CS 540 Introduction to Artificial Intelligence
Review on Search, Games, and RL

Yingyu Liang
University of Wisconsin-Madison

Dec 9, 2021

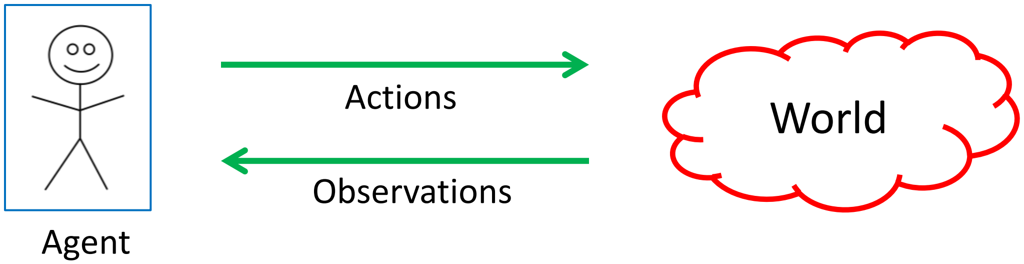
Based on slides by Fred Sala

Announcements (details on Piazza)

- Final Exam information
  - On Canvas/Quizzes as midterm; but no one-day window
  - Main: Dec 20 2:45-4:45pm
  - Makeup: Dec 23 2:45-4:45pm

- Course Evaluation
  - Dec 1 to Dec 15
  - Explicit incentive: some details about the final exam if the participation rate reaches 50%/75%/95%

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** maximize rewards.

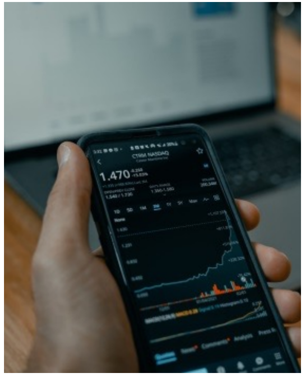
A “policy”

Value function

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)

