CS 540 Introduction to Artificial Intelligence
Reinforcement Learning II / Summary

University of Wisconsin-Madison

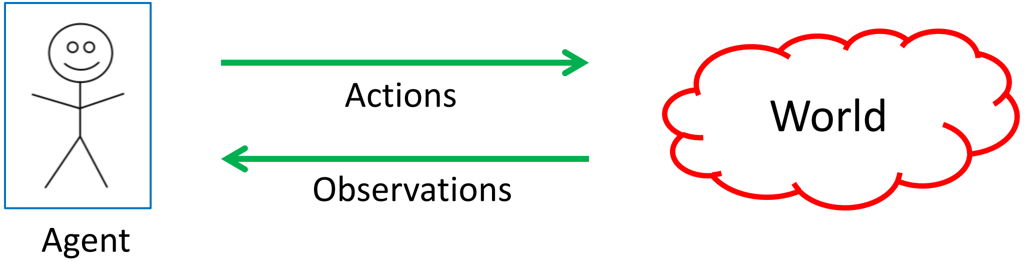
Fall 2022

Outline

- Review of reinforcement learning
  - MDPs, value functions, Bellman Equation, value iteration
- Q-learning
  - Q function, Q-learning

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”

Markov Decision Process (MDP)

The formal mathematical model:

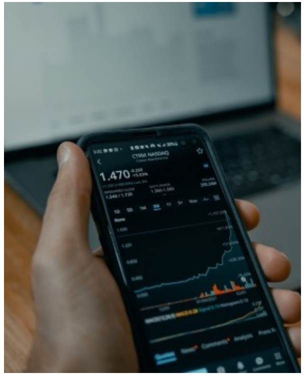
- **State set** S. Initial state s_0. **Action set** A
- **State transition model:** $P(s_{t+1} | s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
- **Reward function:** $r(s_t)$
- **Policy**: $\pi(s) : S \rightarrow A$ action to take at a particular state.

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

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

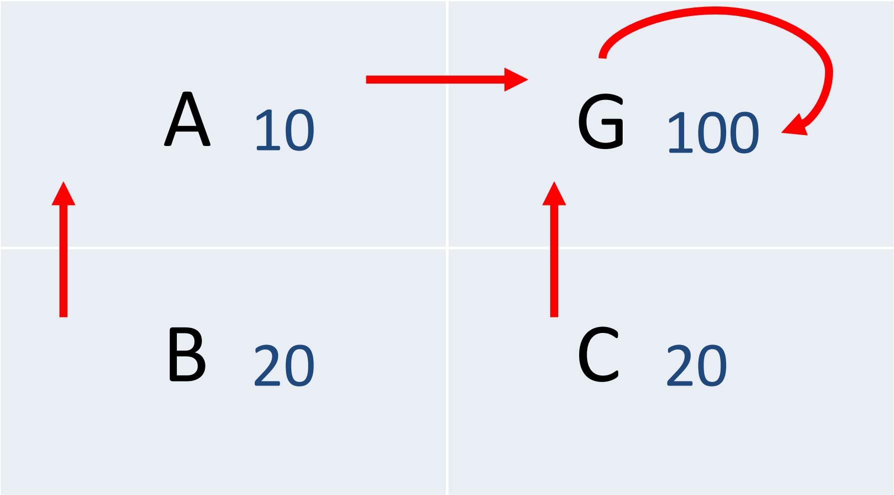
One issue: these are infinite series. **Convergence**?

- Solution

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

- 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

Example



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

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

Obtaining the Optimal Policy

We know the expected utility of an action.

