CS 540 Introduction to Artificial Intelligence
Reinforcement Learning and Search Summary

Josiah Hanna
University of Wisconsin-Madison

December 9, 2021

Slides created by Fred Sala; edited by Josiah Hanna

Announcements

- **Thank you!**
- **Homeworks**:
  - HW10 due Tuesday
- **Office Hours**: Today, 12:30-1:30pm
- Final Exam Rescheduling
- Course Evaluation Survey
- Class roadmap:

Thursday, December 9	RL + Search Summary
Tuesday, December 14	AI in the Real World

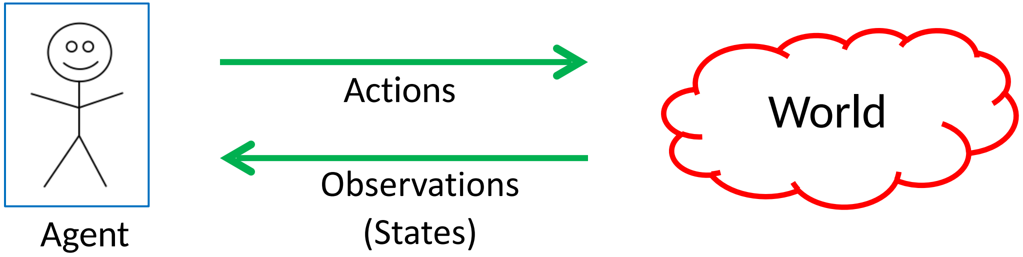
Outline

- Review of reinforcement learning
  - MDPs, value functions, value iteration, Q-learning
- Search Review
  - Uninformed/informed search, optimization
- Games Review
  - Equilibrium, minimax search

Building the Theoretical Model

Basic setup:

- Set of states, S
- Set of actions A



- Information: at time *t*, observe state $s_t \in S$. Get reward r_t
- Agent makes choice $a_t \in A$. State changes to s_{t+1}, continue

Goal: find a map from **states to actions** that maximize rewards.

A "policy"

Markov Decision Process (MDP)

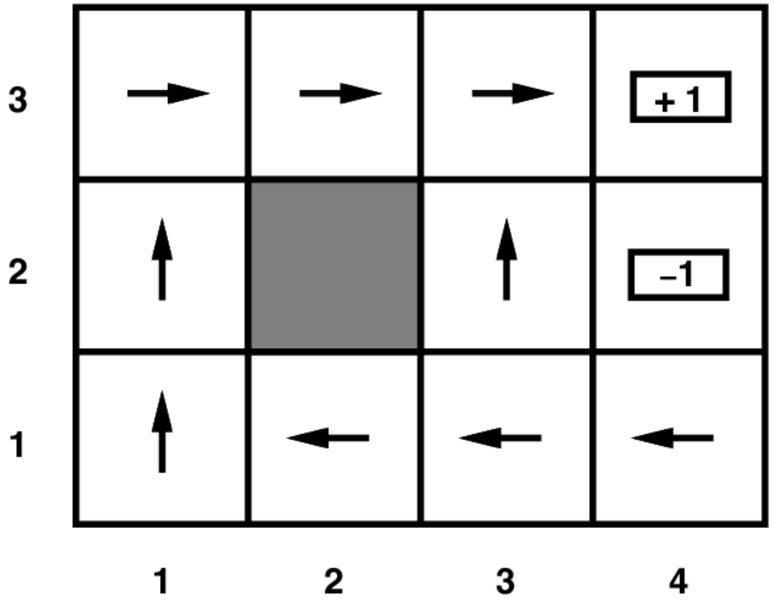
The formal mathematical model:

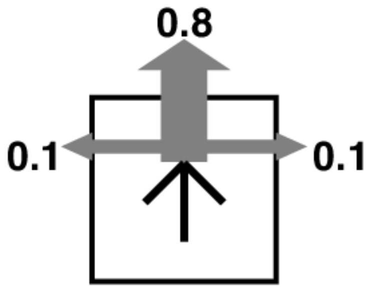
- **State set** S. Initial state $s_{0.}$ **Action set** A
- **State transition model:** $P(s_{t+1}|s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
- **Reward function:** $r(s_t)$
- **Policy**: $\pi(s) : S \rightarrow A$ action to take at a particular state.

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

Grid World Optimal Policy

Note: (i) Robot is unreliable (ii) Reach target fast





$r(s) = -0.04$ for every non-terminal state

Defining the Optimal Policy

For policy π, **expected utility** over all possible state sequences from s_0 produced by following that policy:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences} \\ \text{starting from } s_0}} P(\text{sequence})\,U(\text{sequence})$$

Called the **value function** (for π, s_0)

Discounting Rewards

One issue: these are infinite series. **Convergence**?

- One Solution

$$U(s_0, s_1 \ldots) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \ldots = \sum_{t \geq 0} \gamma^t r(s_t)$$

- Discount factor γ between 0 and 1
  - Set according to how important **present** is vs **future**
  - Note: has to be less than 1 for convergence

Values and Policies

Now that $V^{\pi}(s_0)$ is defined what *a* should we take?

- First, set V*(s) to be expected utility for **optimal** policy from s
- What's the expected utility of an action?
  - Specifically, action a in state s?

$$\sum_{s'} P(s'|s,a)V^*(s')$$

All the states we could go to

Transition probability

Expected rewards under optimal policy

Obtaining the Optimal Policy

We know the expected utility of an action.

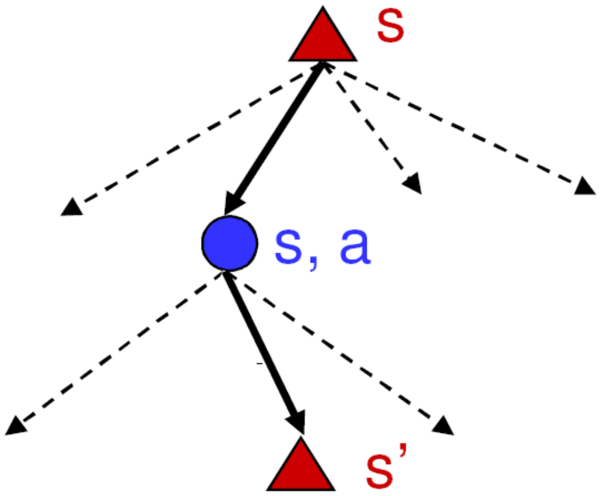
- So, to get the optimal policy, compute

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)V^*(s')$$

All the states we could go to

Transition probability

Expected rewards under optimal policy



Credit L. Lazbenik

Bellman Equation

Let's walk over one step for the value function:

$$V^*(s) = \underbrace{r(s)}_{\text{Current state reward}} + \underbrace{\gamma \max_a \sum_{s'} P(s'|s,a)V^*(s')}_{\text{Discounted expected future rewards}}$$

Current state reward

Discounted expected future **rewards**

- Bellman: inventor of dynamic programming

The Value Iteration Algorithm

Q: how do we find $V^*(s)$?

