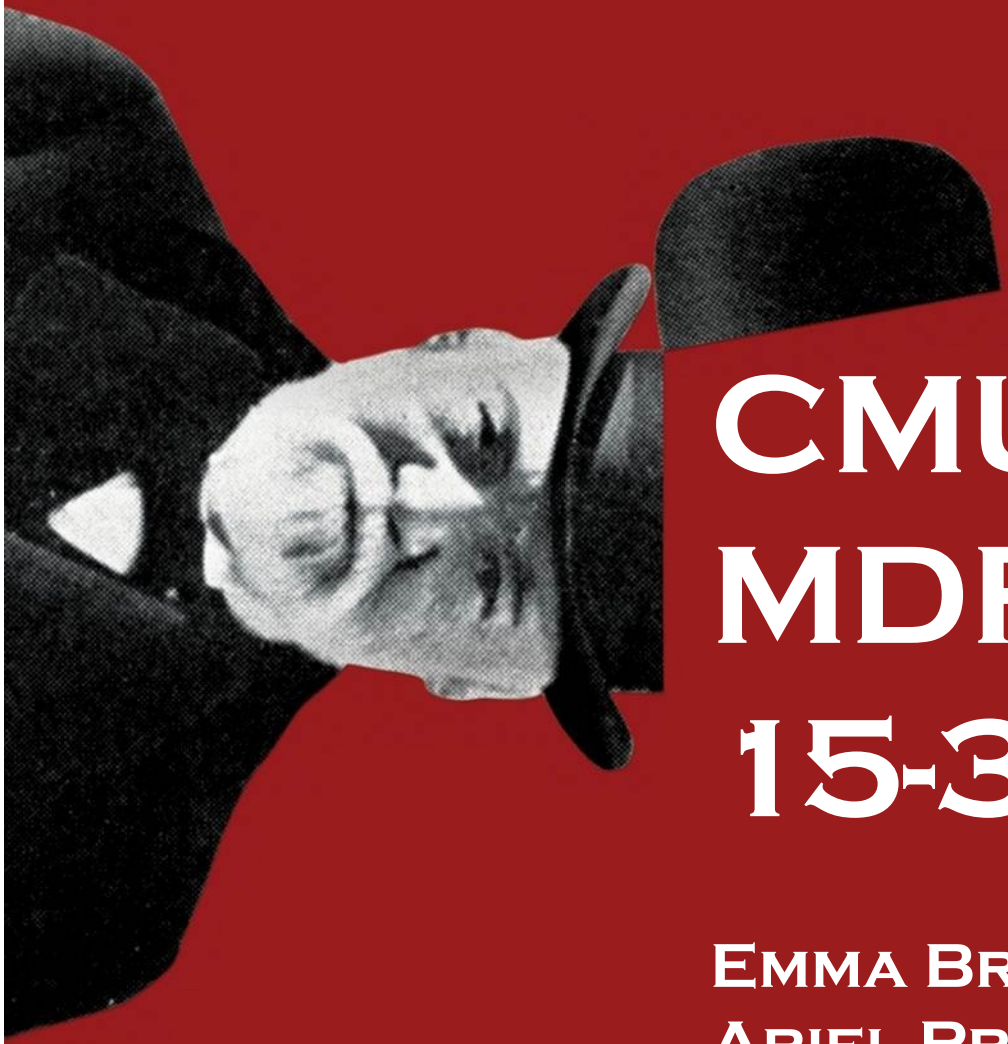
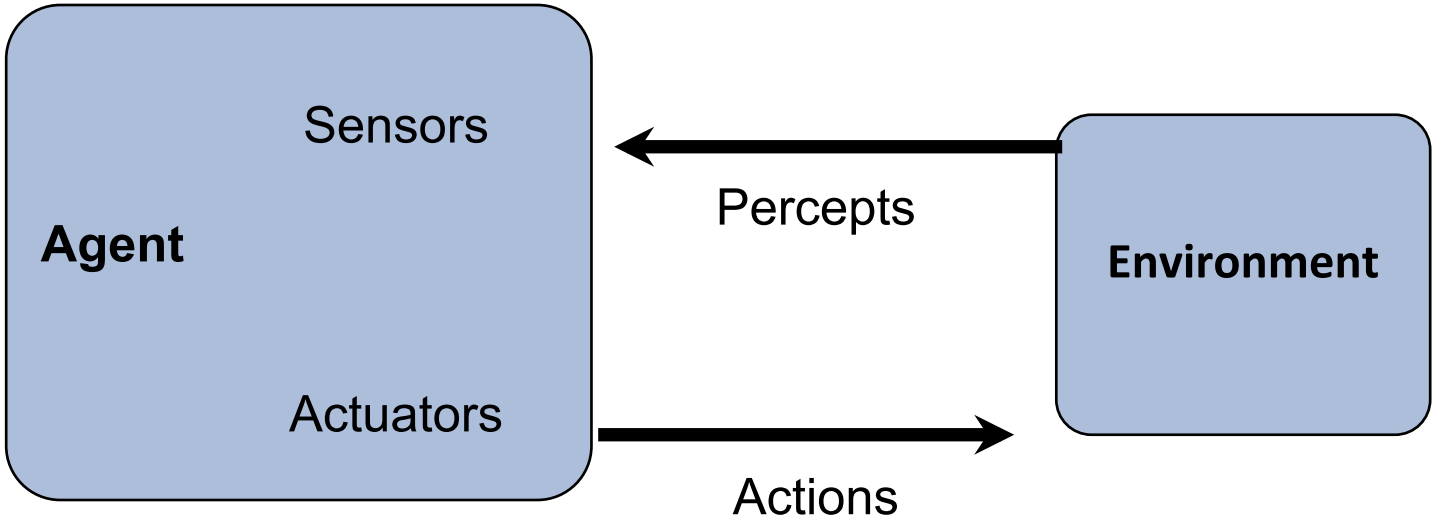