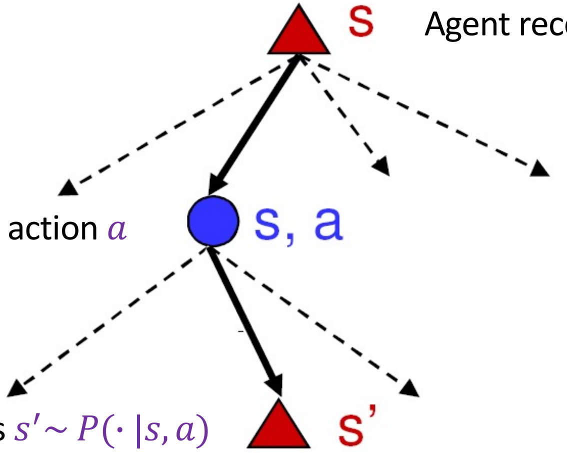


The Bellman equation

Agent receives reward $r(s)$

Agent chooses action a

Environment returns $s' \sim P(\cdot\,|s,a)$

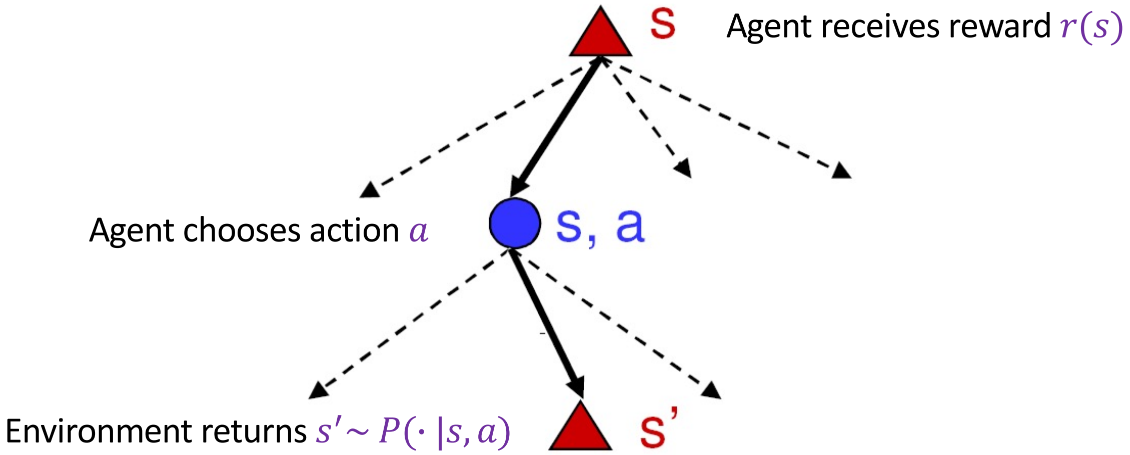


- What is the recursive expression for $V^{\pi}(s)$ in terms of $V^{\pi}(s')$ - the utilities of its successors?

$$V^{\pi}(s) = r(s) + \gamma \sum_{s'} P(s'|s, \pi(s))V^{\pi}(s')$$

Image source: L. Lazbenik

The Bellman equation

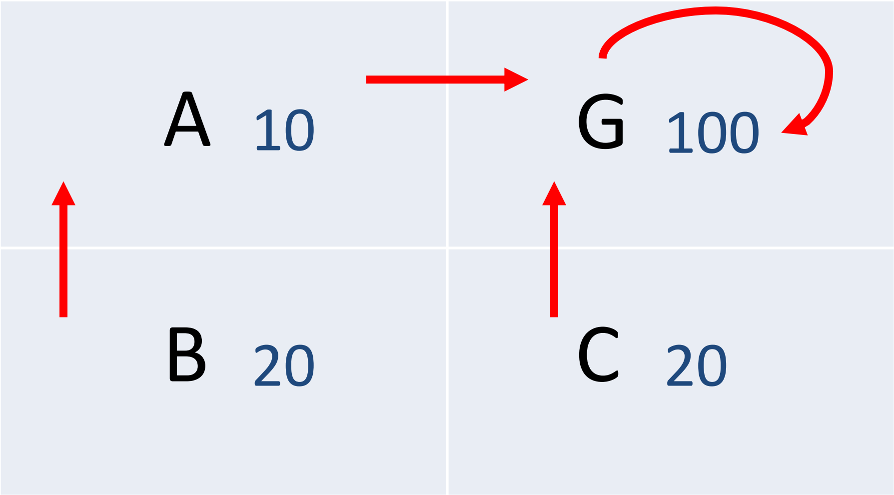


- Applied to the optimal policy:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$$

Image source: L. Lazbenik

Example



Deterministic transition. $\gamma = 0.8$, policy shown in red arrow.

Value Iteration

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

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

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* (including the reward in *s*)

$$Q(s,a) = r(s) + \gamma \sum_{s'} P(s'|s,a)V^*(s')$$

- 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

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

Learning rate

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

Note: We are using a policy to take actions; based on the estimated Q!

Q-Learning: Epsilon-Greedy Policy

How to **explore**?

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

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

Q-Learning: 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

- Called state–action–reward–state–action (**SARSA**)
- Can use with epsilon-greedy policy

Summary of RL

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

Search and Games Review

- Search
  - Uninformed vs Informed
  - Optimization
- Games
  - Game tree, Game-theoretical value, Minimax search
  - Normal form, Equilibrium

