CS 540 Introduction to Artificial Intelligence
Reinforcement Learning II / Summary

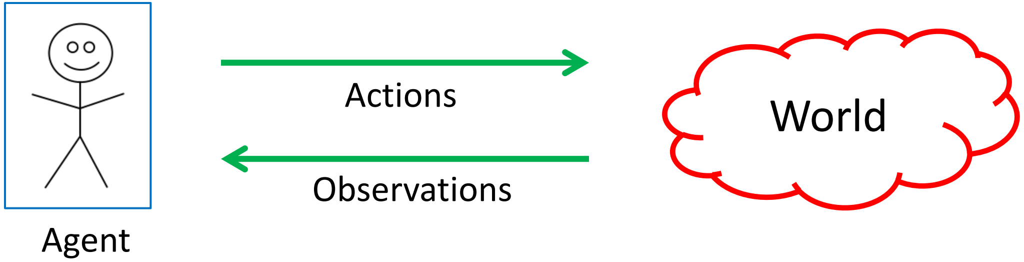
University of Wisconsin-Madison

April 26, 2022

Outline

- Review of reinforcement learning
  - MDPs, value functions, value iteration
- Q-learning
  - Q function, deep Q-learning
- Search + RL Review
  - Uninformed/informed search, optimization, RL

Building The Theoretical Model



Basic setup:

- Set of states, S
- Set of actions A
- Interaction:
  - At time t, observe state $s_t \in S$.
  - Agent makes choice $a_t \in A$.
  - Gets reward r_t, state changes to s_{t+1}, continue

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

Markov Decision Process (MDP)

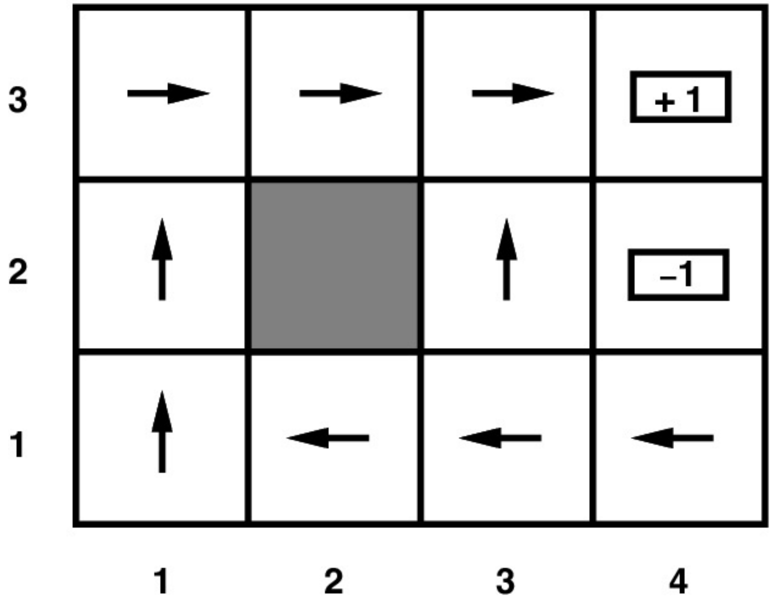
The formal mathematical model $M = (S, A, P, r, \mu, \gamma)$:

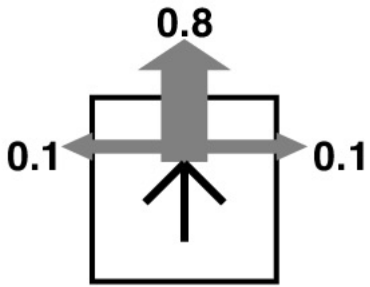
- **State set** S. Initial state s_0. **Action set** A
- **Reward function:** $r(s_t, a_t)$
- **State transition model**: $P(s_{t+1} | s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not earlier history (older actions or states.
  - More generally: $P(r_t, s_{t+1}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

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

Grid World Optimal Policy

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





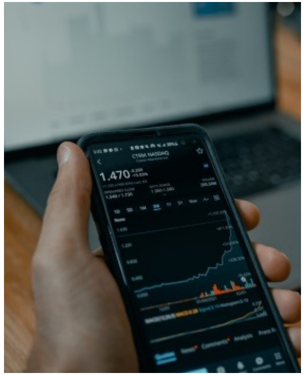
$r(s) = -0.04$ for every non-terminal state