CMU MDPs 15-381/781

Emma Brunskill (THIS TIME)
Ariel Procaccia

- DeepMind

So long certainty…



- Until now, result of taking an action in a state was deterministic

Reasoning Under Uncertainty

	Actions Don't Change State of the World	Actions Change State of the World
Learn model of outcomes	Multi-armed bandits	Reinforcement Learning
Given model of stochastic outcomes	Decision theory	Markov Decision Processes

Expectation

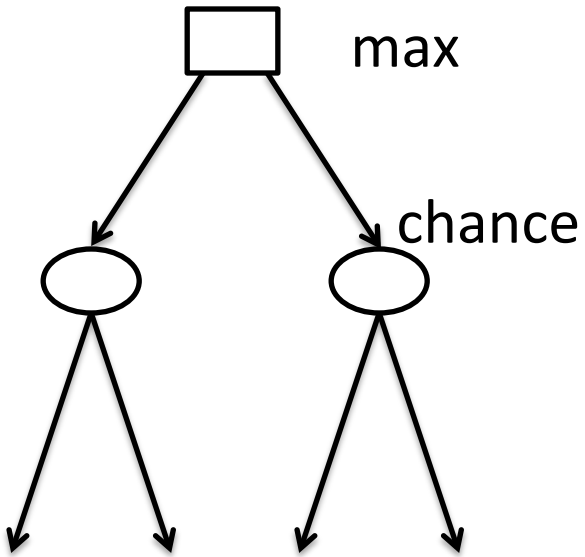
- The expected value of a function of a random variable is the average, weighted by the probability distribution over outcomes

- Example: expected time if take the bus
- Time: 5 min + 30 min
- Probability: 0.7 + 0.3

⇨ 12.5 min

Where Do Probabilities Come from?

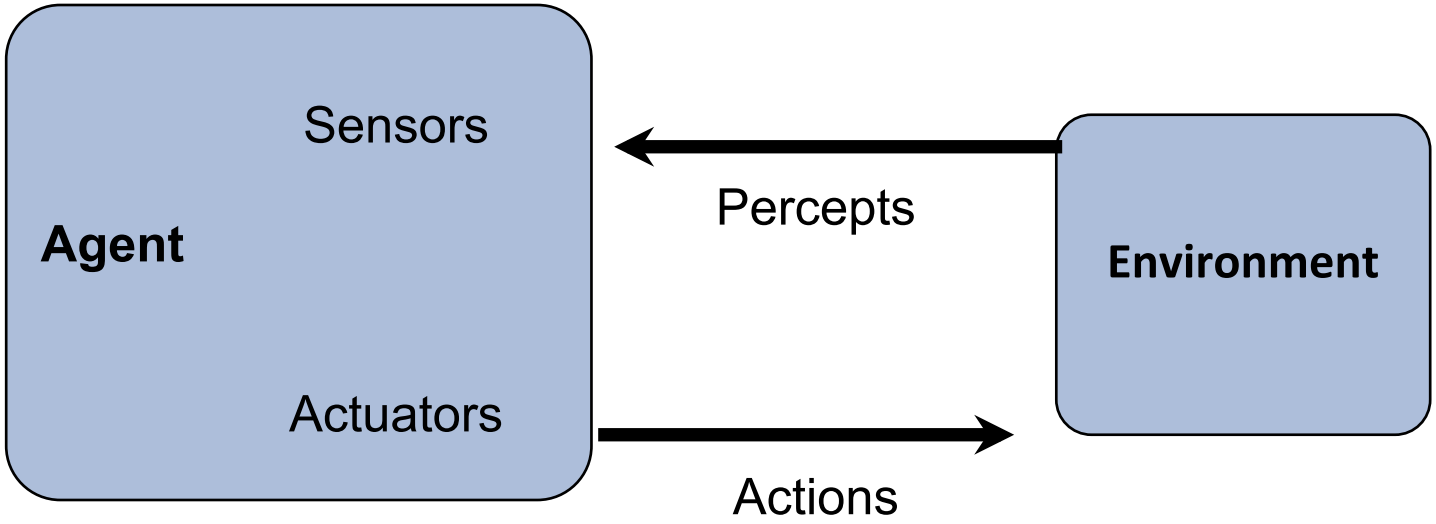
- Models
- Data
- For now assume we are given the probabilities for any chance node