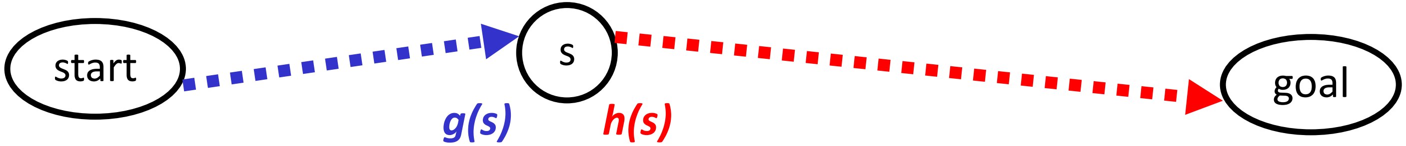
Uninformed vs Informed Search

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 (recall game heuristic)

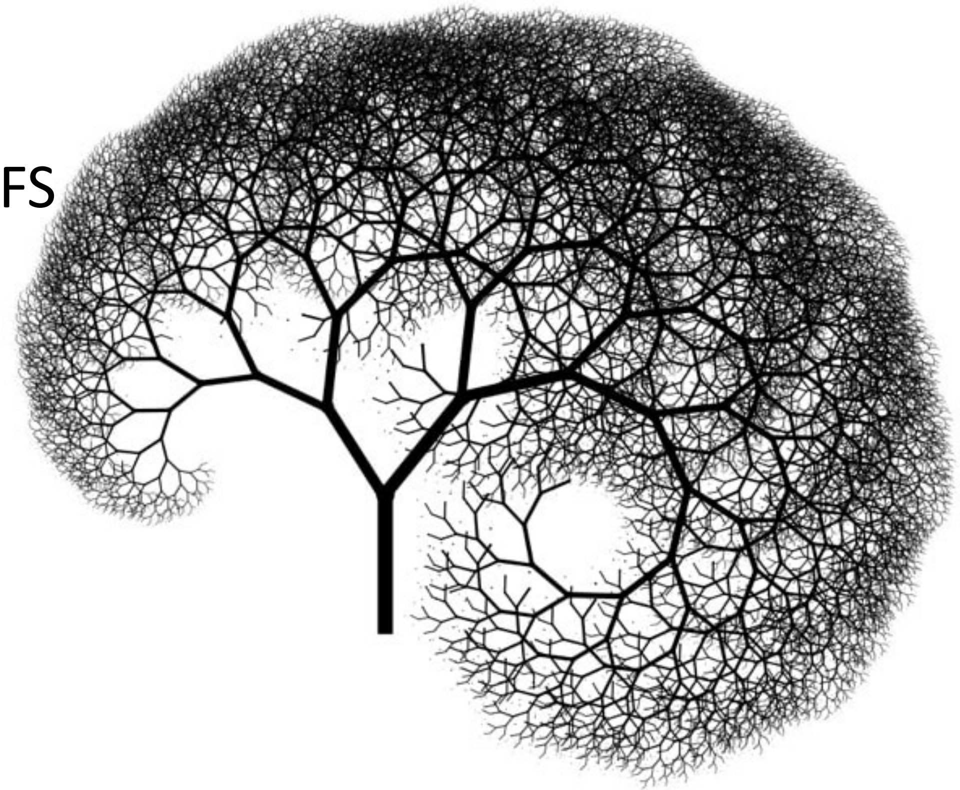


Uninformed Search: Iterative Deepening DFS

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**

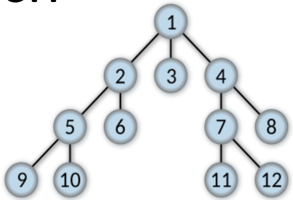
A*: Expand best $g(s) + h(s)$, with one requirement