- Why do we want it? Can use it to get the best policy
- Know: reward $r(s)$, transition probability $P(s'|s,a)$
- Also know $V^*(s)$ satisfies Bellman equation (recursion above)

A: Use the property. Start with $V_0(s)=0$. Then, update

$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s')$$

Break & Quiz

Q 1.1 Consider an MDP with 2 states {A, B} and 2 actions: **"stay"** at current state and "**move**" to other state. Let **r** be the reward function such that **r**(A) = 1, **r**(B) = 0. Let γ be the discounting factor. What is the optimal policy $\pi(A)$ and $\pi(B)$? What are $V^*(A)$, $V^*(B)$?

- A. Stay, Stay, 1/(1-γ), 1
- B. Stay, Move, 1/(1-γ), 1/(1-γ)
- C. Move, Move, 1/(1-γ), 1
- D. Stay, Move, 1/(1-γ), γ/(1-γ)

Break & Quiz

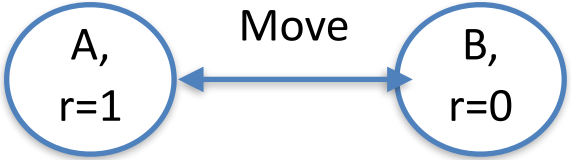
Q 1.1 Consider an MDP with 2 states {*A, B*} and 2 actions: **"stay"** at current state and "**move**" to other state. Let **r** be the reward function such that **r**(*A*) = 1, **r**(*B*) = 0. Let γ be the discounting factor. What is the optimal policy $\pi(A)$ and $\pi(B)$? What are $V^*(A)$, $V^*(B)$?

- A. Stay, Stay, $1/(1-\gamma)$, 1
- B. Stay, Move, $1/(1-\gamma)$, $1/(1-\gamma)$
- C. Move, Move, $1/(1-\gamma)$, 1
- **D. Stay, Move, $1/(1-\gamma)$, $\gamma/(1-\gamma)$**

Break & Quiz

Q 1.1 Consider an MDP with 2 states {*A*, *B*} and 2 actions: **"stay"** at current state and "**move**" to other state. Let **r** be the reward function such that **r**(*A*) = 1, **r**(*B*) = 0. Let γ be the discounting factor. What is the optimal policy $\pi(A)$ and $\pi(B)$? What are $V^*(A)$, $V^*(B)$?

- A. Stay, Stay, 1/(1-γ), 1
- B. Stay, Move, 1/(1-γ), 1/(1-γ)
- C. Move, Move, 1/(1-γ), 1
- **D. Stay, Move, 1/(1-γ), γ/(1-γ)** Note: want to stay at A, if at B, move to A. Starting at A, sequence A,A,A,… rewards 1, γ, γ^2,…. Start at B, sequence B,A,A,… rewards 0, γ, γ^2,…. Sums to 1/(1-γ), γ/(1-γ).



Q-Learning

What if we don't know transition probability $P(s'|s,a)$?

- Need a way to learn to act without it.
- **Q-learning**: get an action-utility function $Q(s,a)$ that tells us the value of doing a in state s
- Note: $V^*(s) = \max_a Q(s,a)$
- Now, we can just do $\pi^*(s) = \arg\max_a Q(s,a)$
  - But need to estimate Q!

Q-Learning Iteration

How do we get Q(s,a)?

- Similar iterative procedure
- In state s, take action a, observe r(s), and next state:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Learning rate

Idea: combine old value and new estimate of future value.

Note: Policy derived from Q; take action with maximal action-value.

Exploration Vs. Exploitation

General question!

- **Exploration:** take an action with unknown consequences
  - Pros:
    - Get a more accurate Q function
    - Discover higher-reward states than the ones found so far
  - Cons:
    - When exploring, not maximizing your utility
    - Something bad might happen
- **Exploitation:** go with the best strategy found so far
  - Pros:
    - Maximize reward as reflected in the current utility estimates
    - Avoid bad stuff
  - Cons:
    - Might also prevent you from discovering the true optimal strategy

Q-Learning: Epsilon-Greedy Policy

How to **explore**?

- With some 0<ε<1 probability, take a random action at each state, or else the action with highest Q(*s*,*a*) value.

$$a = \begin{cases} \text{argmax}_{a \in A} Q(s, a) & \text{uniform}(0,1) > \epsilon \\ \text{random } a \in A & \text{otherwise} \end{cases}$$

SARSA

An alternative:

- Just use the next action, no max over actions:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Learning rate

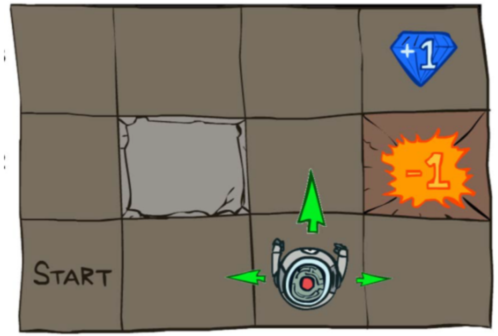
Action actually taken at next step

- Called state–action–reward–state–action (**SARSA**)
- Can use with epsilon-greedy policy
- Slightly different convergence than Q-learning unless epsilon reduced over time.

Q-Learning Details

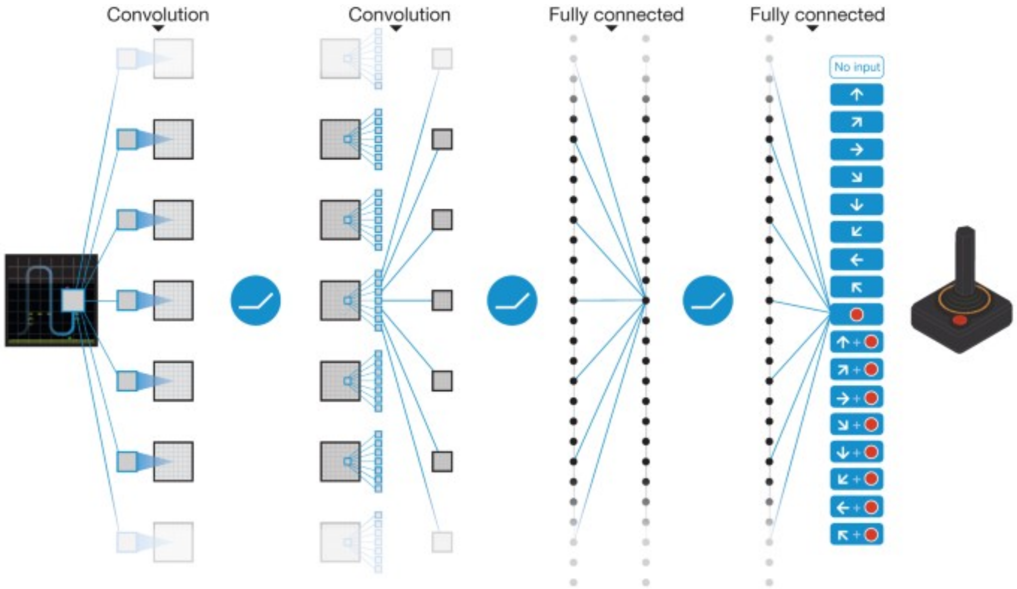
Note: if we have a **terminal** state, the process ends

- An **episode**: a sequence of states ending at a terminal state
- Want to run on many episodes
- Slightly different Q-update for terminal states (see homework!)



