Lecture 10

CS 6380 Artificial Intelligence

Adversarial Search and Games

Department of Computer Science and Engineering

Indian Institute of Technology Madras (IITM), India

Some material based on lecture slides for CSE 571: Artificial Intelligence by Siddharth Srivastava, ASU.

Logistics and Recap

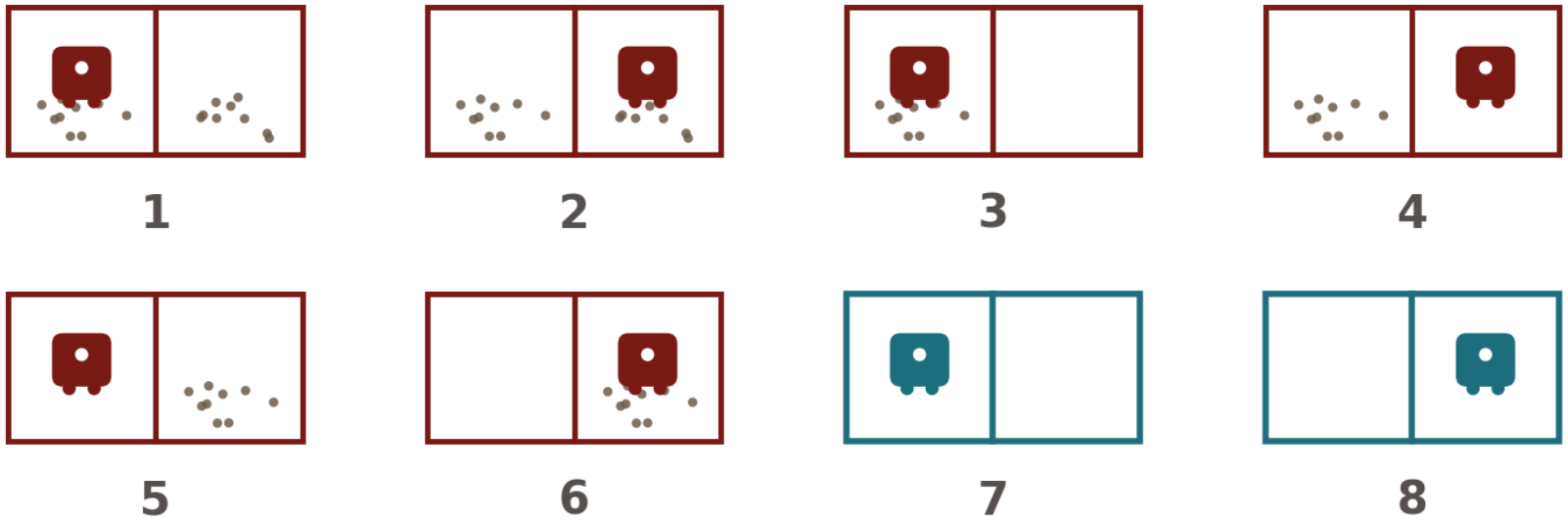
Talk Announcement

- **Title:** Towards Reliable Autonomy: From Individual Agents to Autonomous Teams
- **Date:** August 21, 2026
- **Time: 11**:00 AM – 12:00 PM
- Location: SSB 334

Dr. Sandhya Saisubramanian
Oregon State University, USA
https://www.sandhyasai.com/

What the Agent Knows Instead

- With no sensors the agent still knows the map and what each action does
- What it does not know is which of these states it is actually in
- So it reasons about a belief state, the set of states it could be in



states 7 and 8 are the goal states

The Belief State Space

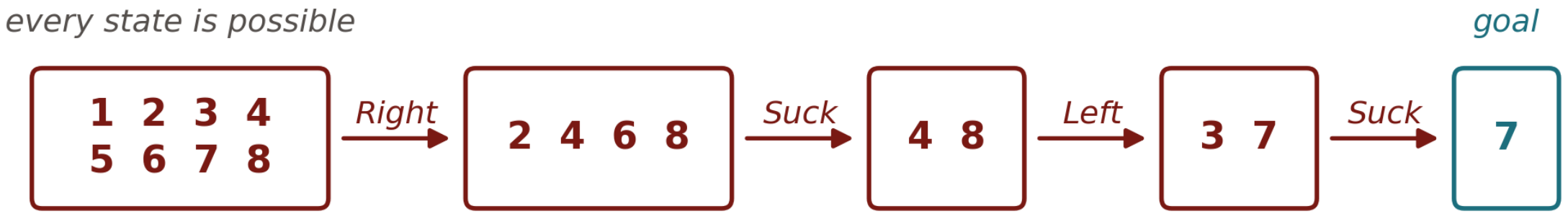
- Belief states inherit everything from the underlying problem, an initial state, actions, a transition model and a goal test

8 physical states, 256 belief states, 12 reachable

- The goal test asks whether every state in the belief state is a goal, since the agent has to be certain
- The blow up is the price of not knowing, and the reachable part is usually far smaller than the worst case

