alpha(n)$

MIN will choose β(n) or lower at n, but ancestor MAX will NEVER choose the subtree at n since at i, there is a better option with higher value α(i)

MAX will choose α(n) or higher at n, but ancestor MIN will NEVER choose the subtree at n since at i, there is a better option with lower value β(i)

α-β Pruning Analysis

- Pruning a branch never affects the final outcome
- Good move ordering improves effectiveness of pruning
  - "Perfect" ordering
    - Time complexity $O(b^{m/2})$
    - Good pruning strategies allow us to search twice as deep!
  - Example: Chess
    - Simple ordering gets you close to best-case result
      - Checks
      - Take pieces
      - Forward moves
      - Backwards move
- Expansion-order heuristics will improve the search
- Random ordering gives complexity $O(b^{3m/4})$ for $b < 1000$

Issue with α-β Pruning

- Original Problem
  - Most games have very large game trees
- Solution
  - α-β pruning can remove large parts of search trees
- Unresolved Issue
  - Maximum depth of tree
    - Backwards induction works backwards from terminal states
    - Still have to traverse to a terminal states
  - Standard solution – Heuristic Minimax
    - Cutoff test – e.g., depth limit (DLS)
    - Evaluation function – estimates expected utility of state

Heuristic Minimax

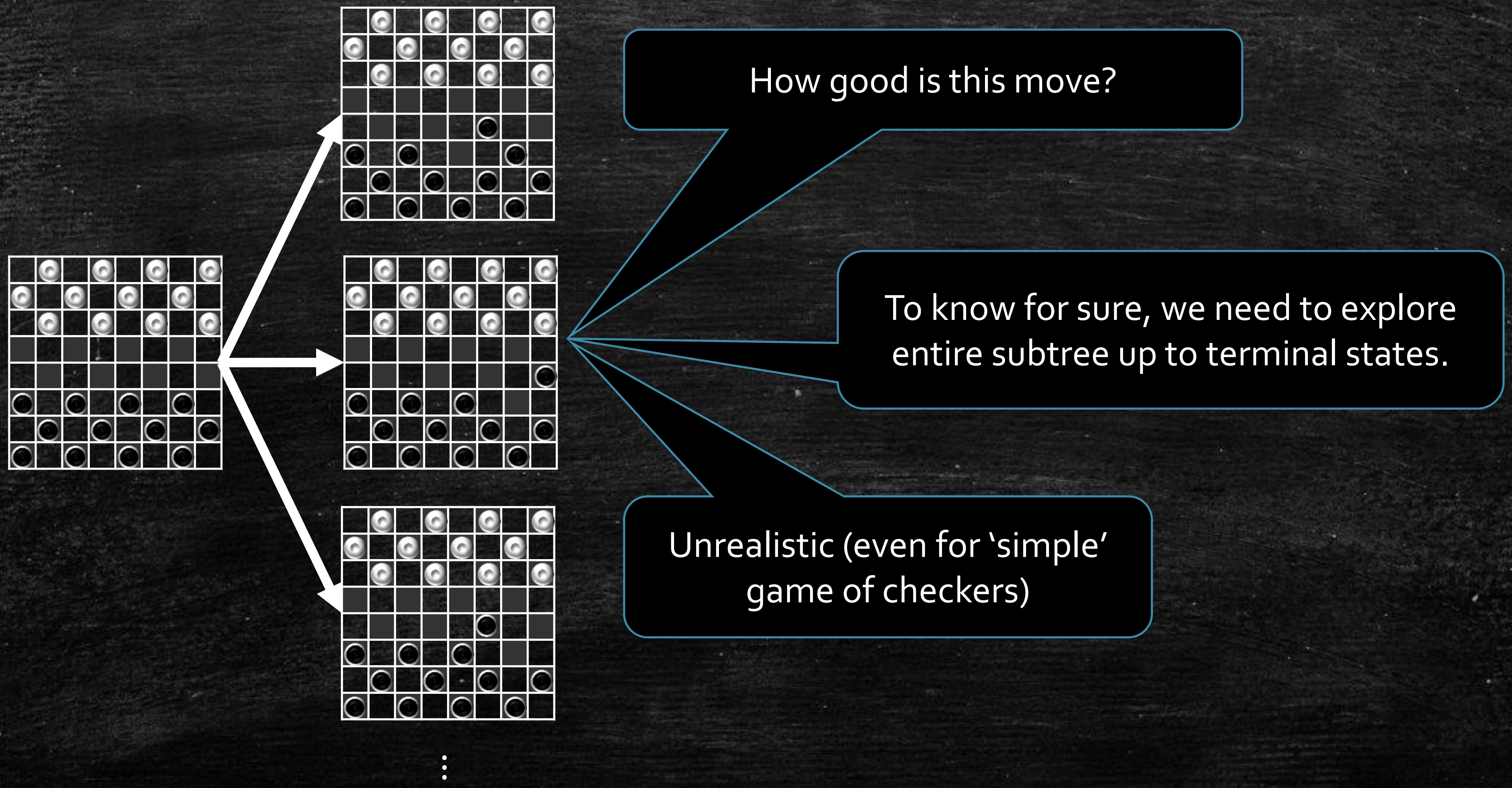
Heuristic Minimax Value

$$Minimax(s) = \begin{cases} Utility(s, \mathrm{MAX}) \text{ if Is}-\text{Terminal}(s) \\ \max_{a \in \mathrm{Actions}(s)} \mathrm{Minimax}\big(\mathrm{Result}(s,a)\big) \text{ if To}-\text{Move}(s) = \mathrm{MAX} \\ \min_{a \in \mathrm{Actions}(s)} \mathrm{Minimax}\big(\mathrm{Result}(s,a)\big) \text{ if To}-\text{Move}(s) = \mathrm{MIN} \end{cases}$$

$$H-Minimax(s, d) = \begin{cases} Eval(s, \mathrm{MAX}) \text{ if Cutoff}-\text{Test}(s,d) \\ \max_{a \in \mathrm{Actions}(s)} \mathrm{H}-\mathrm{Minimax}(\mathrm{Result}(s,a), d+1) \text{ if To}-\text{Move}(s) = \mathrm{MAX} \\ \min_{a \in \mathrm{Actions}(s)} \mathrm{H}-\mathrm{Minimax}(\mathrm{Result}(s,a), d+1) \text{ if To}-\text{Move}(s) = \mathrm{MIN} \end{cases}$$