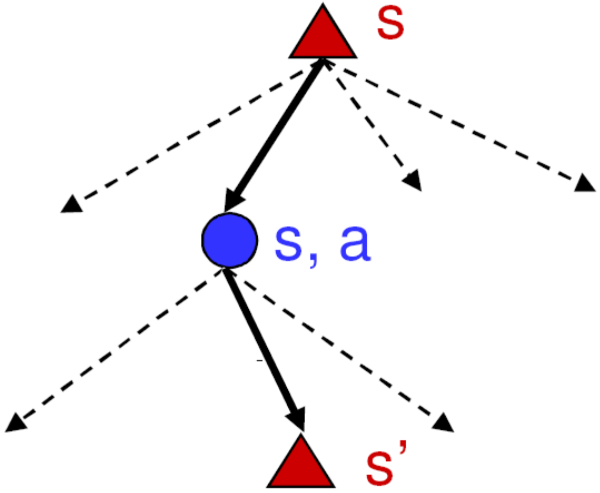
- So, to get the optimal policy, compute

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

All the states we could go to

Transition probability

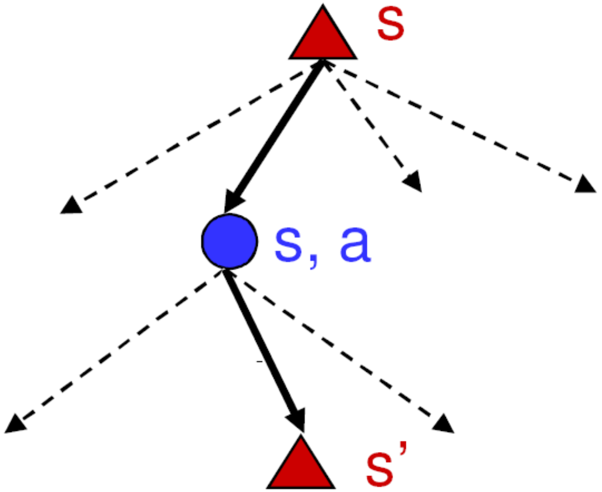
Expected rewards



Credit L. Lazbenik

Bellman Equation

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



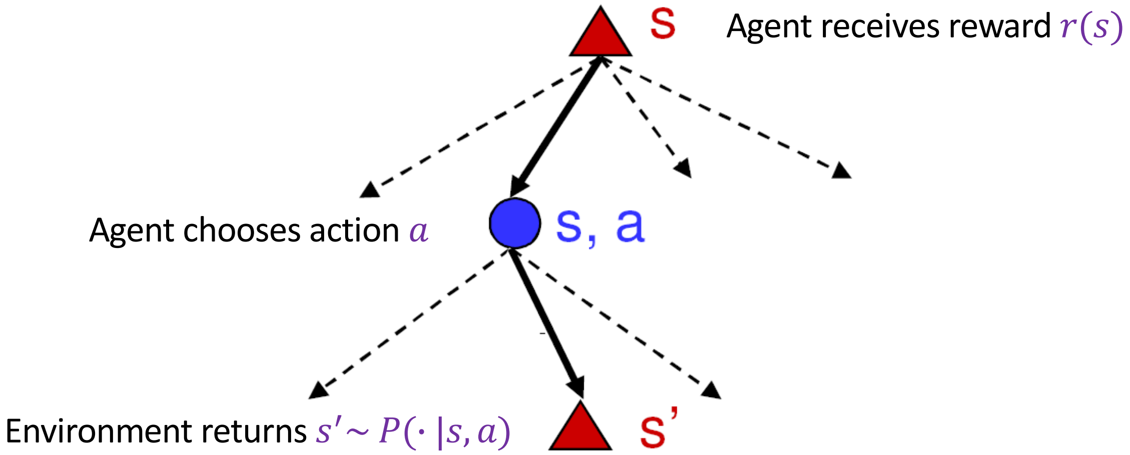
Credit L. Lazbenik

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

Current state reward

Discounted expected future **rewards**

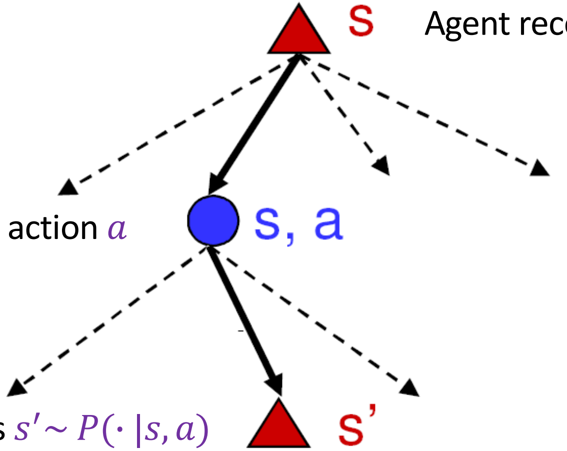
The Bellman equation



- Define state utility $V^*(s)$ as the expected sum of discounted rewards if the agent executes an *optimal* policy starting in state s