Coercing the World

- The agent cannot sense, so it acts to force the world into a configuration it can be sure of
- Right drives the agent to the right wall from every starting state at once
- The belief state shrinks until every state left in it is a goal



It Is Ordinary Search Again

- Belief states are just states, so breadth first, uniform cost and A star all apply unchanged
- The solution is a sequence of actions, because a sensorless agent has nothing to branch on

A plan for a belief state solves every subset of it

- So you can solve one physical state first, then test that plan against the rest and repair it as needed
- The real difficulty is size, since each node in the search is a set rather than a single state

Partial Observability

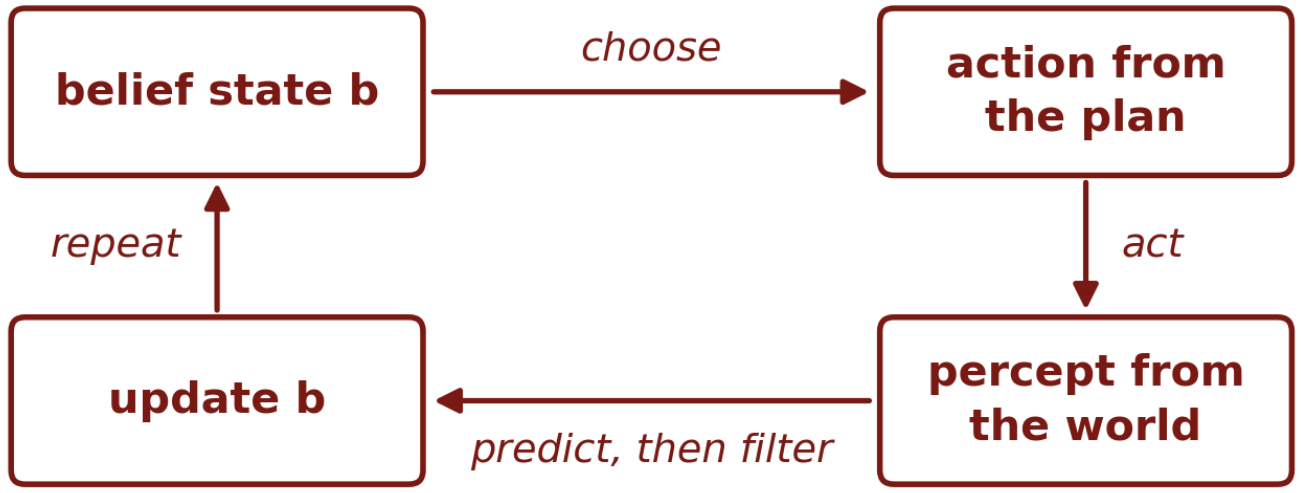
- Now the agent has sensors, but they do not identify the state
- In the local sensing vacuum world the agent sees its own square and nothing else

A percept rules possibilities out, it does not pin the state down

- Fully observable and sensorless are the two ends of one scale, and everything real lives in between

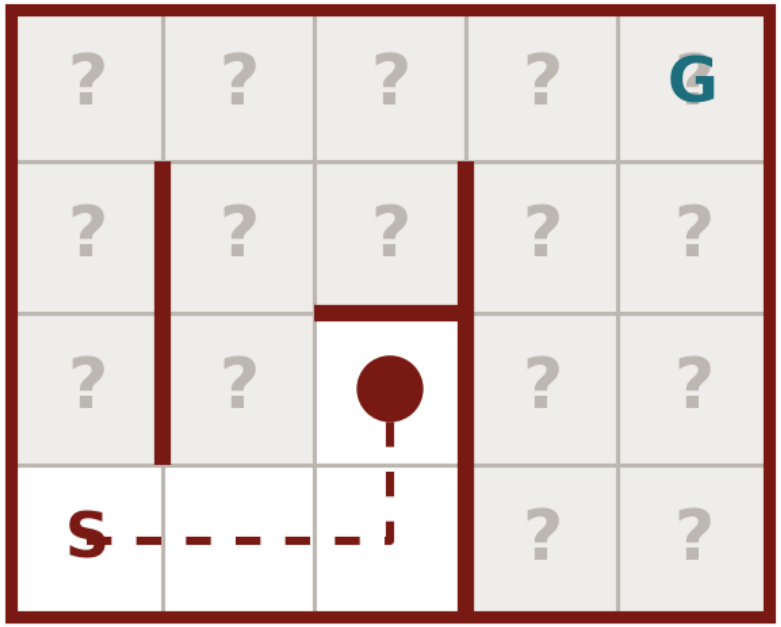
An Agent for Partially Observable Environments

- The agent executes its plan and maintains its belief state as it goes
- After each action it predicts, then filters by the percept it actually received
- This is state estimation, and the same recursive update returns in filtering later in the book