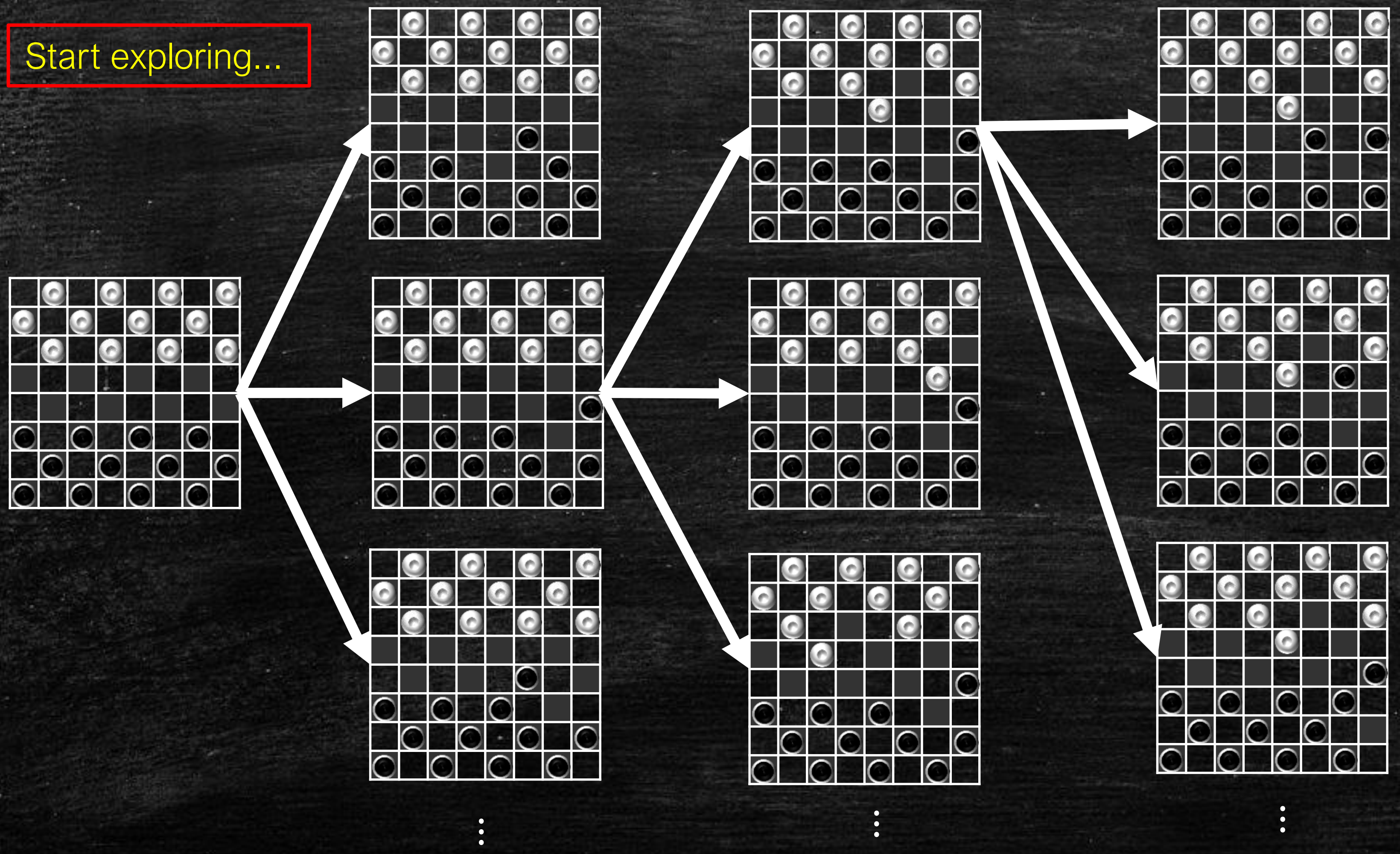
Run Minimax until depth d; then start using the evaluation function to choose nodes