- 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

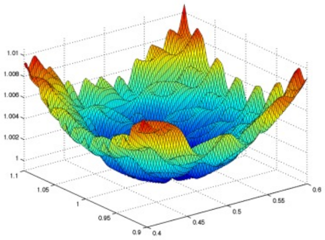
Search vs. Optimization

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

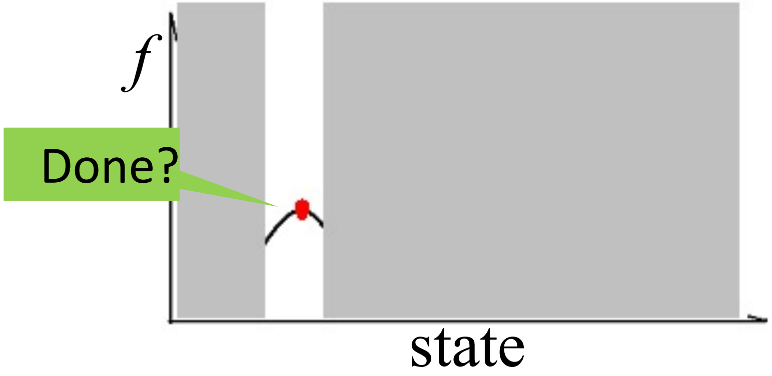
Pseudocode:

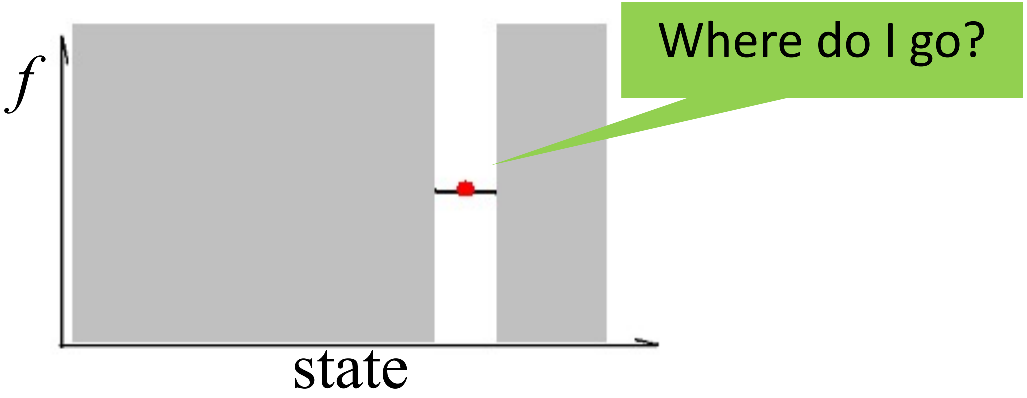
1. Pick initial state *s*
2. Pick *t* in **neighbors**(*s*) with the largest ***f***(*t*)
3. if ***f***(*t*) ≤ ***f***(*s*) THEN stop, return *s*
4. *s* ← *t*. goto 2.

What could happen? **Local optima!**

Hill Climbing: Local Optima

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

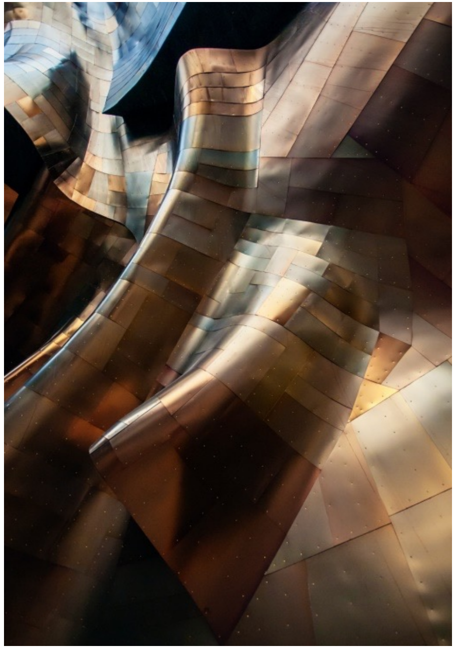




Simulated Annealing

A more sophisticated optimization approach.