Deep Q-Learning

How do we get Q(*s*,*a*) with a large number of states?



Mnih et al, "Human-level control through deep reinforcement learning"

Deep Q-Learning

How do we get Q(s,a) with a large number of states?

- Function approximation!
- Deep Q-learning uses a neural network to approximate Q(s,a)
- Similar to regression using (s, a) as input and $y = r(s) + \gamma \max_{a'} Q(s', a')$ as output.
- Loss function: $\mathscr{L}(\theta) = (y - Q_\theta(s, a))^2$

DQN Pseudocode

1. Initialize replay memory, D, and action-value neural network, Q_θ.
2. $Q_{\texttt{target}} \leftarrow Q_\theta$
3. For episode =1,M do:
    1. Initialize s_t = s_0
    2. For t=1,T do:
        1. Select a_t with epsilon greedy action selection
        2. Take action a_t and observe s' and reward.
        3. Add (s_t, a_t, s', r) to replay memory D
        4. Sample minibatch of (s,a,s',r) tuples from D.
        5. For each tuple in minibatch, set $y = r(s) + \gamma \max_{a'} Q_{\texttt{target}}(s', a')$
        6. Perform gradient descent on $\mathscr{L}(\theta) = (y - Q_\theta(s,a))^2$
        7. Every k steps update target Q network: $Q_{\texttt{target}} \leftarrow Q_\theta$

Summary of RL

- Reinforcement learning setup
- Mathematical formulation: MDP
- Value functions & the Bellman equation
- Value iteration
- Q-learning

Search and RL Review

- Search
  - Uninformed vs Informed
  - Optimization
- Games
  - Minimax search
- Reinforcement Learning
  - MDPs, value iteration, Q-learning, SARSA

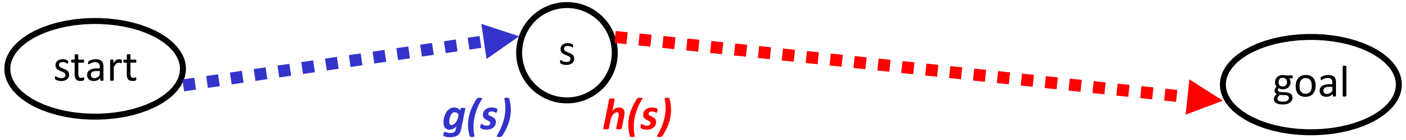
Uninformed vs Informed Search

Uninformed search (all of what we saw). Know:

- Path cost ***g***(*s*) from start to node *s*
- Successors.

Informed search. Know:

- All uninformed search properties, plus
- Heuristic **h(s)** from s to goal

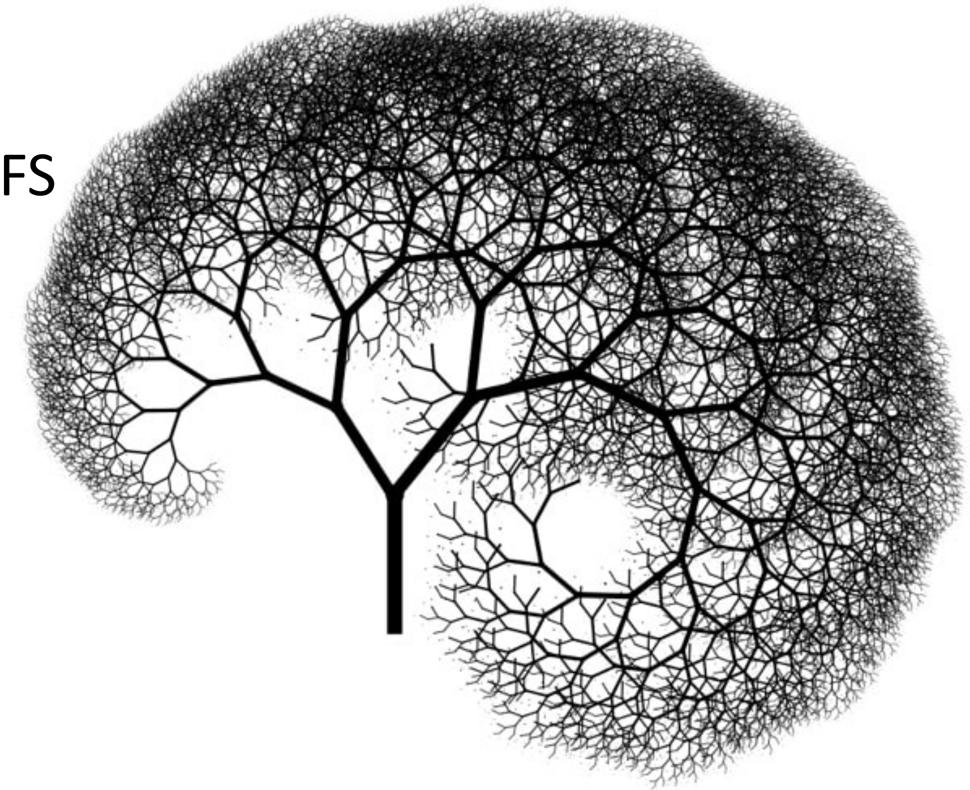


Uninformed Search: Iterative Deepening DFS

Repeated limited DFS

- Search like BFS, fringe like DFS
- **Properties**:
  - Complete
  - Optimal (if edge cost 1)
  - Time O(b^d)
  - Space O(bd)

A good option!