Evaluation Functions – Checkers Example

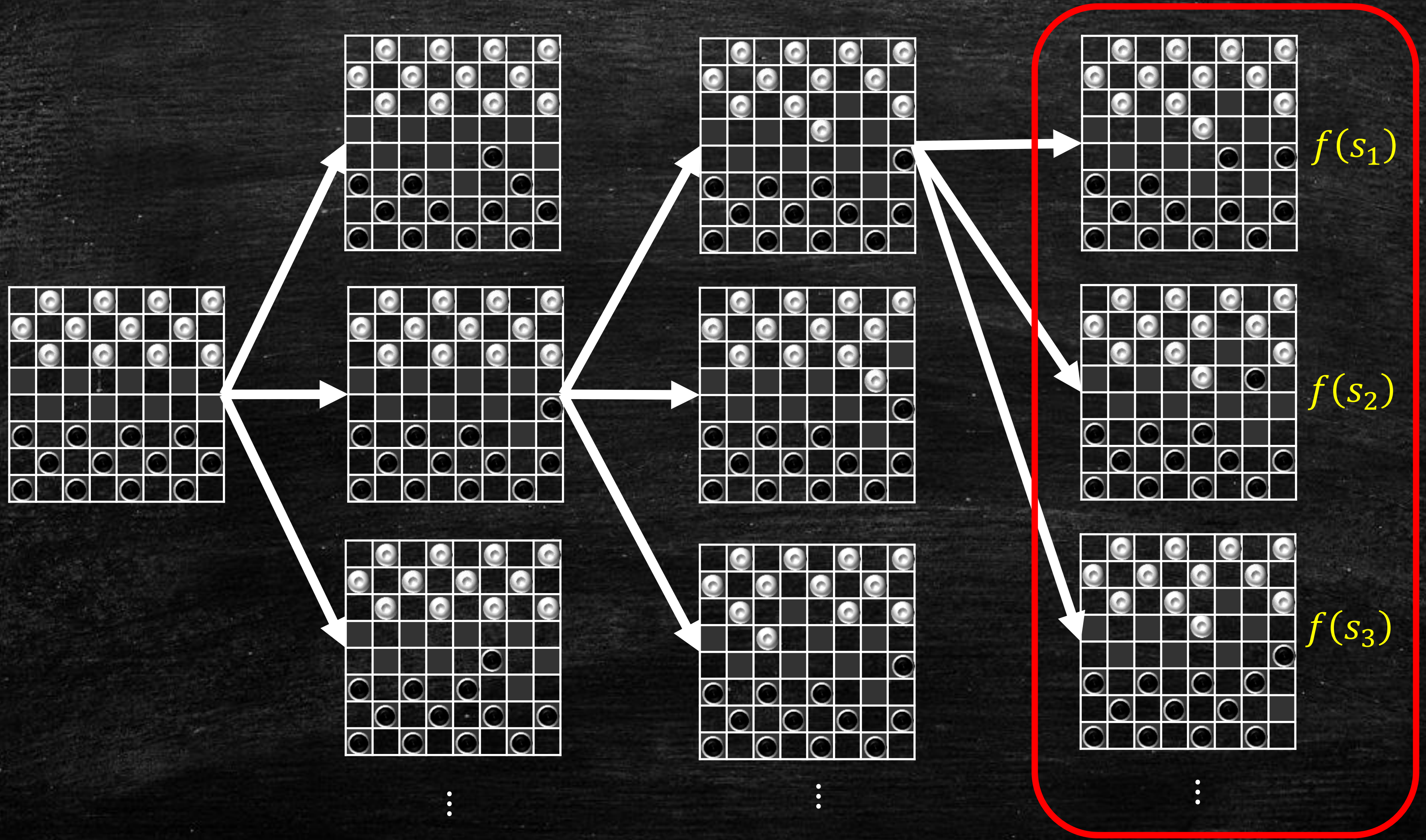


Evaluation Functions – Checkers Example

Start exploring...



Evaluation Functions – Checkers Example



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

Evaluation Functions

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

$$f(s) = \begin{cases} Utility(s, MAX) \text{ if } Is\text{–}Terminal(s) \\ 0 \; otherwise \end{cases}$$

No information on quality of non-terminal nodes

- Determine a function to estimate value that is strongly correlated to actual chances of winning
  - Modelling problem (similar to heuristic design problem from informed/local search)

Evaluation Functions

- Determine important features/variables
- Chess example
  - \# of pieces ($NPcs$)
  - \# of queens ($NQns$)
  - \# of controlled squares ($CtlSqs$)
  - \# of threatened opponent pieces ($ThrPcs$)
  - ...
- $f(n) = w_1 \times (NPcs) + w_2 \times (NQns) + w_3 \times (CtlSqs) + w_4 \times (ThrPcs)$