Online Search Problems

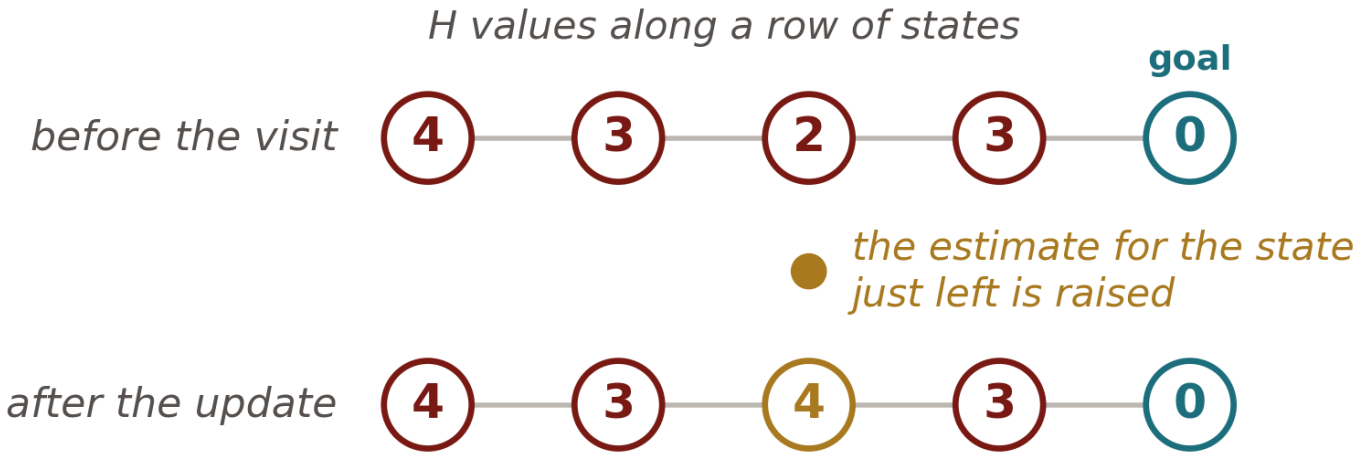
- The agent knows the actions available in the state it currently occupies
- It learns the cost of an action only after taking it
- It can tell whether the state it is in is a goal
- It cannot see where an action leads until it goes there, so it must act in order to learn



the agent knows only the squares it has walked

Learning While Acting

- Store an estimate H(s) of the cost to reach a goal from each state visited
- On leaving a state, update its estimate to the cost of the best action plus the estimate of where that action leads
- Untried actions are assumed cheap, so the agent explores what it has not seen
- Repeated visits raise the estimates in a local minimum until it fills in and the agent walks out



Adversarial Search and Games

Chapter 5, where the other agent is planning against us

Games as Multiagent Environments

- Competitive environments, in which the agents' goals are in conflict, give rise to adversarial search problems, better known as games

`two-player, turn-taking, zero-sum, perfect information`

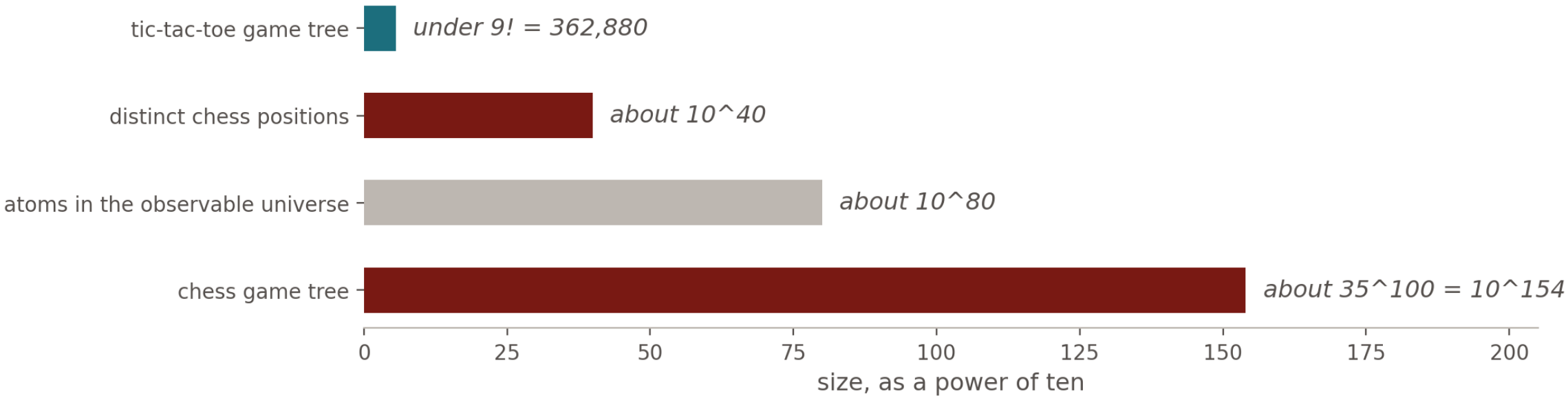
- Zero-sum means the payoffs always add to the same total, so whatever one player gains the other loses, and that opposition is what makes it adversarial
- Games appeal to AI because a game state is easy to represent and the legal moves are few, precisely defined and completely known
- Physical games such as croquet have messy rules and huge action spaces, and apart from robot soccer they have drawn little attention