Image source: L. Lazbenik

The Bellman equation



Agent receives reward $r(s)$

Agent chooses action a

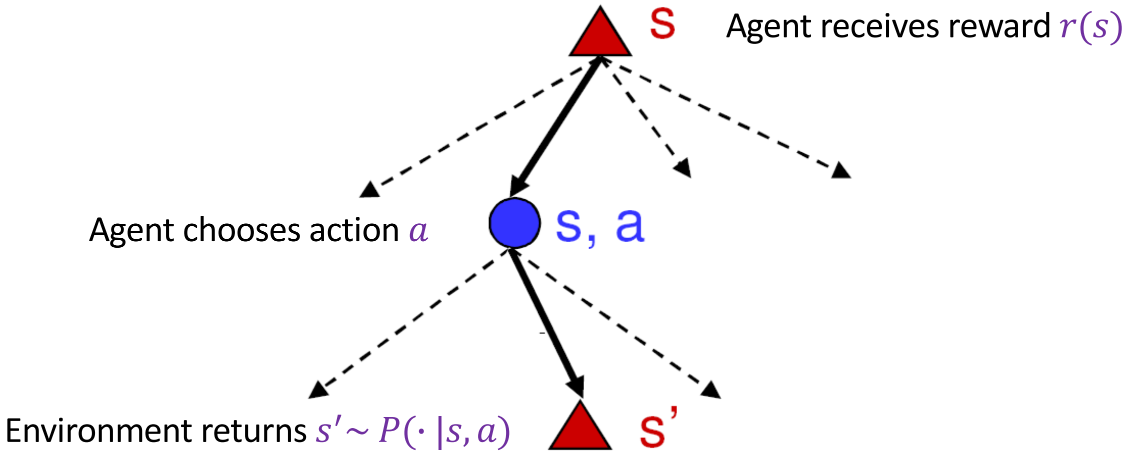
Environment returns $s' \sim P(\cdot\,|s,a)$

- What is the expected utility of taking action **a** in state **s**?

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

Image source: L. Lazbenik

The Bellman equation

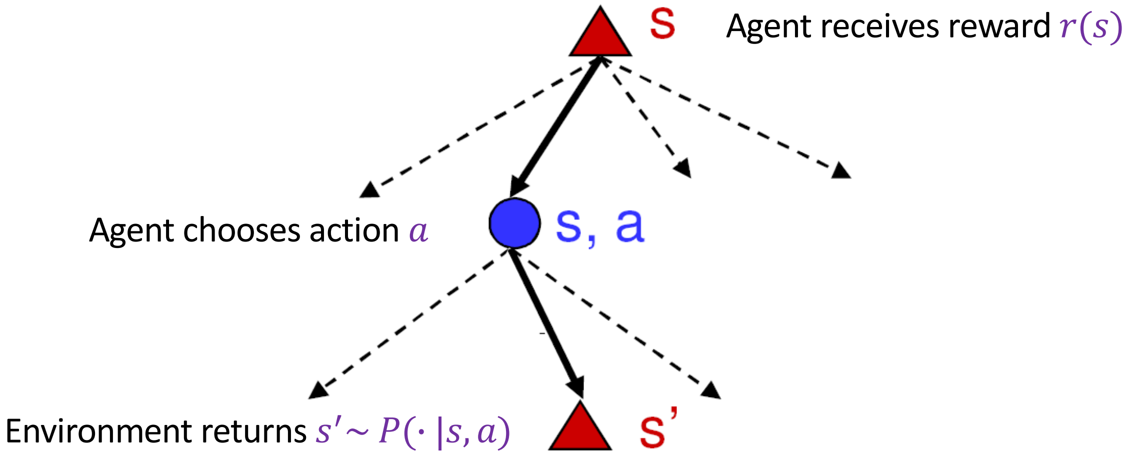


- What is the recursive expression for $V^*(s)$ in terms of $V^*(s')$ - the utilities of its successors?