Determine values for w_1, ..., w_4

Cutting Off Search

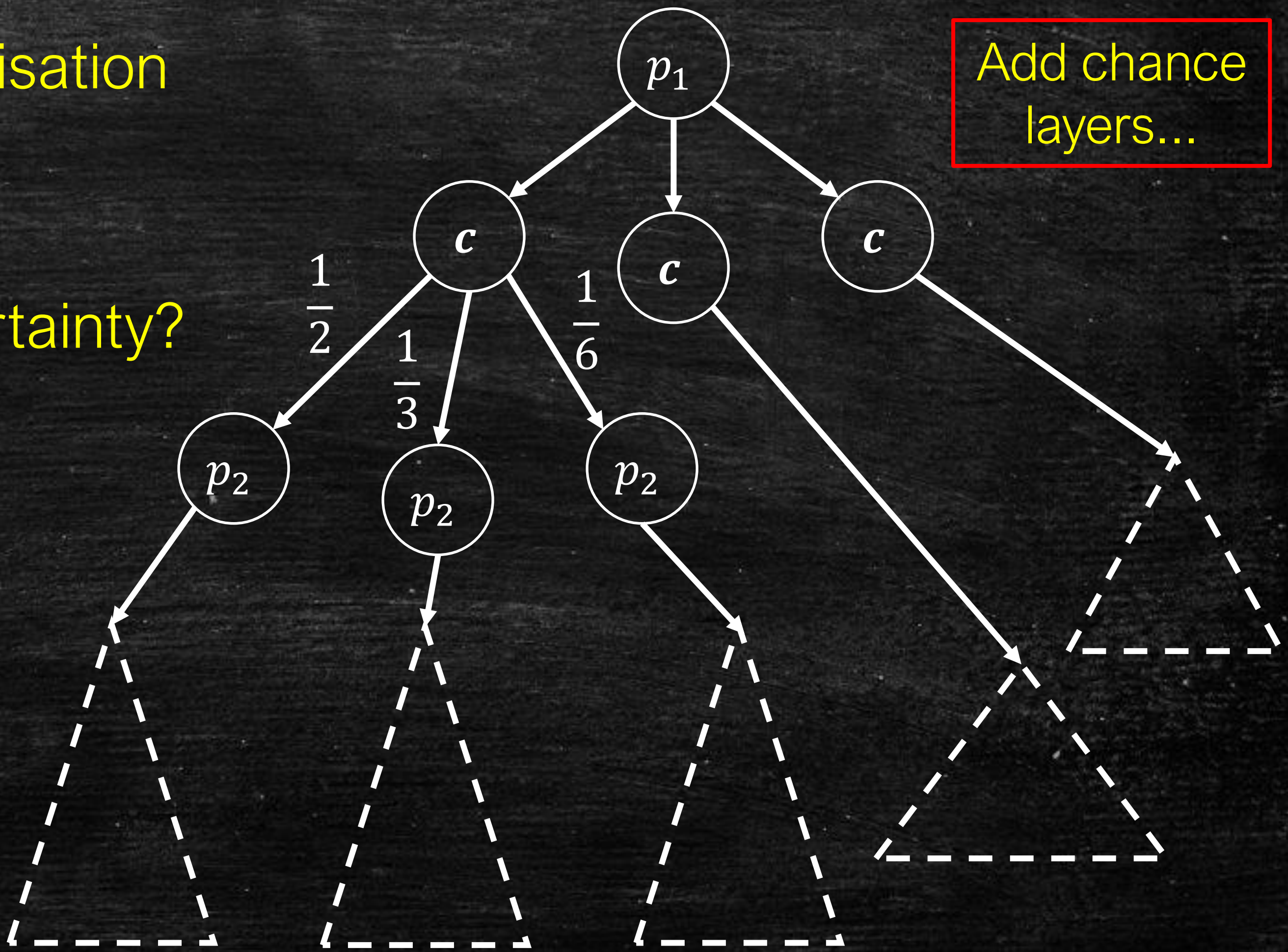
- Modify Minimax or α-β Pruning algorithms by replacing
  - Is-Terminal(s) with Cutoff-Test(s, d)
  - Utility(s, p) with Eval(s, p)

- Can replace DLS strategy with IDS

Stochastic Games

- Many games have randomisation
  - Settlers of Catan
  - Poker
- How do we deal with uncertainty?
  - Can still use Minimax
  - Search tree is larger

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



Add chance layers...

Questions about the Lecture?

- Was anything unclear?
- Do you need to clarify anything?
- Ask on Archipelago
  - Specify a question
  - Upvote someone else's question

Invitation Link (Use NUS Email --- starts with E)
https://archipelago.rocks/app/resend-invite/64238273017