Why Games Are Hard

- Chess has a branching factor near 35 and runs to about 100 plies, so its game tree holds on the order of 10^154 nodes

10^40 distinct positions, 10^154 nodes in the tree

- So the agent has to make some decision when the optimal decision is out of reach, and make it before the clock runs out



A Game as a Search Problem

- S0 the initial state, and PLAYER(s), which says whose turn it is to move
- ACTIONS(s) and RESULT(s, a), the legal moves and the transition model, exactly as in every search problem so far
- TERMINAL-TEST(s), true when play has ended, and UTILITY(s, p), the payoff to player p in that terminal state

`chess: +1 win, 0 loss, 1/2 draw, and every game totals 1`

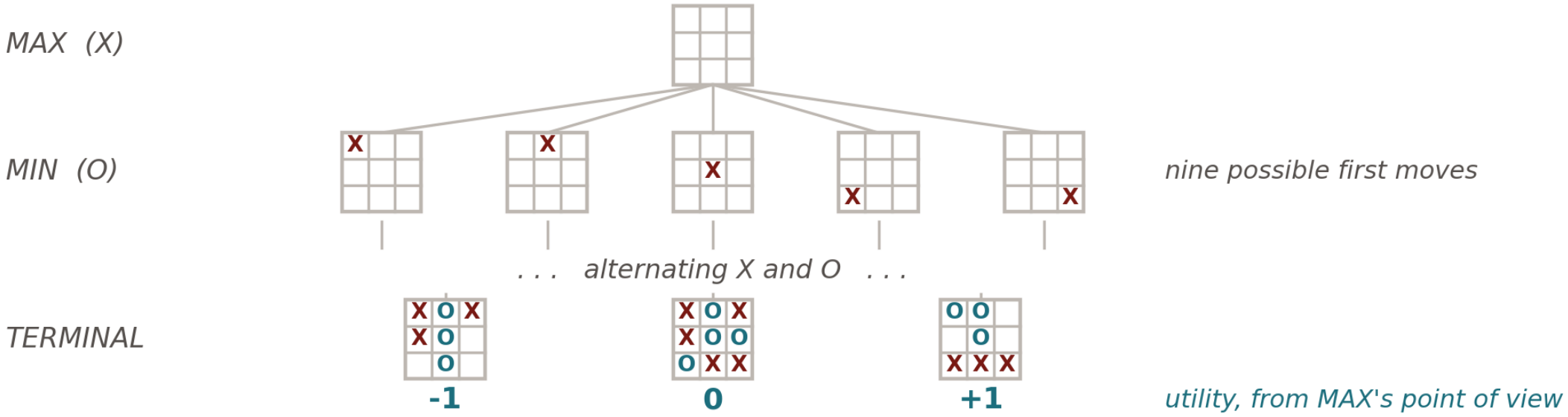
- MAX moves first and MIN replies, and one such half-move is called a ply

S0 *how the board starts*	**PLAYER(s)** *whose turn it is*	**ACTIONS(s)** *the legal moves*
RESULT(s, a) *the board after a move*	**TERMINAL-TEST(s)** *has play ended?*	**UTILITY(s, p)** *what the ending is worth*

highlighted: the two that are new in a game

The Game Tree

- The initial state with ACTIONS and RESULT defines the game tree, whose nodes are game states and whose edges are moves
- Utility is written from MAX's point of view, so high is good for MAX and bad for MIN, which is where the two names come from
- Tic-tac-toe has under 362,880 terminal nodes, chess over 10^40, so for chess the game tree is a theoretical object we can never build



Optimal Decisions in Games

- In an ordinary search problem the answer is a sequence of actions that reaches a goal
- Here MIN gets to reply, so any fixed sequence can be refuted
- MAX needs a contingent strategy, a first move plus an answer to every reply MIN might make