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

Image source: L. Lazbenik

The Bellman equation

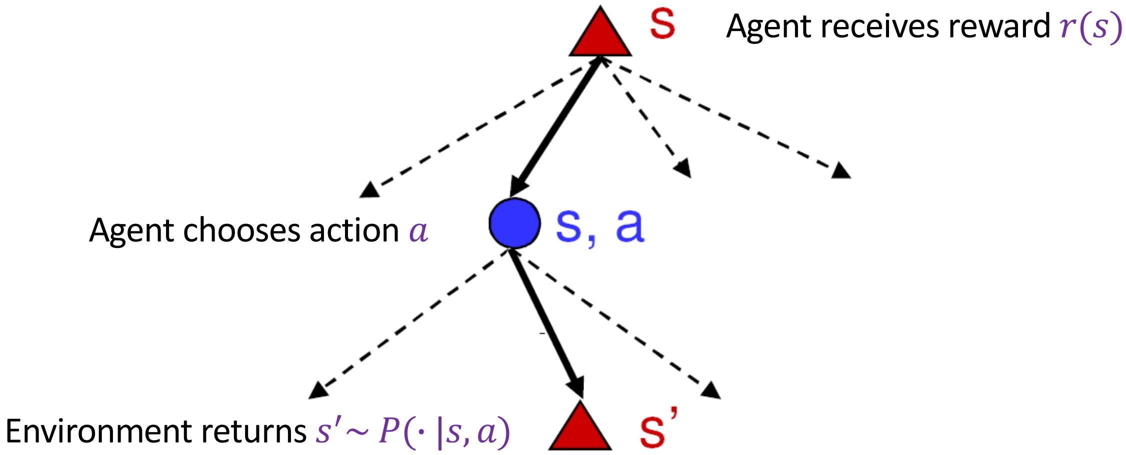


- How do we choose the action?

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

Image source: L. Lazbenik

The Bellman equation

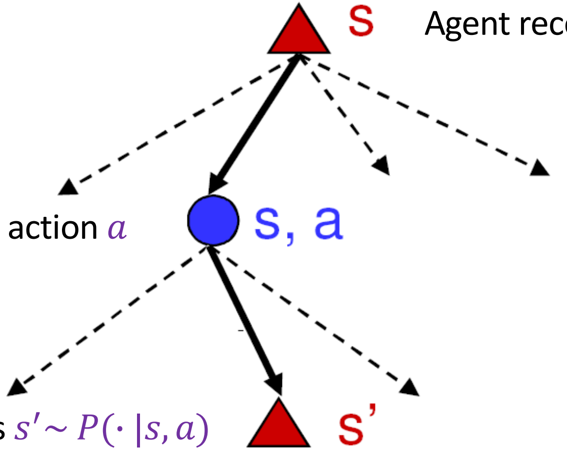


- What is the recursive expression for $V^*(s)$ in terms of $V^*(s')$ - the utilities of its successors?

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

Image source: L. Lazbenik

The Bellman equation



Agent receives reward $r(s)$

Agent chooses action a

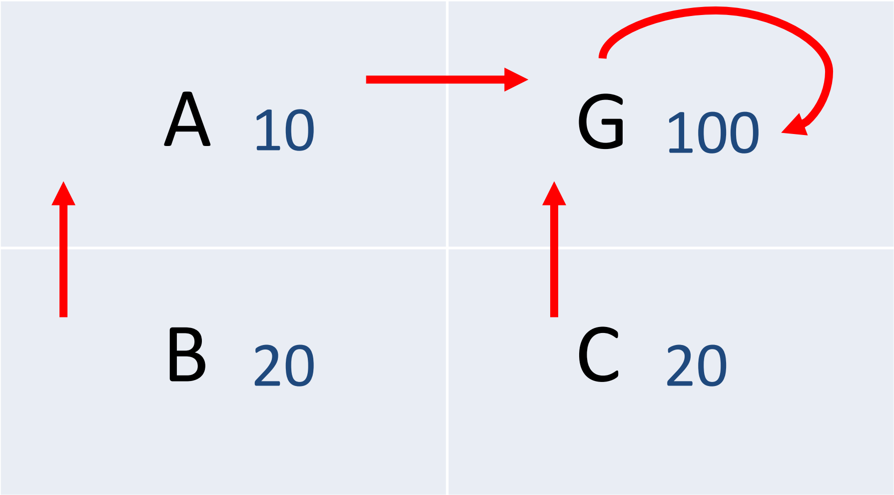
Environment returns $s' \sim P(\cdot\,|s,a)$