Defining the Optimal Policy

For policy π, **the value** starting from s_0 produced by following that policy:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences } (s_t, a_t, r_t, s_{t+1}) \\ \text{starting from } s_0}} P(\text{sequence})\, U(\text{sequence})$$

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



Discounted Rewards

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

- Solution

$$U(sequence) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots = \sum_{t \geq 0} \gamma^t r_t$$

- Discount factor $\gamma \in (0,1)$
  - Set according to how important **present** is VS **future**
  - Note: has to be less than 1 for convergence

Bellman Equation

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

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

immediate reward

Discounted expected future **rewards**

- Bellman: inventor of dynamic programming

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

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

From Value to Policy

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

- Optimal policy $\pi^* \in argmax_{\pi} V^{\pi}(s_0)$
- At any state s, we should take action $a = \pi^*(s)$
- Define $V^*(s) = V^{\pi^*}(s)$
- If we know V^*, we can extract π^*:

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

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-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-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

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

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

- Equivalent to

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

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

	0	1	2	3
0	Up: -4.10 Right: -3.44 Down: -3.44 Left: -4.10	Up: -3.44 Right: -2.71 Down: -2.71 Left: -4.10	Up: -2.71 Right: -1.90 Down: -1.90 Left: -3.44	Up: -1.90 Right: -1.90 Down: -1.00 Left: -2.71
1	Up: -4.10 Right: -2.71 Down: -4.10 Left: -3.44	Up: -3.44 Right: -1.90 Down: -100.00 Left: -3.44	Up: -2.71 Right: -1.00 Down: -100.00 Left: -2.71	Up: -1.90 Right: -1.00 Down: 0.00 Left: -1.90
2	Up: -3.44 Right: -100.00 Down: -4.10 Left: -4.10	CLIFF		Up: 0.00 Right: 0.00 Down: 0.00 Left: 0.00
	↑ START			↑ END

Q-TABLE

Possible Actions

Possible States (Row, Column)	Up	Right	Down	Left
0, 0	-4.10	-3.44	-3.44	-4.10
0, 1	-3.44	-2.71	-2.71	-4.10
0, 2	-2.71	-1.90	-1.90	-3.44
0, 3	-1.90	-1.90	-1.00	-2.71
1, 0	-4.10	-2.71	-4.10	-3.44
1, 1	-3.44	-1.90	-100.00	-3.44
1, 2	-2.71	-1.00	-100.00	-2.71
1, 3	-1.90	-1.00	0.00	-1.90
2, 0	-3.44	-100.00	-4.10	-4.10
2, 1	0.00	0.00	0.00	0.00
2, 2	0.00	0.00	0.00	0.00
2, 3	0.00	0.00	0.00	0.00

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 prevent you from discovering the true optimal strategy

Q-Learning: ε-Greedy Behavior Policy

Getting data with both **exploration and exploitation**

- With probability ε, take a random action; else the action with the highest (current) Q(*s*,*a*) value.

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

The 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

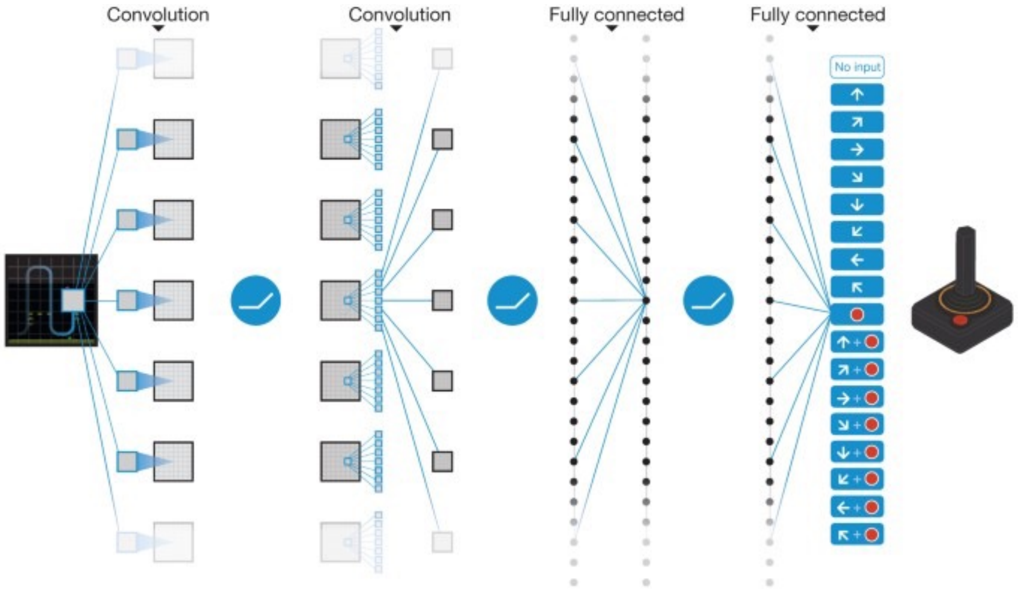
The Q-learning algorithm

- 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:

$$\pi^*(s) \in \mathrm{argmax_a Q(s, a)}$$
$$V^*(s) = \max_a Q^*(s, a)$$

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

Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- A. Visit every state and try every action
- B. Perform at least 20,000 iterations.
- C. Re-start with different random initial table values.
- D. Prioritize exploitation over exploration.

Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- **A. Visit every state and try every action**
- B. Perform at least 20,000 iterations.
- C. Re-start with different random initial table values.
- D. Prioritize exploitation over exploration.

Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- **A. Visit every state and try every action**
- B. Perform at least 20,000 iterations. (No: this is dependent on the particular problem, not a general constant).
- C. Re-start with different random initial table values. (No: this is not necessary in general).
- D. Prioritize exploitation over exploration. (No: insufficient exploration means potentially unupdated state action pairs).

Search and RL Review

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

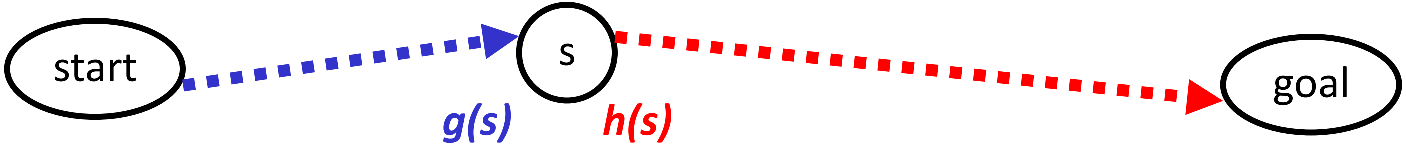
Uninformed vs Informed Search

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)

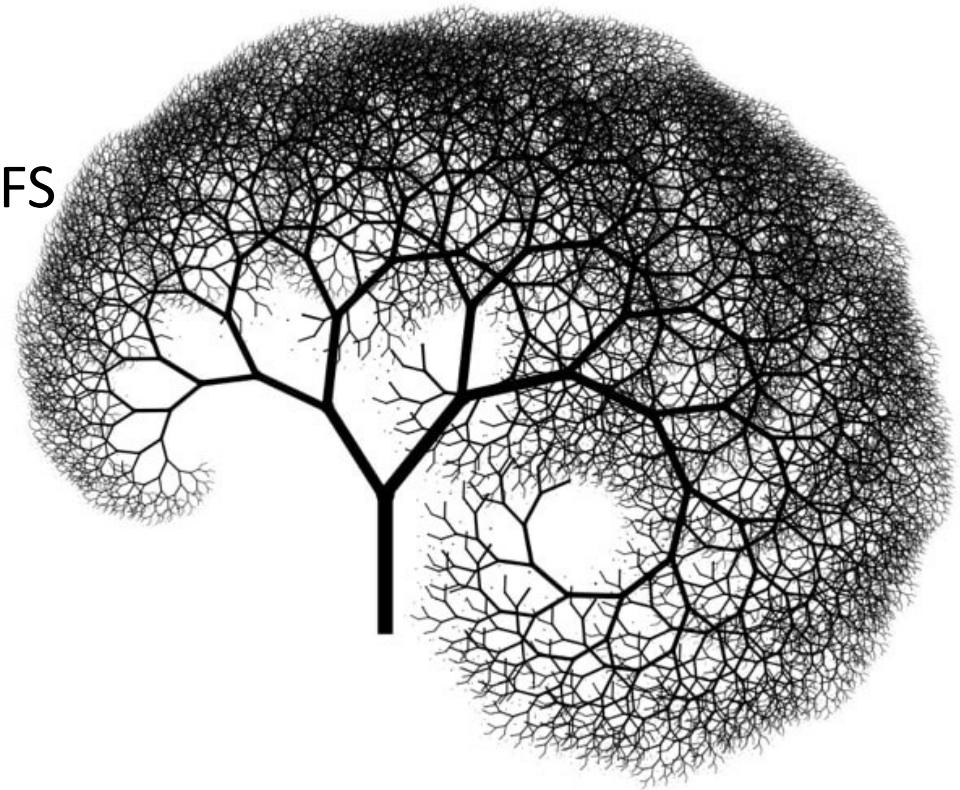


Uninformed Search: Iterative Deepening DFS

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!