Ordinary search

a sequence of actions leading to a goal

Adversarial search

a move for every reply the opponent might make

MIN gets a vote, so a fixed plan is not enough

The Minimax Value

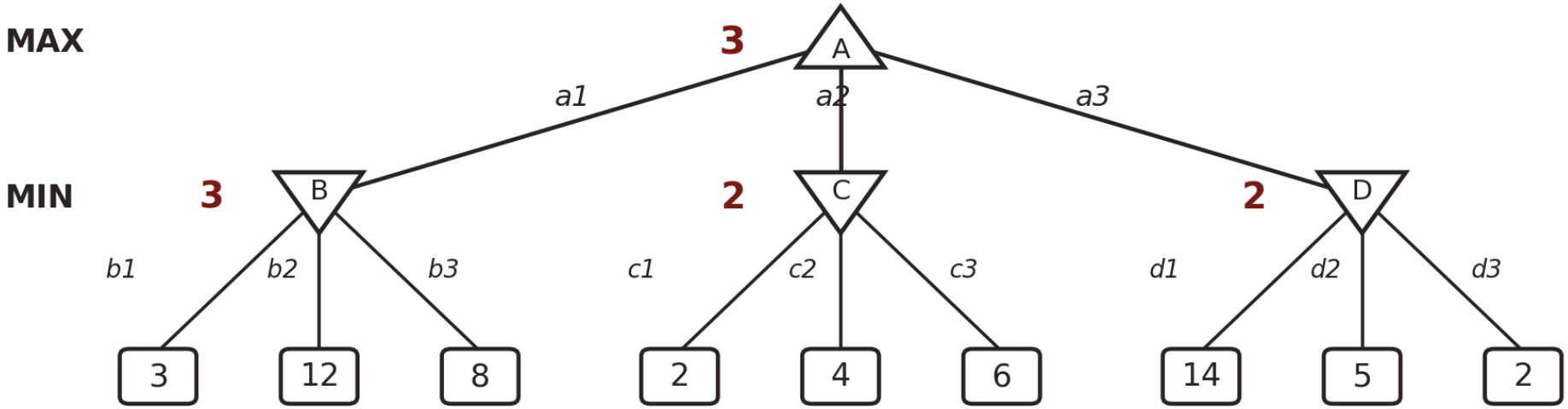
- The minimax value of a node is the utility for MAX of being in that state, assuming both players play optimally to the end
- For a terminal state it is simply the utility, and every other value is built up from those
- MAX moves to the successor of maximum value and MIN moves to the successor of minimum value

```
MINIMAX(s) =
    UTILITY(s)                                  if TERMINAL-TEST(s)
    max over a of MINIMAX(RESULT(s,a))          if PLAYER(s) = MAX
    min over a of MINIMAX(RESULT(s,a))          if PLAYER(s) = MIN
```

the value of a node when both sides play optimally from there on

A Two Ply Game Tree

- This game ends after one move each, so the tree is two plies deep
- B backs up the minimum of 3, 12 and 8, which is 3, and C and D both back up 2
- The root backs up the maximum of 3, 2 and 2, so a1 is the minimax decision



What Minimax Assumes

- The definition maximises the worst case, because it credits MIN with playing perfectly
- If MIN plays badly, MAX does at least as well as the minimax value and usually better
- Strategies aimed at a weak opponent can beat it faster, and they lose ground against a strong one

A perfect opponent

minimax is exactly right and cannot be improved on

A fallible opponent

minimax still does at least as well, often better

it maximises the worst case, so nothing worse can happen

The Minimax Search Algorithm

- The recursion runs all the way down to the leaves, and the values are backed up as it unwinds
- MAX-VALUE and MIN-VALUE are the two cases of the definition, written out and calling each other

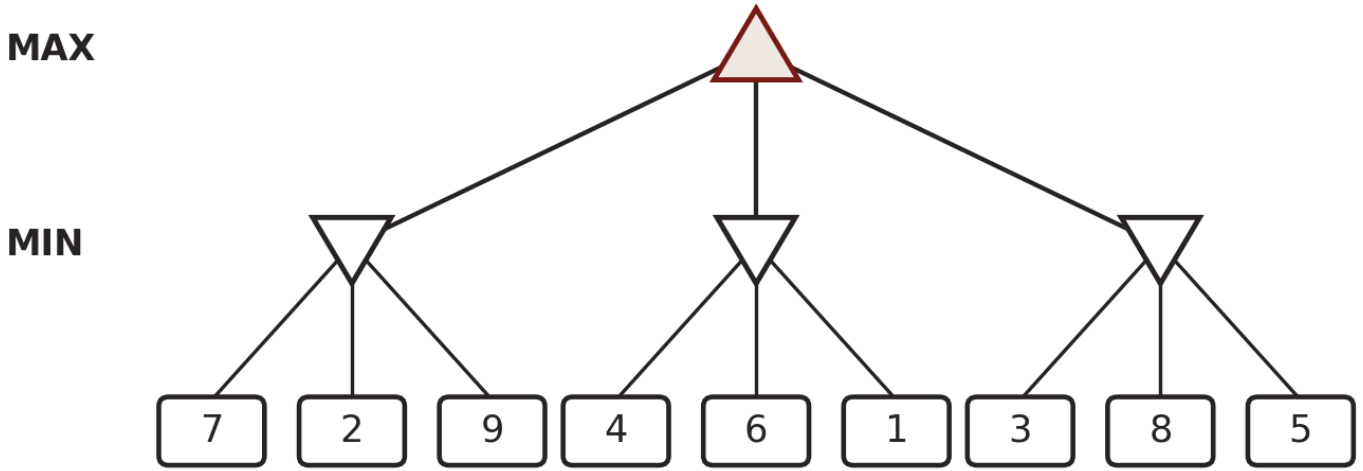
```
MAX-VALUE(state)
if TERMINAL-TEST(state) return UTILITY(state)
v <- minus infinity
for each a in ACTIONS(state)
    v <- MAX(v, MIN-VALUE(RESULT(state,a)))
return v
```

```
MIN-VALUE(state)
if TERMINAL-TEST(state) return UTILITY(state)
v <- plus infinity
for each a in ACTIONS(state)
    v <- MIN(v, MAX-VALUE(RESULT(state,a)))
return v
```