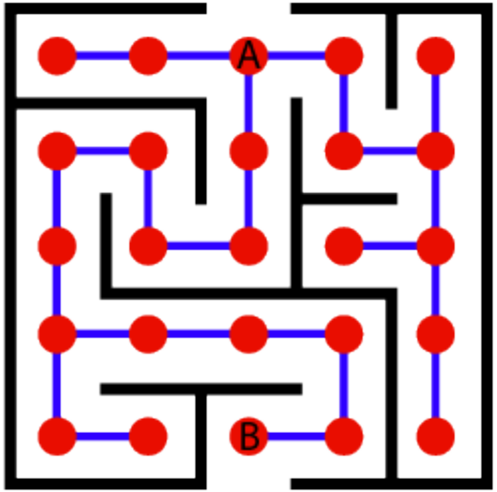


Fractalsaco

Informed Search: **A* Search**

A*: Expand best $g(s) + h(s)$, with one requirement

- Demand that $h(s) \leq h^*(s)$
- If heuristic has this property, "admissible"
  - Optimistic! Never over-estimates
- Still need $h(s) \geq 0$
  - Negative heuristics can lead to strange behavior

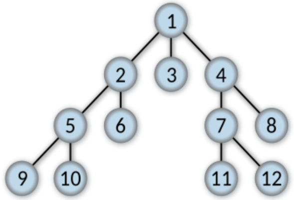


V. Batoćanin

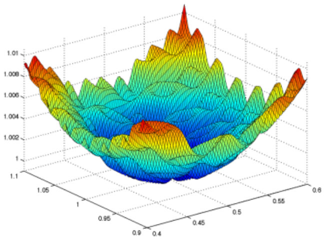
Search vs. Optimization

Before: wanted a path from start state to goal state

- Uninformed search, informed search



Wiki



TuringFin

New setting: optimization

- States *s* have values ***f***(*s*)
- Want: *s* with optimal value ***f***(*s*) (i.e, **optimize** over states)
- Challenging setting: **too many states** for previous search approaches, but maybe not a continuous function for SGD.

Hill Climbing Algorithm

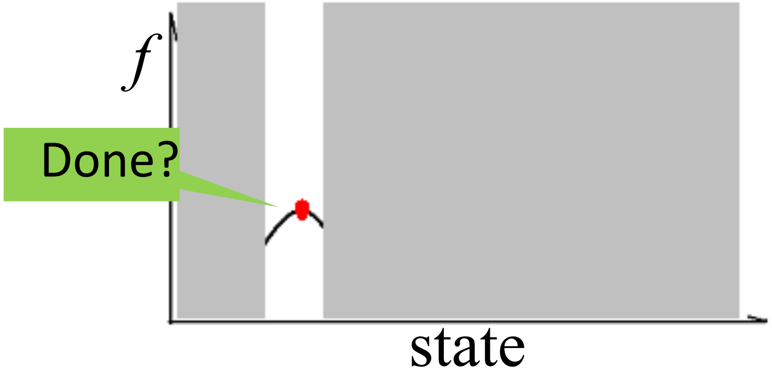
Pseudocode:

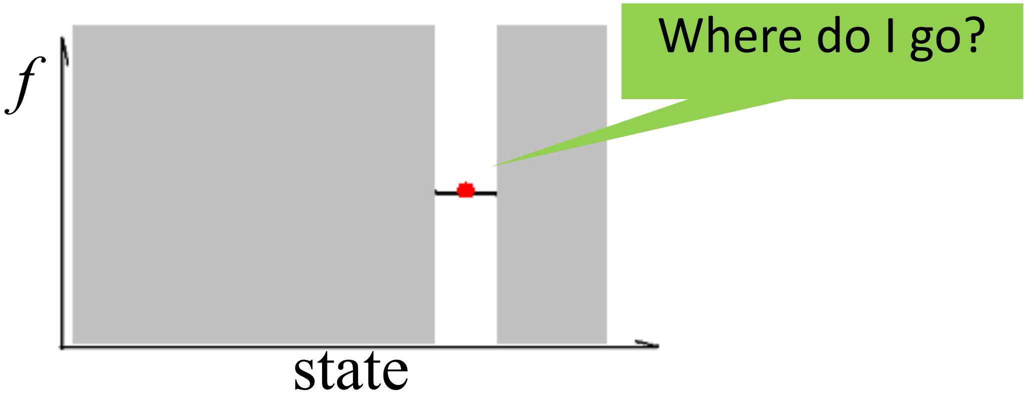
1. Pick initial state s
2. Pick t in **neighbors**(s) with the largest $\boldsymbol{f}(t)$
3. if $\boldsymbol{f}(t) \leq \boldsymbol{f}(s)$ THEN stop, return s
4. $s \leftarrow t$. goto 2.

What could happen? **Local optima!**

Hill Climbing: Local Optima

Note the **local optima**. How do we handle them?

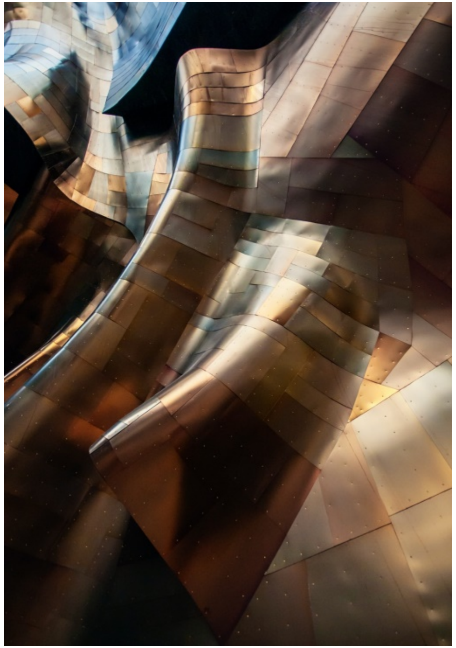




Simulated Annealing

A more sophisticated optimization approach.

- **Idea**: move quickly at first, then slow down
- Pseudocode:



Pick initial state s

For k = 0 through k_{max}:

  $\boldsymbol{T} \leftarrow$ temperature($(k+1)/k_{max}$)

  Pick a random neighbor, $t \leftarrow$ neighbor(s)

The interesting bit

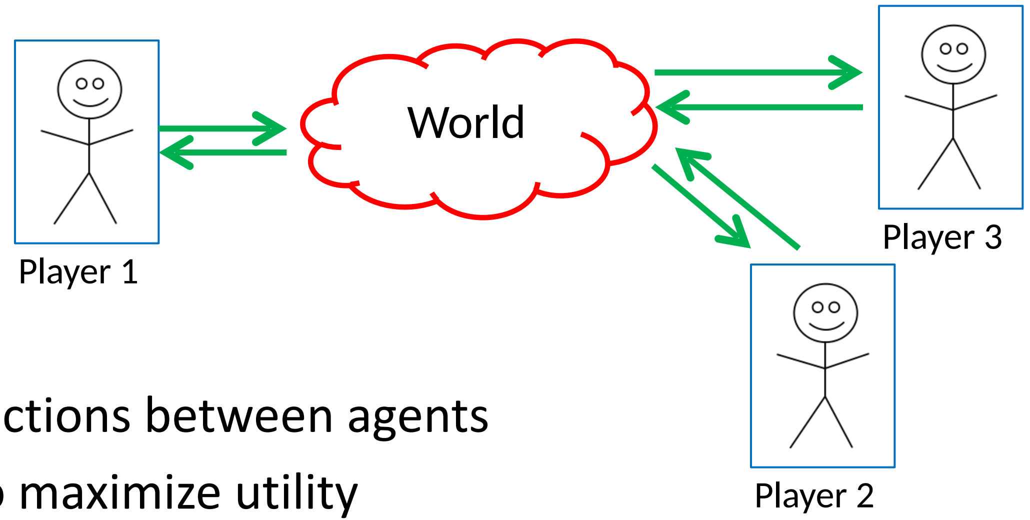
  If $f(s) \leq f(t)$, then $s \leftarrow t$

  Else, with prob. $P(f(s), f(t), \boldsymbol{T})$ then $s \leftarrow t$

Output: the final state s

Games Setup

Games setup: **multiple** agents



- Now: interactions between agents
- Still want to maximize utility
- **Strategic** decision making.