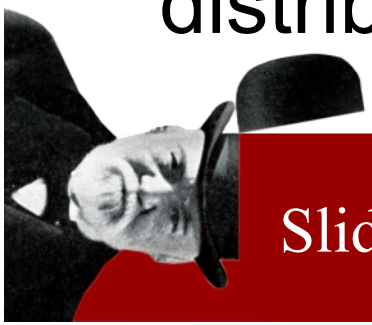
Reasoning Under Uncertainty

Given model of stochastic outcomes	Decision theory	**Markov Decision Processes**
	Actions Don't Change State of the World	Actions Change State of the World

(Stochastically) Change the World

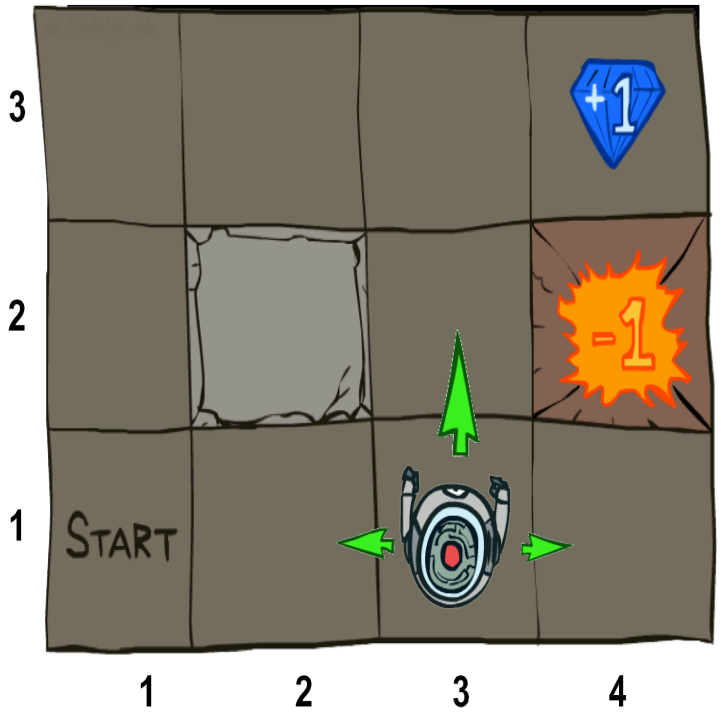


- Like planning/search, actions impact world
- But exact impact is stochastic: probability distribution over next states



Example: Grid World

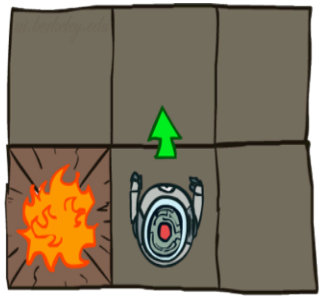
- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- The agent receives rewards each time step
  - Small "living" reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards
- Noisy movement: actions do not always go as planned
  - 80% of the time, action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have gone, agent stays put