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



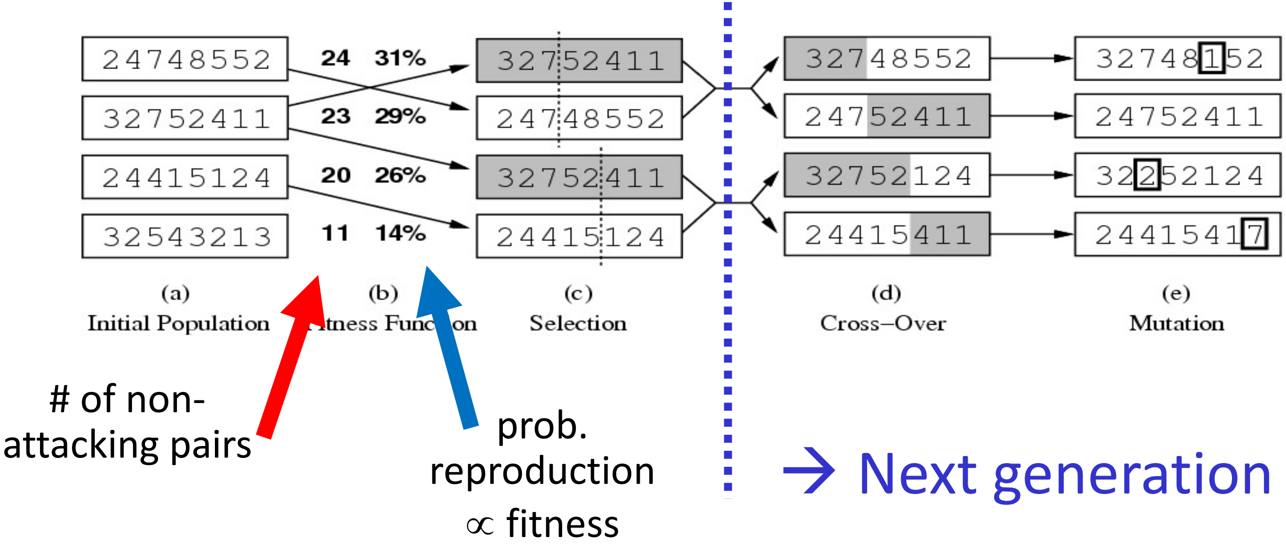
Pick initial state s
For k = 0 through k_{max}:
  $\boldsymbol{T}$ ← temperature($(k+1)/k_{max}$)
  Pick a random neighbor, t ← neighbor(s)
  If $\boldsymbol{f}(s) \leq \boldsymbol{f}(t)$, then s ← t
  Else, with prob. $P(\boldsymbol{f}(s), \boldsymbol{f}(t), \boldsymbol{T})$ then s ← t
Output: the final state s

The interesting bit

Genetic Algorithm

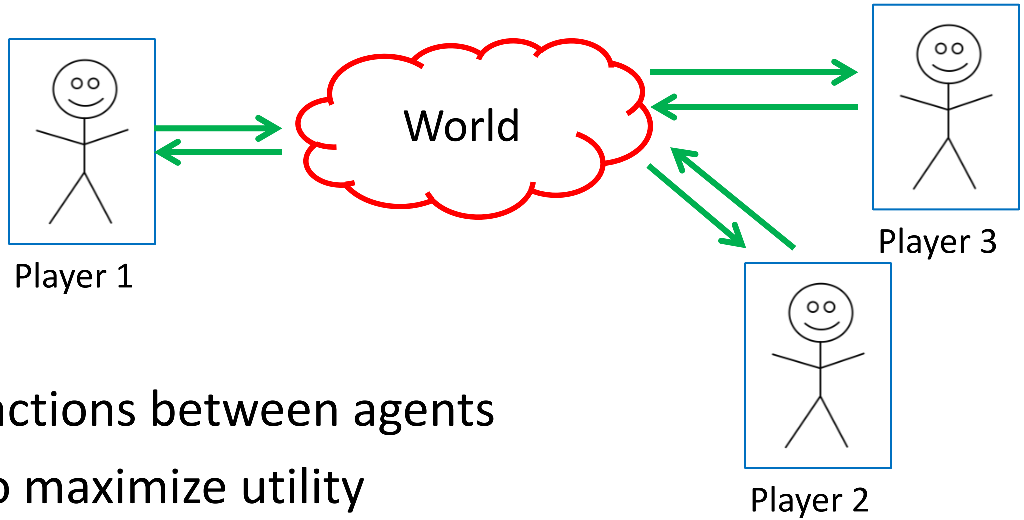
Goal of genetic algorithms: optimize using principles inspired by mechanism for evolution