Equilibrium

*a** is an equilibrium if all the players do not have an incentive to **unilaterally deviate**

$$u_i(a_i^*, a_{-i}^*) \geq u_i(a_i, a_{-i}^*) \quad \forall a_i \in A_i$$

- All players dominant strategies -> equilibrium
- Converse doesn't hold (don't need dominant strategies to get an equilibrium)

Pure and Mixed Strategies

So far, all our strategies are deterministic: "**pure**"

- Take a particular action, no randomness

Can also randomize actions: "**mixed**"

- Assign probabilities x_i to each action

$$x_i(a_i), \text{ where } \sum_{a_i \in A_i} x_i(a_i) = 1, x_i(a_i) \geq 0$$

- Note: have to now consider **expected rewards**

Nash Equilibrium

Consider the mixed strategy $x^* = (x_1^*, \ldots, x_n^*)$

- This is a **Nash equilibrium** if

$$u_i(x_i^*, x_{-1}^*) \geq u_i(x_i, x_{-i}^*) \quad \forall x_i \in \Delta_{A_i}, \forall i \in \{1, \ldots, n\}$$

Better than doing anything else, "**best response**"

Space of probability distributions

- Intuition: nobody can **increase expected reward** by changing only their own strategy. A type of solution!

Minimax Value

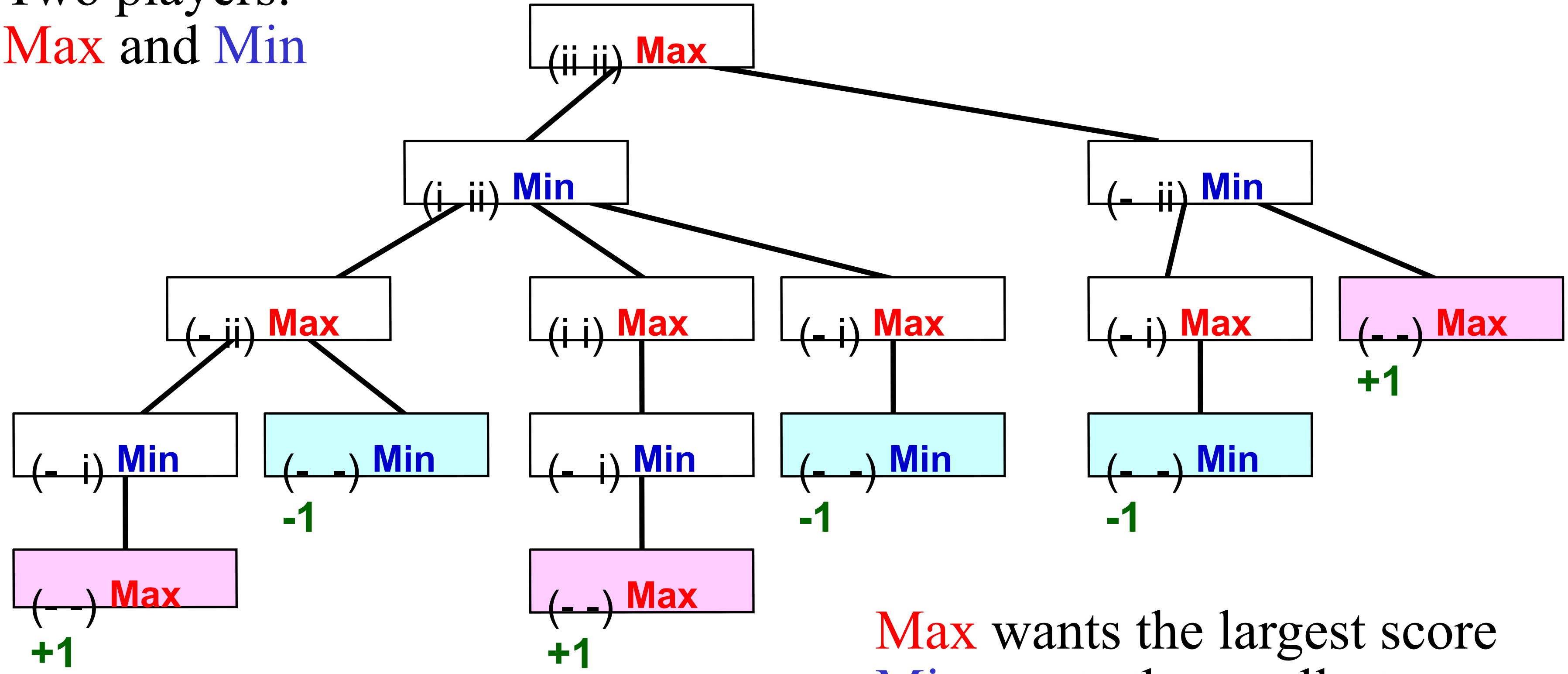
Also called **game-theoretic value**.

- Score of terminal node if both players play optimally.
- Computed bottom up; basically search