the two routines call each other all the way down to the leaves

Question 1

What is the minimax value at the root of this tree



left to right generation order

A 1

B 2

C 3

D 9

The Cost of Minimax

- A complete depth first exploration of the game tree, so time is exponential in the depth
- Space is O(bm) generating all actions at once, or O(m) generating them one at a time
- For chess this is hopeless, and yet minimax stays the standard every practical method is measured against

Time

O(b to the power m)

Space

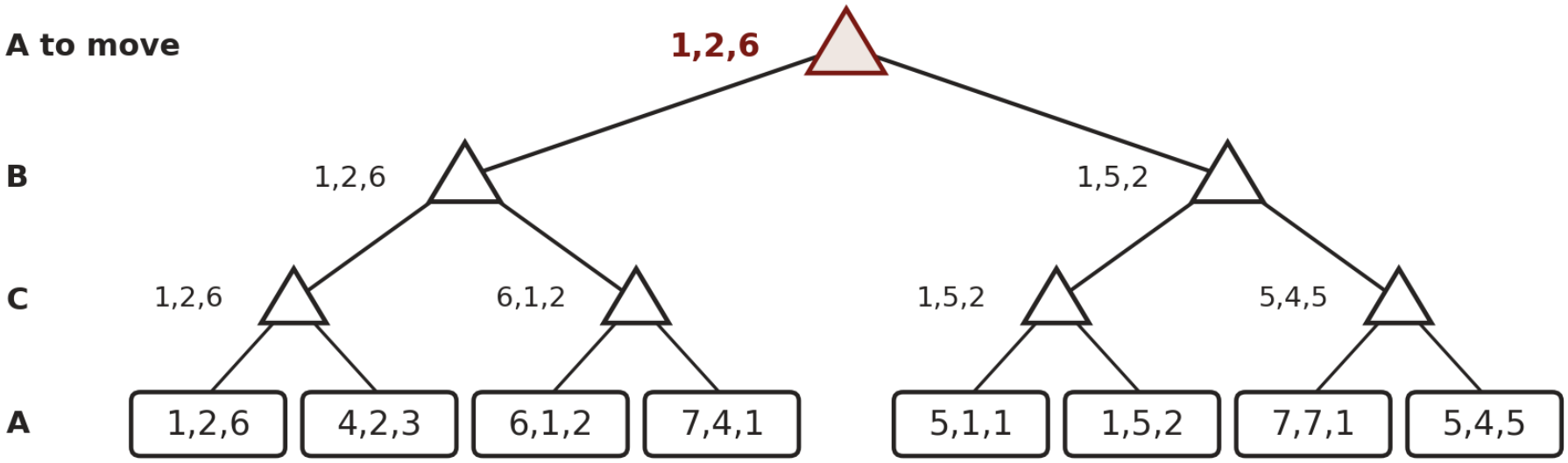
O(bm), or O(m) generating one action at a time

Chess

b near 35 and m near 100

Optimal Decisions in Multiplayer Games

- Replace the single value at each node with a vector holding one utility per player
- Each player backs up the vector whose own component is largest, so there is no minimising any more
- Two player zero sum collapses back to one number because the two components are always opposite



each node carries one utility per player

Alliances

- If A and B are weak and C is strong, attacking C is optimal for both of them separately
- Collaboration emerges from purely selfish play, and it dissolves the moment C stops being the threat
- When the game is not zero sum, even two players can be driven to cooperate