- E.g., analogous to **natural selection, cross-over**, and **mutation**



Games Setup

Games setup: **multiple** agents

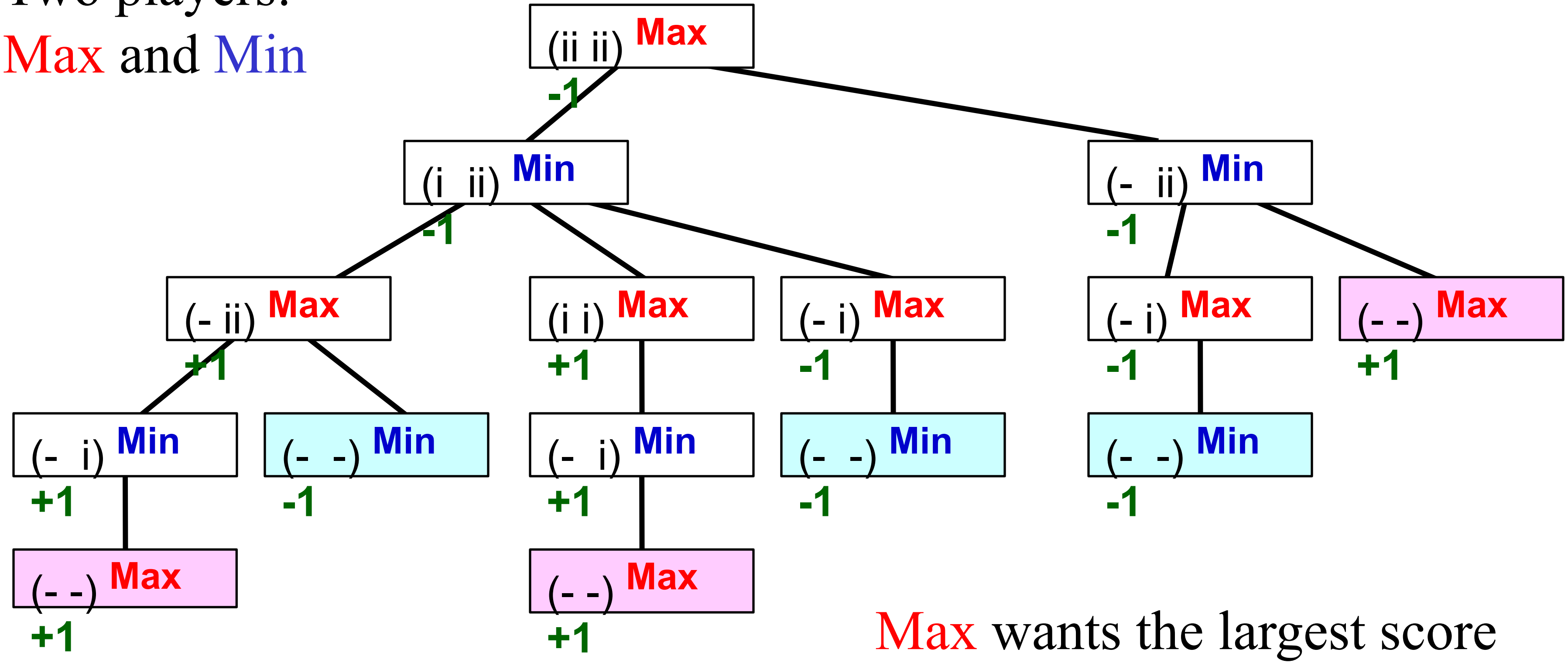


- Now: interactions between agents
- Still want to maximize utility
- **Strategic** decision making.