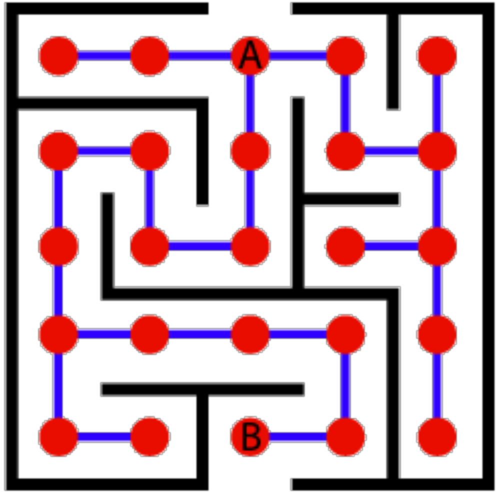


Fractalsaco

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

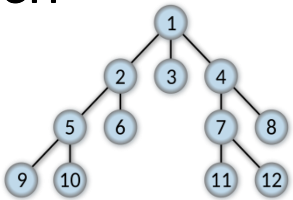


V. Batoćanin

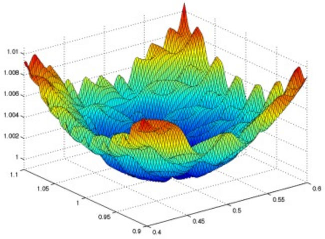
Search vs. Optimization

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

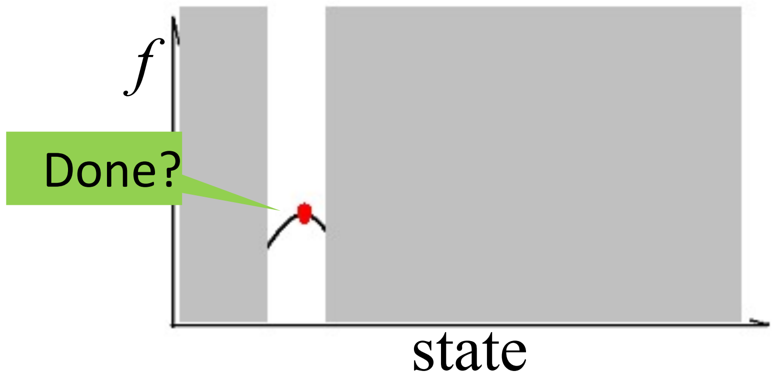
Pseudocode:

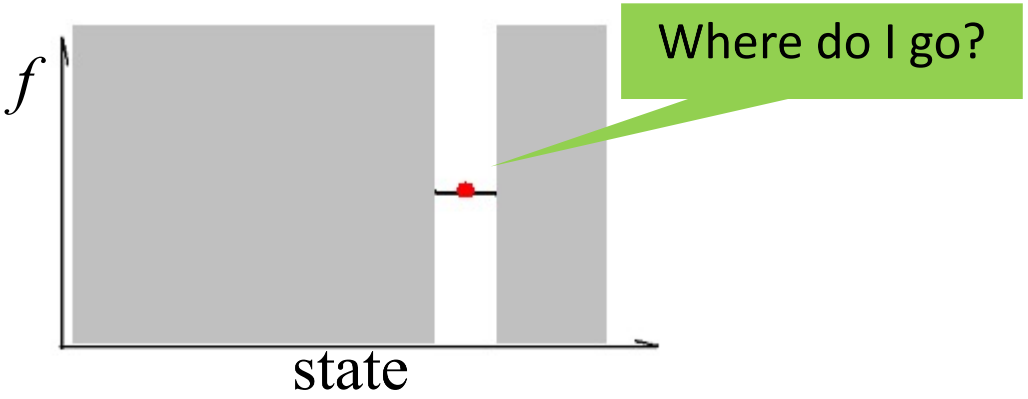
1. Pick initial state s
2. Pick t in **neighbors**(s) with the largest $f(t)$
3. if $f(t) \leq f(s)$ THEN stop, return s
4. $s \leftarrow t$. goto 2.