- Let’s see this for example game

Game tree for II-Nim

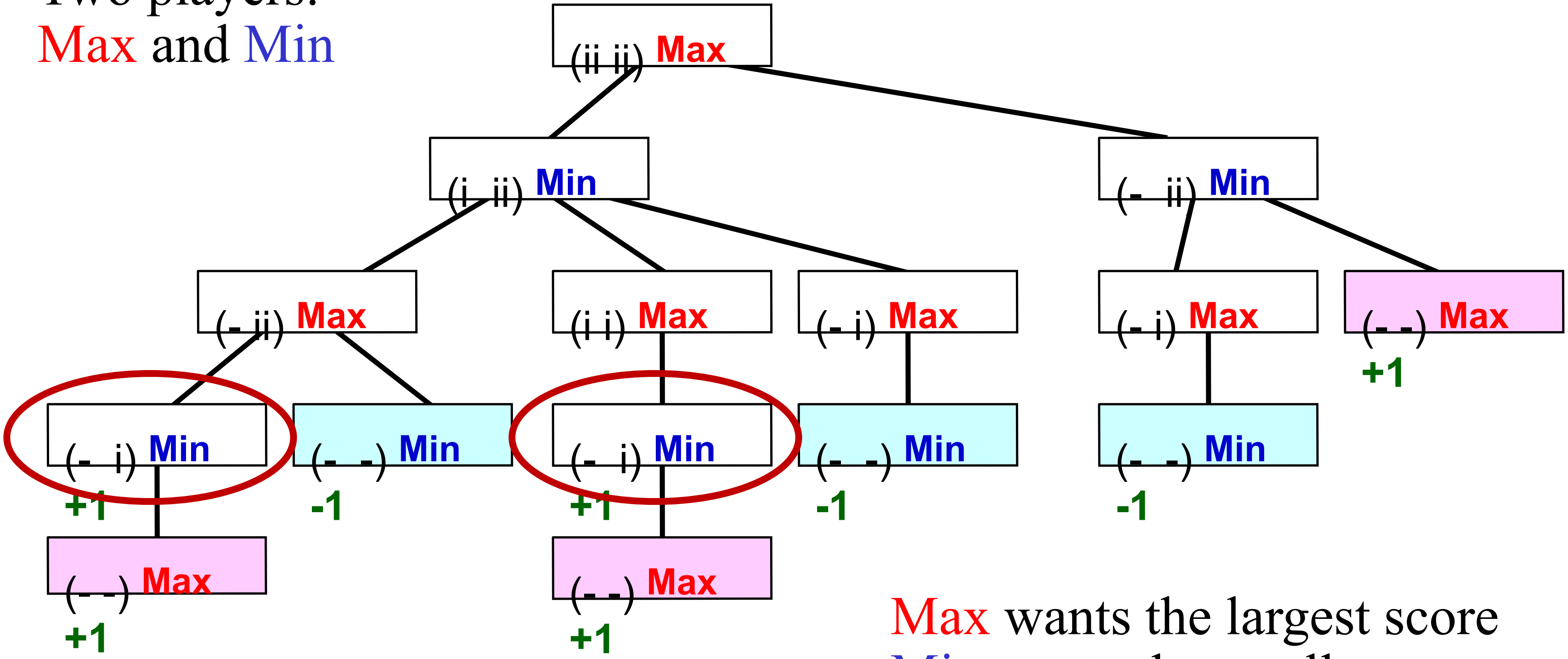
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

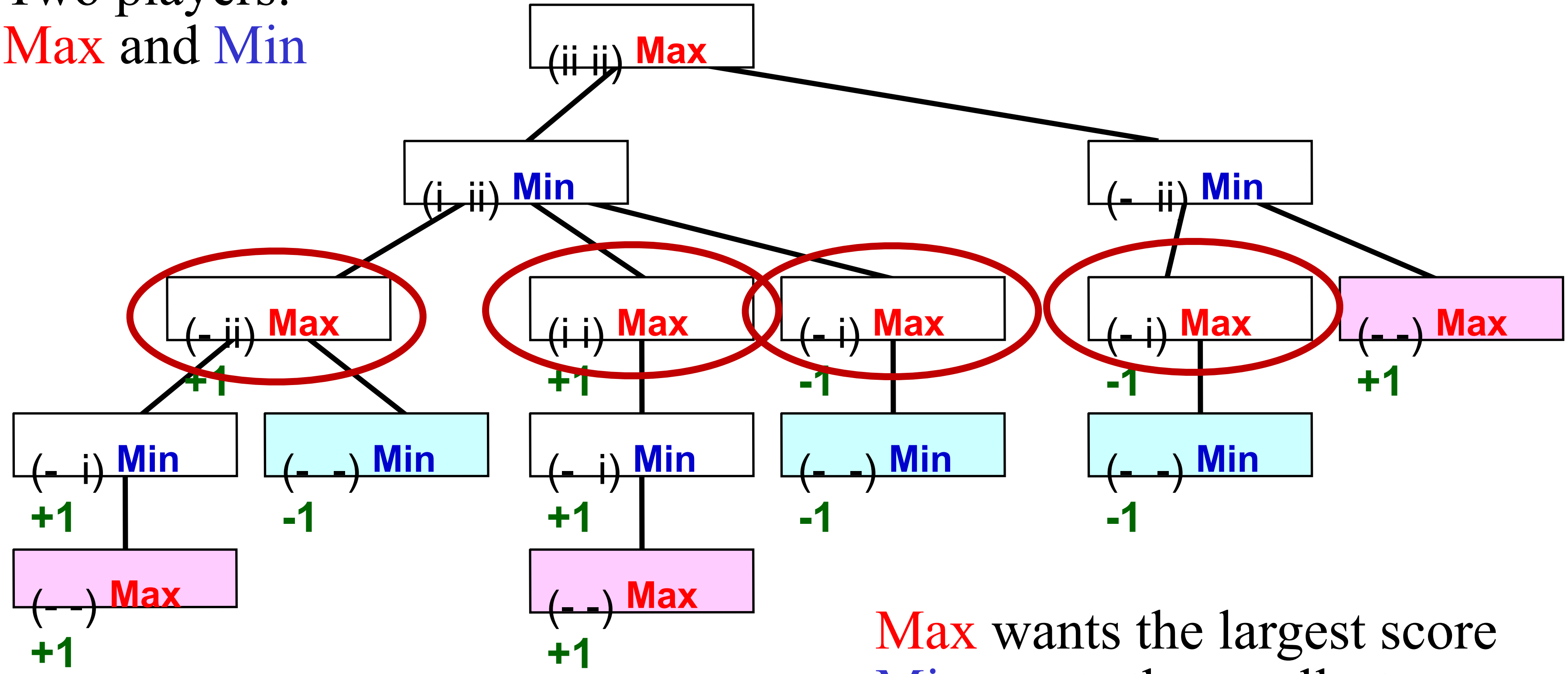
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