A and B are weak

each is safer while C is the one under attack

C is strong

left alone it destroys both of them in turn

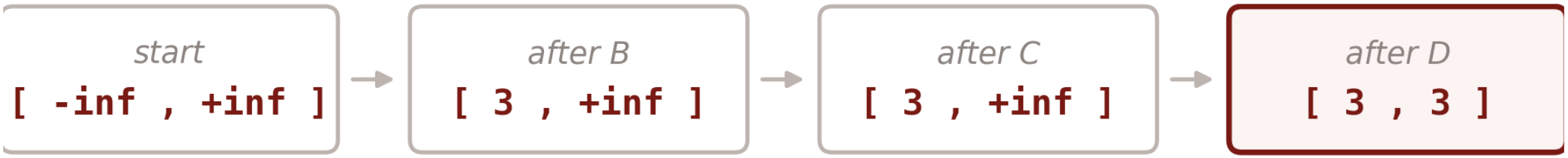
The alliance ends

the moment C is no longer the biggest threat

cooperation here is a consequence of selfishness, not of goodwill

Pruning the Game Tree

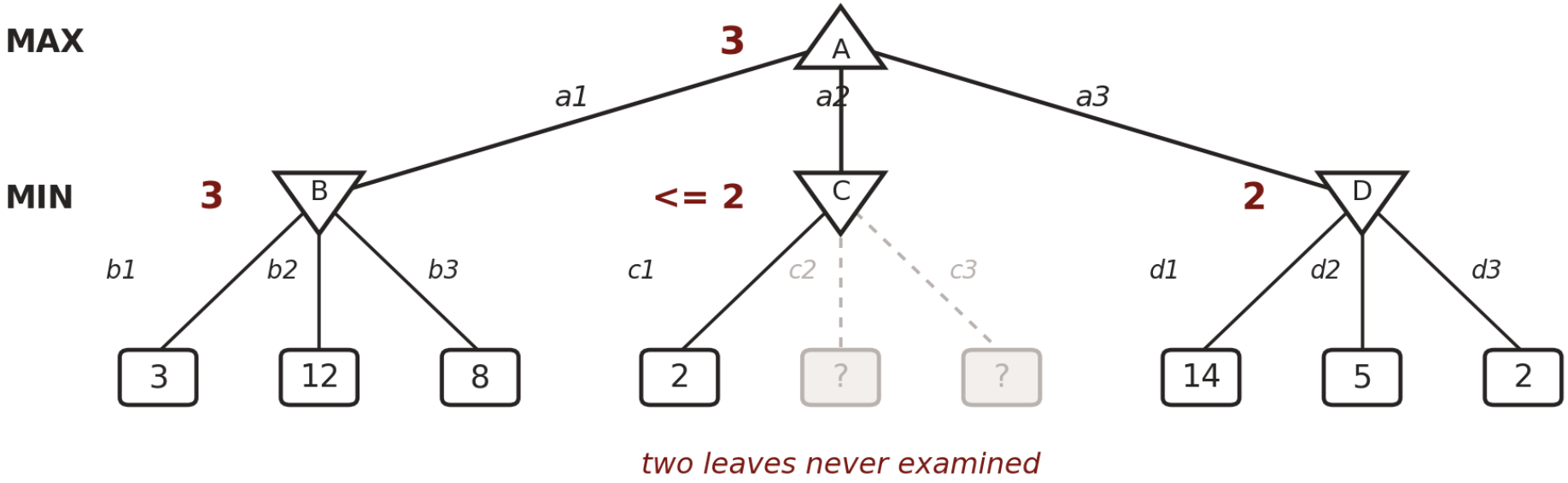
- The number of states minimax examines is exponential in the depth, and that exponent will not go away
- But we can compute the correct minimax decision without looking at every node
- Alpha beta pruning returns exactly the move minimax returns, while never generating branches that cannot change it



what we know about the root value as the search proceeds

Alpha Beta on the Two Ply Tree

- After finishing B we know the root is worth at least 3
- The first leaf under C is 2, so C is a MIN node worth at most 2
- MAX would never choose C, so the other two children of C are never generated



Why the Pruned Leaves Do Not Matter

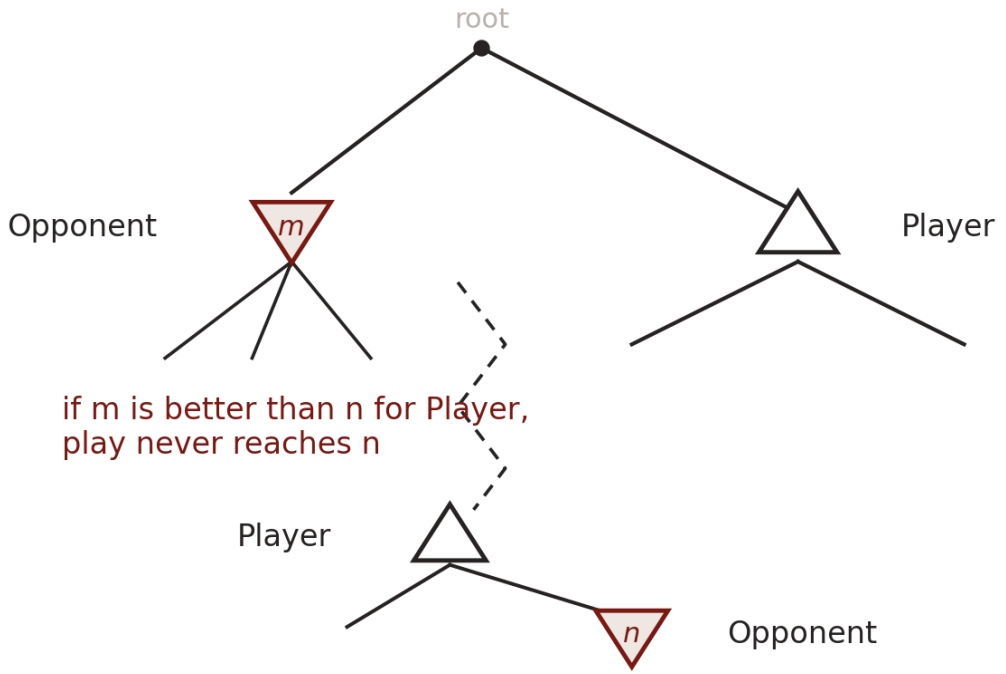
- Call the two unexamined children of C by the names x and y and finish the calculation anyway
- The root value comes out as 3 regardless of what x and y turn out to be

```
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 )     where z = min(2,x,y) <= 2
              = 3
```