- The same reasoning gives the Bellman equation for a general policy:

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

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

Our first reinforcement learning algorithm

- Don't know the whole r and P. But can see interaction trajectory (s_t, a_t, r_t, s_{t+1})
- **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**!

The Q*(s,a) function

- Starting from state s, perform (perhaps suboptimal) action a. THEN follow the optimal policy

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

- Equivalent to

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

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!

Offline Q-Learning

Estimate Q*(s,a) from data $\{(s_t, a_t, r_t, s_{t+1})\}$:

1. Initialize Q(.,.) arbitrarily (eg all zeros)
    1. Except terminal states Q($s_{terminal}$,.)=0
2. Iterate over data until Q(.,.) converges:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_b Q(s_{t+1}, b))$$

Learning rate

Online Q-Learning: Exploration Vs. Exploitation

General question!

- **Exploration:** take an action with unknown consequences
  - Pros:
    - Get a more accurate model of the environment
    - 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

Online 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.

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

Online Q-learning Algorithm

Input: step size α, greedy parameter ϵ

1. Q(.,.)=0
2. for each episode
3. draw initial state $s \sim \mu$
4. while (s not terminal)
5. perform $a = \epsilon$-greedy(Q), receive r, s'
6. $Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_b Q(s',b))$
7. $s \leftarrow s'$
8. endwhile
9. endfor

Note: step 5 can use any other behavior policies

Online Q-learning Algorithm

- Step 5 can use any other behavior policies to choose action a, as long as all actions are chosen frequently enough
- The cumulative rewards during Q-learning may not be the highest
- But after Q-learning converges, can extract an optimal policy:

$$\pi^*(s) \in \operatorname{argmax}_a Q(s,a)$$
$$V^*(s) = \max_a Q^*(s,a)$$

Q-Learning: SARSA

An alternative update rule:

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

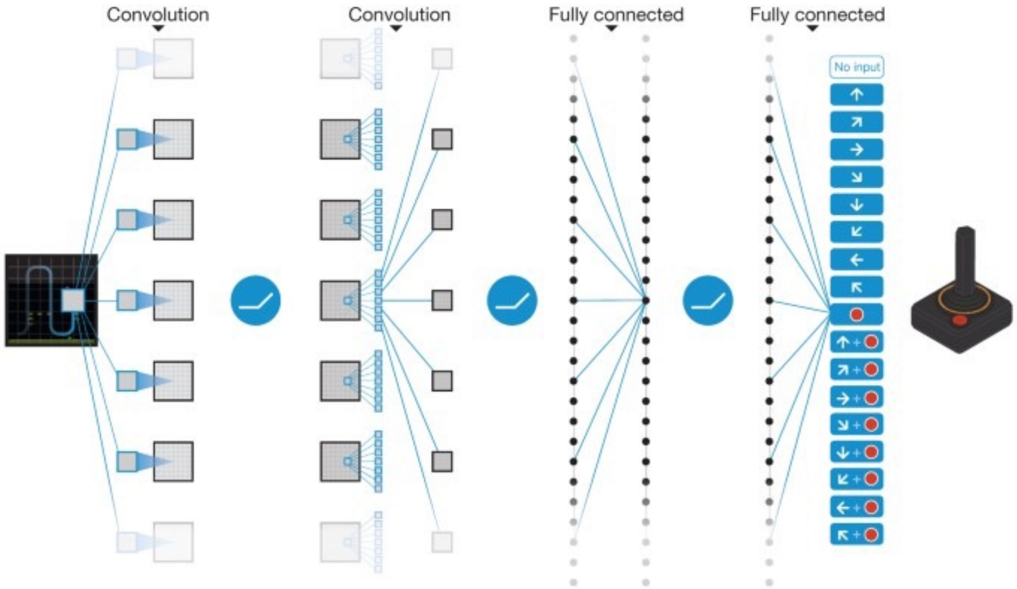
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Learning rate

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

Deep Q-Learning

How do we get Q(*s*,*a*)?