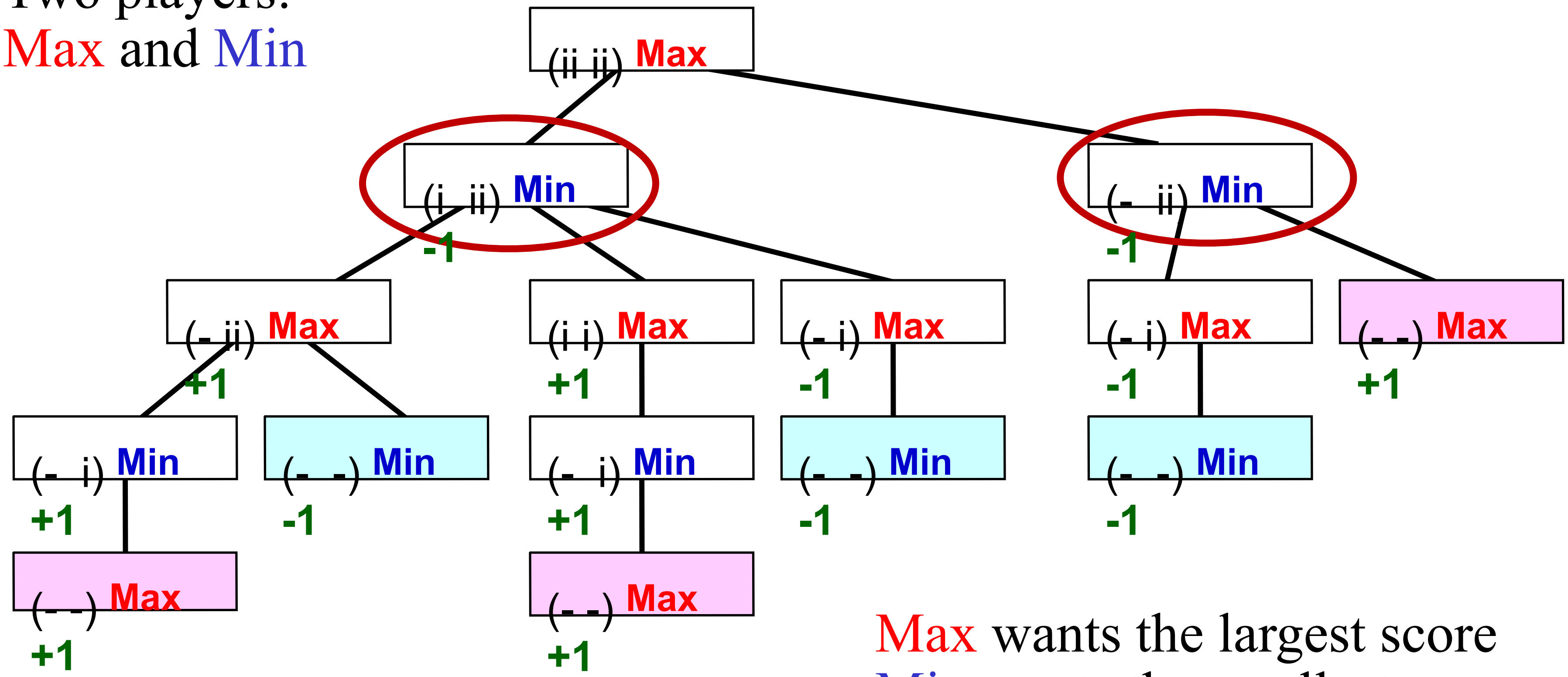
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

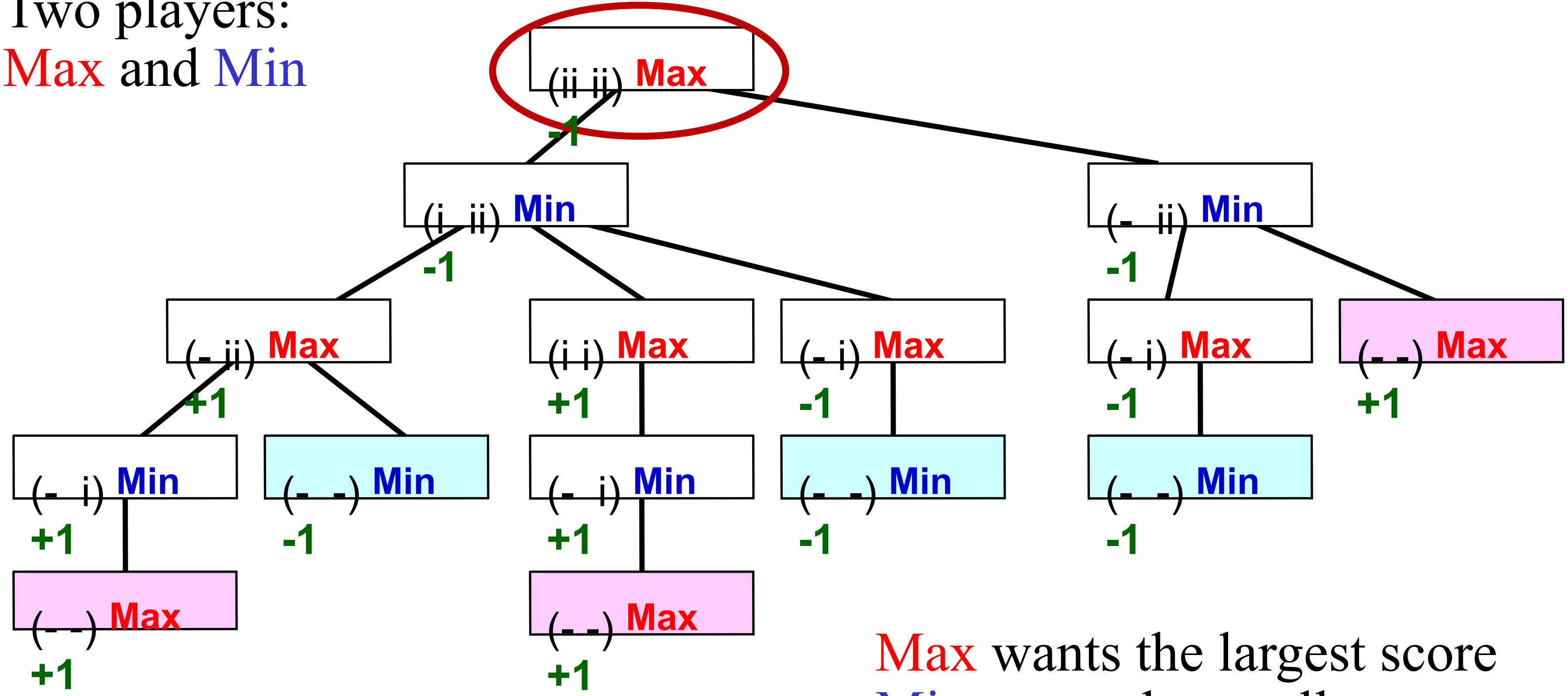
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