the root value does not depend on x and y

The General Principle

- Consider any node n somewhere in the tree that Player could move to
- If Player has a better choice m at the parent of n, or at any choice point further up, then play never reaches n
- Once enough of n has been examined to know this, the rest of n can be discarded



Alpha and Beta

- The search is depth first, so at any moment only one path from the root is live
- Alpha is the best value found so far for MAX anywhere along that path, and beta is the best found so far for MIN
- Prune as soon as the value of the current node is known to be worse than alpha for MAX or beta for MIN

alpha

best value found so far for MAX on the current path

beta

best value found so far for MIN on the current path

search is depth first, so only one path is live at any moment

The Alpha Beta Algorithm

- These are the minimax routines with two lines added to each
- One line returns early once the current value can no longer matter to the caller, and the other keeps the bound up to date

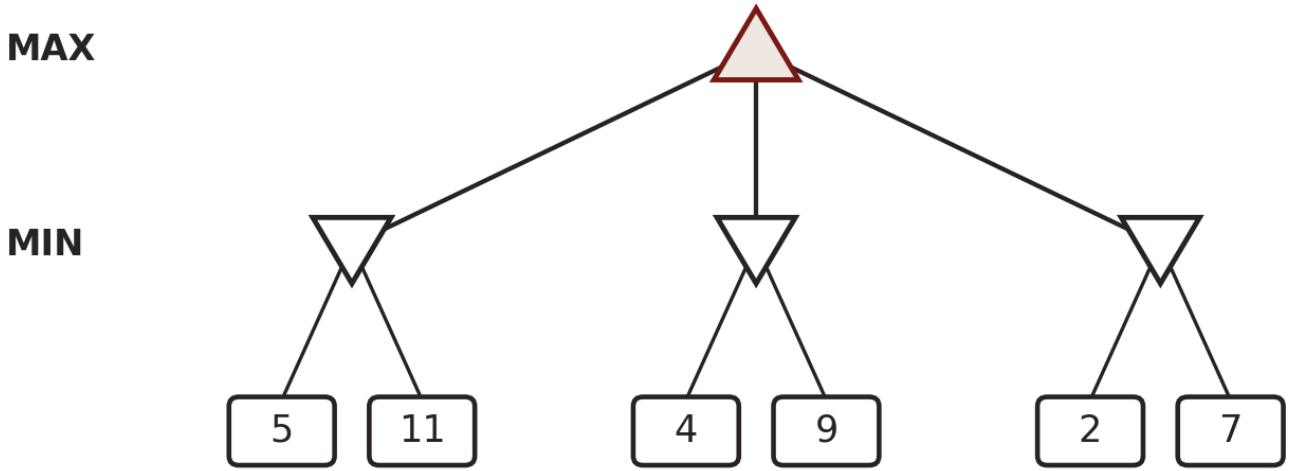
```
MAX-VALUE(state, alpha, beta)
if TERMINAL-TEST(state) return UTILITY(state)
v <- minus infinity
for each a in ACTIONS(state)
    v <- MAX(v, MIN-VALUE(RESULT(state,a), alpha, beta))
    if v >= beta then return v
    alpha <- MAX(alpha, v)
return v
```

```
MIN-VALUE(state, alpha, beta)
if TERMINAL-TEST(state) return UTILITY(state)
v <- plus infinity
for each a in ACTIONS(state)
    v <- MIN(v, MAX-VALUE(RESULT(state,a), alpha, beta))
    if v <= alpha then return v
    beta <- MIN(beta, v)
return v
```

minimax plus the four marked lines, and nothing else

Question 2

Generating children left to right, how many leaves does alpha beta never examine



alpha beta examines leaves left to right

A 0	**B** 1	**C** 2	**D** 3