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

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

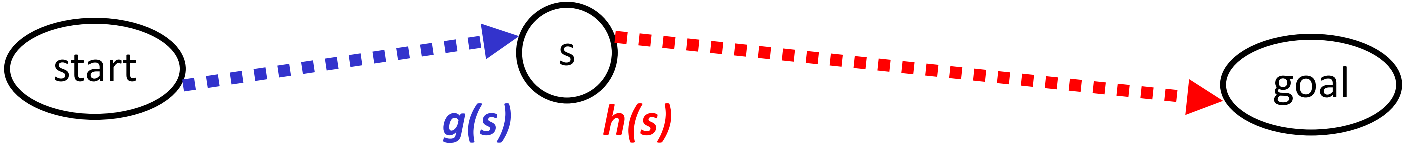
Uninformed vs Informed Search

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)

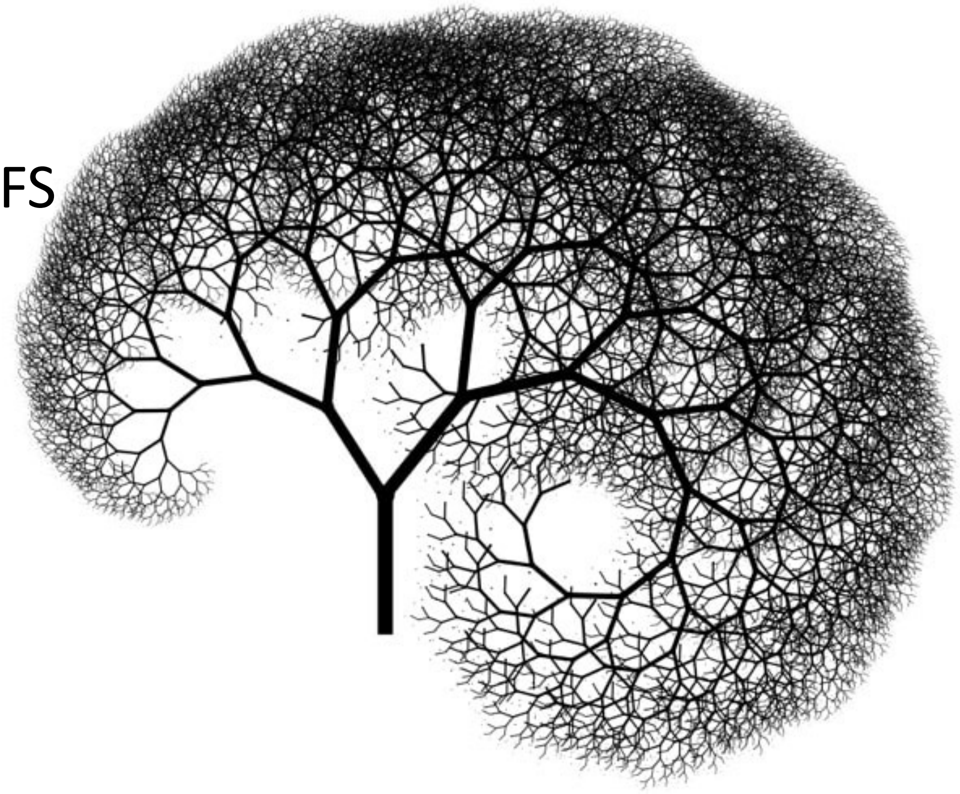


Uninformed Search: Iterative Deepening DFS

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!