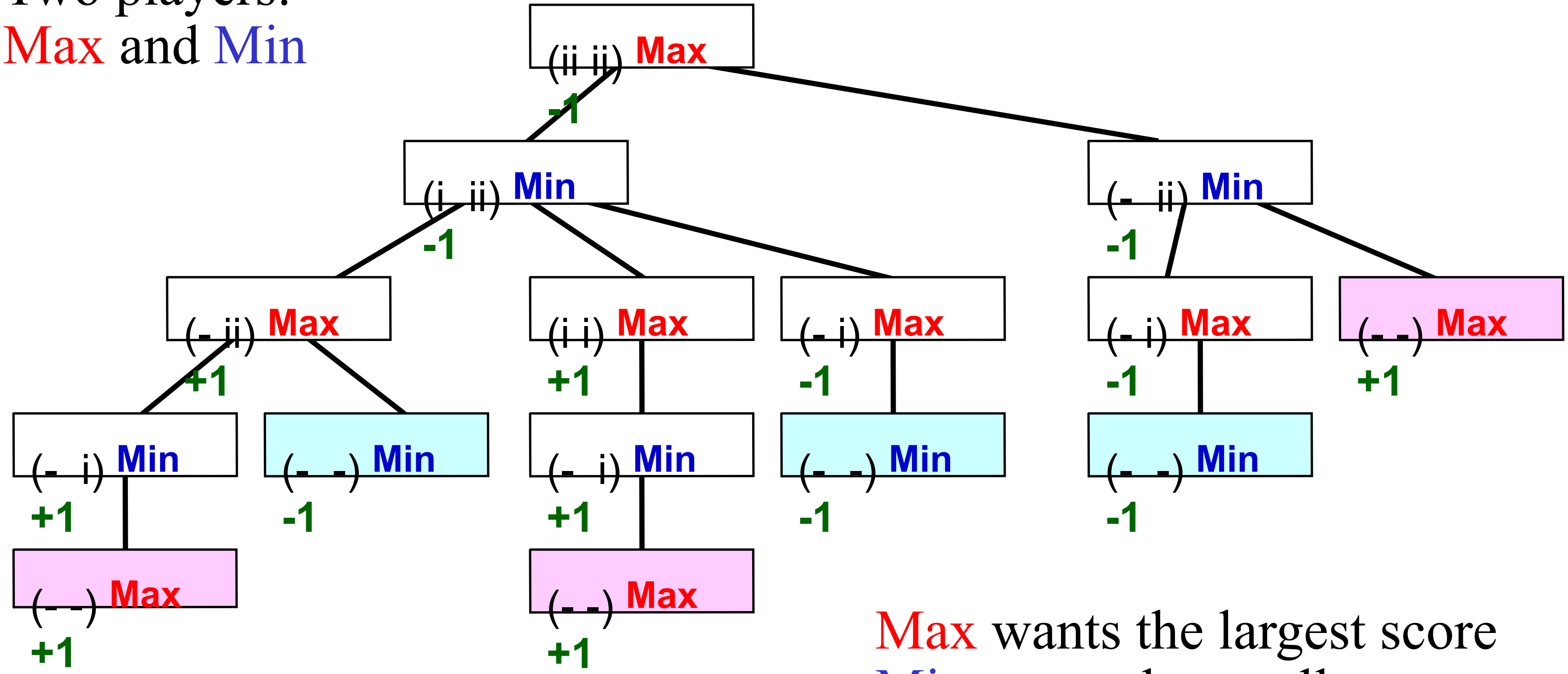
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

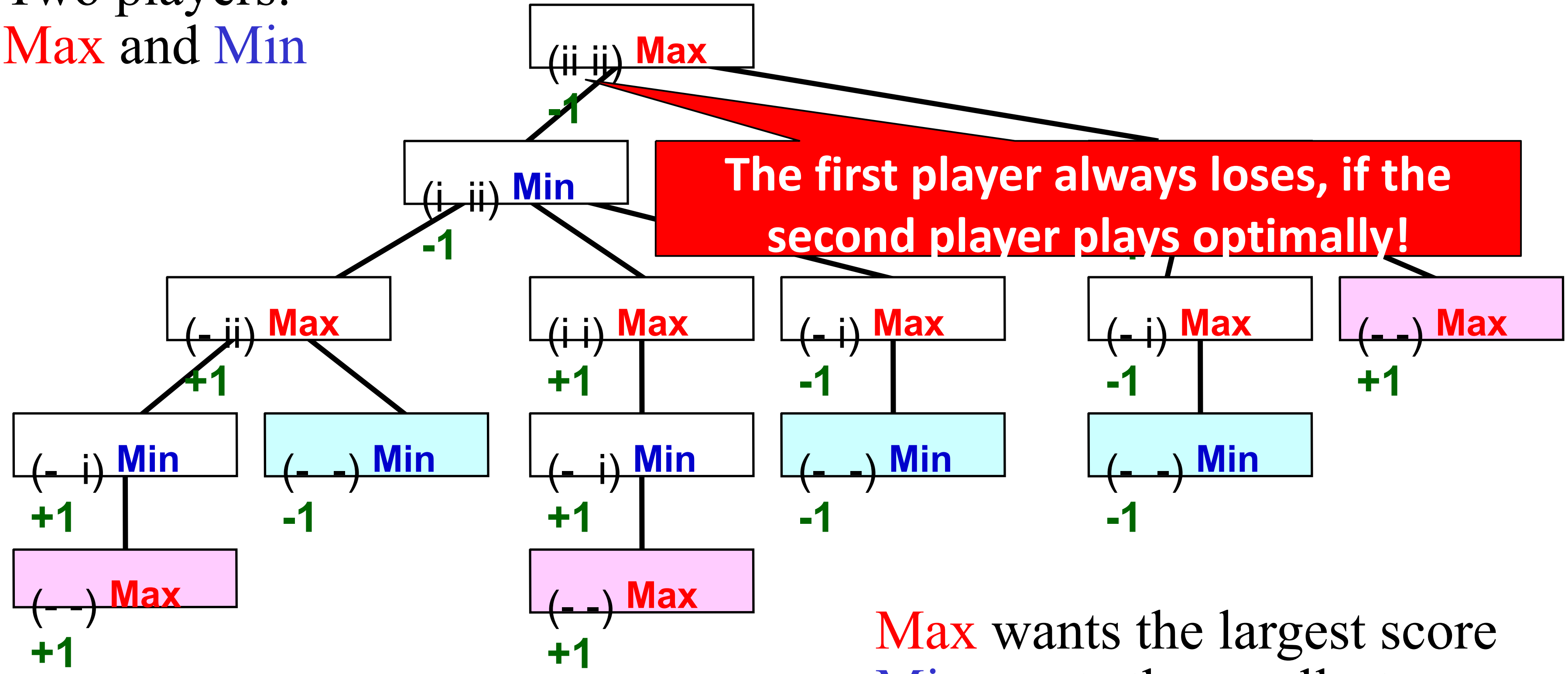
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game tree for II-Nim

Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Minimax Search

Note that long games yield huge computation

- To deal with this: limit ***d*** for the search depth
- **Q**: What to do at depth *d*, but no termination yet?
  - **A**: Use a heuristic evaluation function *e(x)*

function MINIMAX(x, d) **returns** an estimate of x's utility value
  inputs: x, current state in game
      d, an upper bound on the search depth
  if x is a terminal state **then return** Max's payoff at x
  else if $d = 0$ **then return** $e(x)$
  else if it is Max's move at x **then**
    return $\max\{\text{MINIMAX}(y, d-1) : y \text{ is a child of } x\}$
  else return $\min\{\text{MINIMAX}(y, d-1) : y \text{ is a child of } x\}$

Credit: Dana Nau

Acknowledgements: Based on slides from Yin Li, Jerry Zhu, Fred Sala, Svetlana Lazebnik, Yingyu Liang, David Page, Mark Craven, Pieter Abbeel, Dan Klein