What could happen? **Local optima!**

Hill Climbing: Local Optima

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





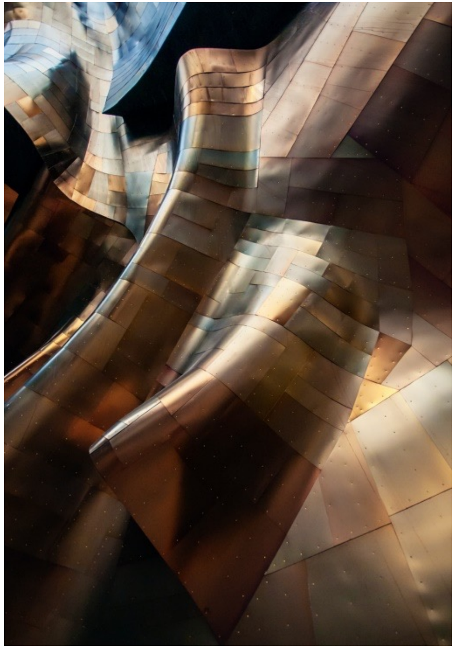
Simulated Annealing

A more sophisticated optimization approach.

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

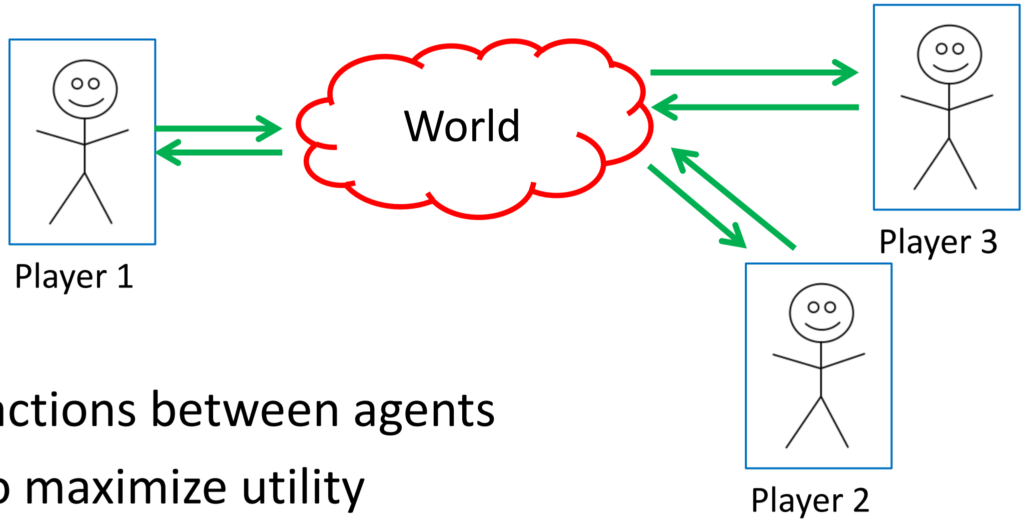
Pick initial state s
For k = 0 through k_{max}:
    $\boldsymbol{T}$ ← temperature($(k+1)/k_{max}$)
    Pick a random neighbour, t ← neighbor(s)
    If $\boldsymbol{f}(s) \leq \boldsymbol{f}(t)$, then $s \leftarrow t$
    Else, with prob. $P(\boldsymbol{f}(s), \boldsymbol{f}(t), \boldsymbol{T})$ then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

The interesting bit



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

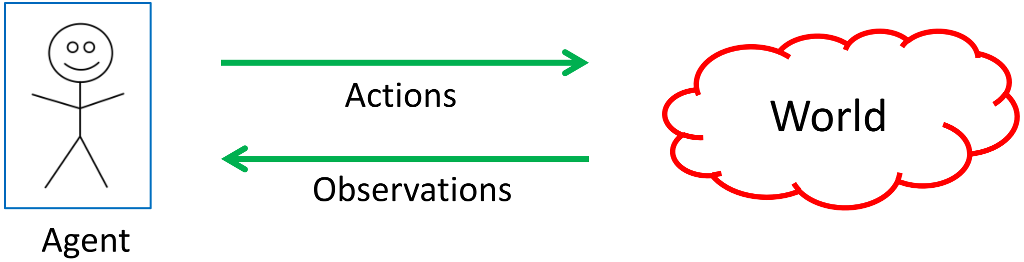
- To deal with this: limit ***d*** for the search depth
- **Q**: What to do at depth *d*, but no termination yet?
  - **A**: Use a heuristic evaluation function *e(x)*

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 $\max\{$MINIMAX$(y, d-1) : y$ is a child of $x\}$
  else return $\min\{$MINIMAX$(y, d-1) : y$ 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”