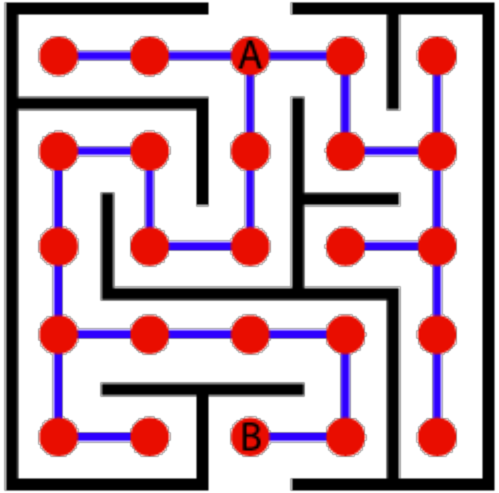


Fractalsaco

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

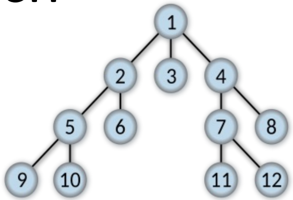


V. Batoćanin

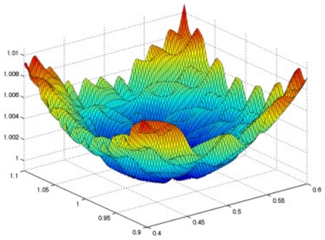
Search vs. Optimization

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

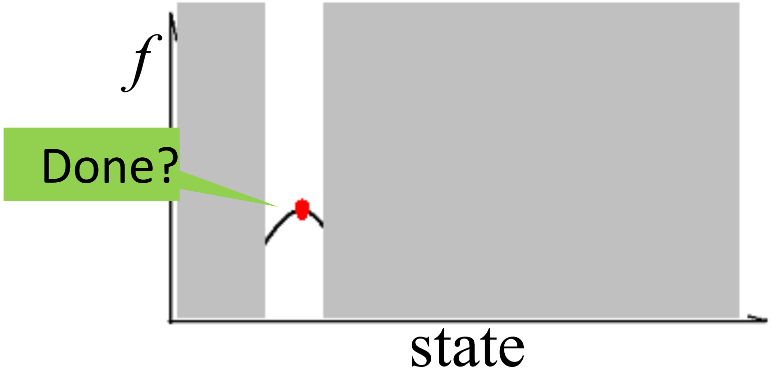
Pseudocode:

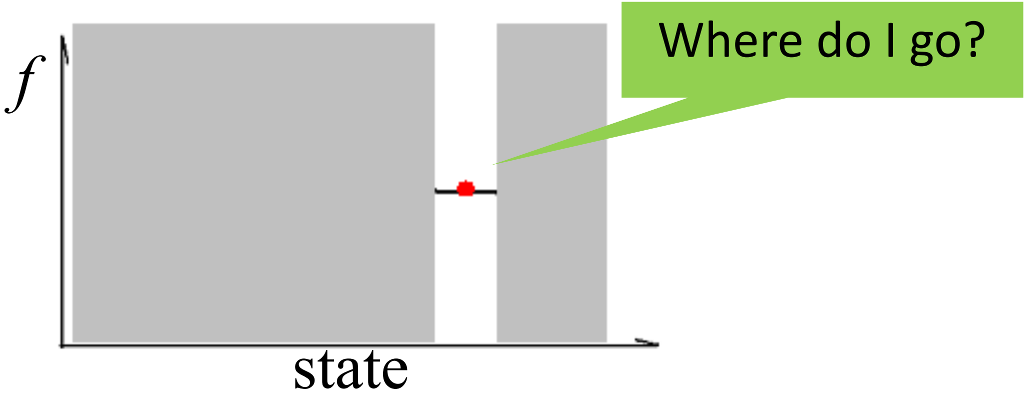
1. Pick initial state s
2. Pick t in **neighbors**(s) with the largest $f(t)$
3. if $f(t) \leq f(s)$ THEN stop, return s
4. $s \leftarrow t$. goto 2.