Game tree for Sequential Game II-Nim

Two players:

Max and Min



Max wants the largest score

Min wants the smallest score

Minimax Algorithm

```
function Max-Value(s)
inputs:
      s: current state in game, Max about to play
output: best-score (for Max) available from s

      if ( s is a terminal state )
      then return ( terminal value of s )
      else
                α := – infinity
                for each s' in Succ(s)
                    α := max( α , Min-value(s'))
      return α
```

```
function Min-Value(s)
output: best-score (for Min) available from s

      if ( s is a terminal state )
      then return ( terminal value of s)
      else
                β := infinity
                for each s' in Succs(s)
                    β := min( β , Max-value(s'))
      return β
```

Time complexity?

- $O(b^m)$

Space complexity?

- $O(bm)$

Simultaneous Games

The players make moves simultaneously

- Can express reward with a simple diagram (Normal form)
- Ex: for prisoner's dilemma

Player 2 Player 1	*Stay silent*	*Betray*
Stay silent	−1, −1	−3, 0
Betray	0, −3	−2, −2

Nash Equilibrium

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

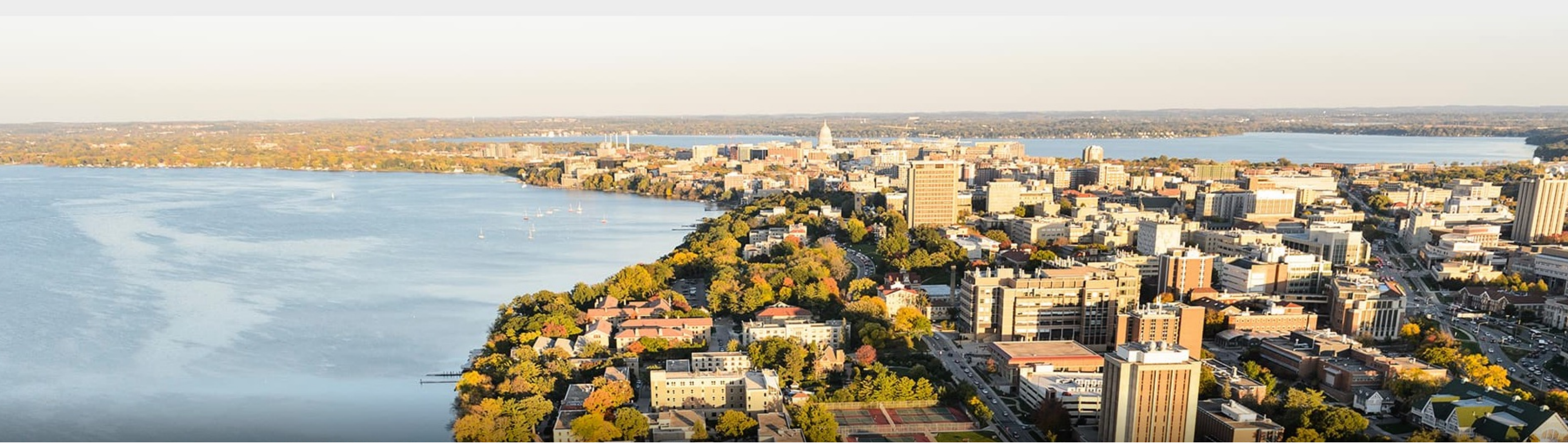
- This is a **Nash equilibrium** if

$$u_i(x_i^*, x_{-i}^*) \geq u_i(x_i, x_{-i}^*) \quad \forall x_i \in \Delta_{A_i}, \forall i \in \{1, 2, \ldots, n\}$$

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!



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