What could happen? **Local optima!**

Hill Climbing: Local Optima

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

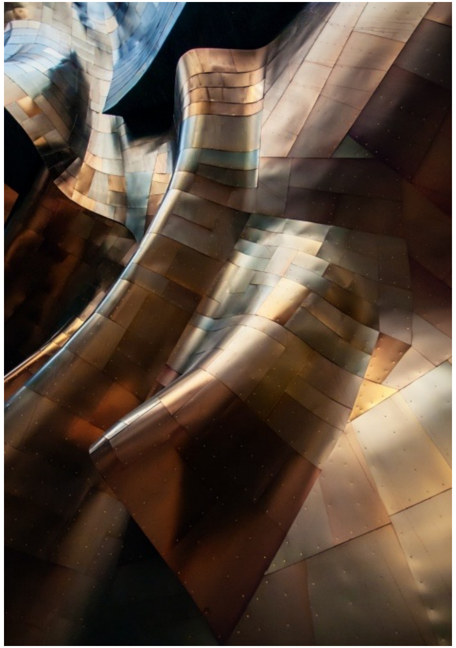




Simulated Annealing

A more sophisticated optimization approach.

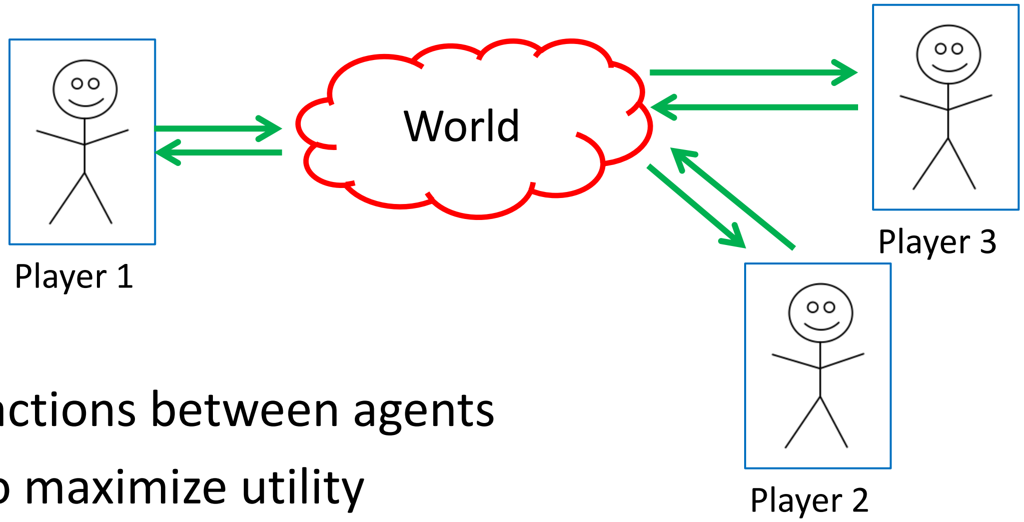
- **Idea**: move quickly at first, then slow down
- Pseudocode:



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

Minimax Search

Note that long games are yield huge computation

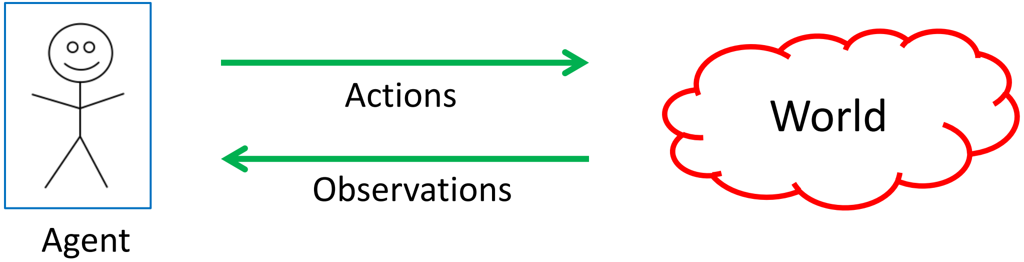
- To deal with this: limit ***d*** for the search depth
- **Q**: What to do at depth *d*, but no termination yet?
  - **A**: Use a heuristic evaluation function *e(x)*

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

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”