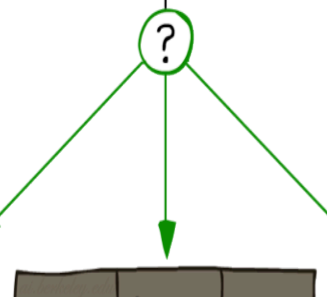


Grid World Actions

Deterministic Grid World

Stochastic Grid World

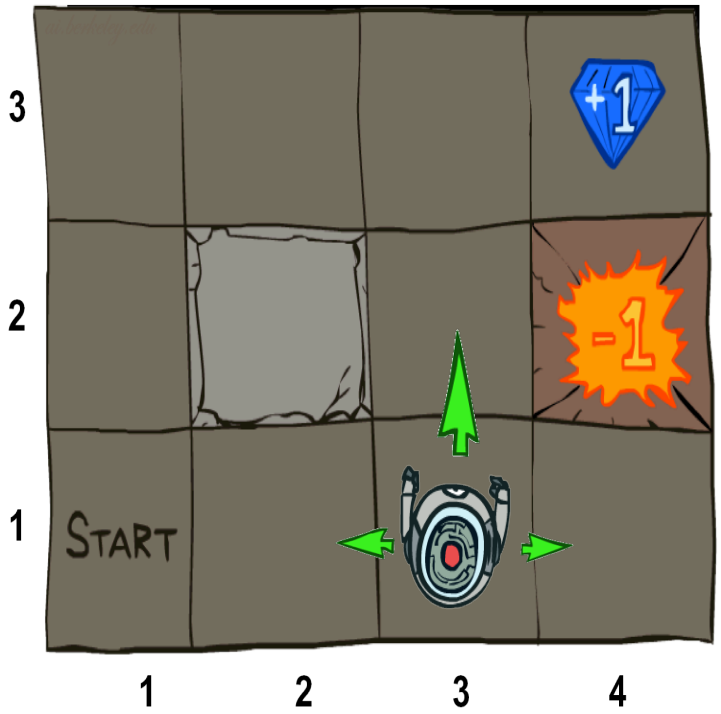




Slide adapted from Klein and Abbeel

Markov Decision Processes

- Set of states $s \in S$
- Set of actions $a \in A$
- Transition func. T(s, a, s')
  - Probability that a from s leads to s', i.e., P(s'| s, a)
- Reward func. R(s, a, s') / R(s) / R(s,a)
- Start state or states (could be all S)
- Maybe a terminal state
- Discount factor
- MDPs are non-deterministic search problems



Slide adapted from Klein and Abbeel

Markov Decision Processes

Markov Property

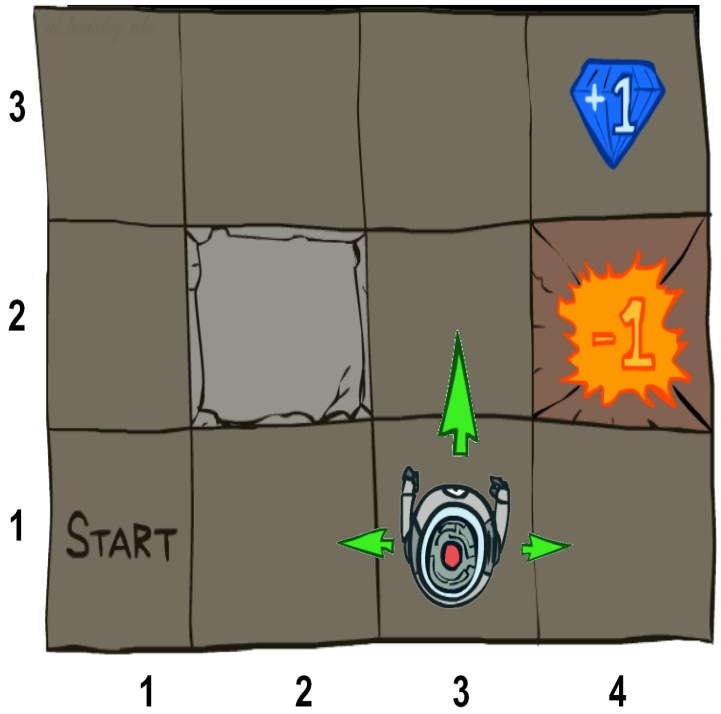
- Called **Markov** decision process because the outcome of an action depends only on the current state
- $p(s_{t+1}|s_1,a_1,s_2,a_2,\ldots s_t,a_t)=p(s_{t+1}|s_t,a_t)$

Policies

- In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal
- In MDPs instead of plans, we have a policies
- A policy $\pi^*: S \rightarrow A$
  - Specifies what action to take in each state

Poll: How Many Policies?

- How many non-terminal states?
- How many actions?
- How many deterministic policies over non-terminal states?



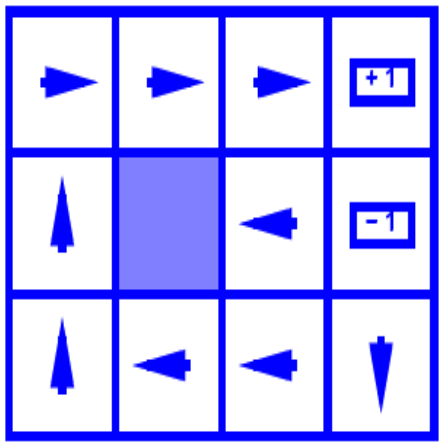
Optimal Policies

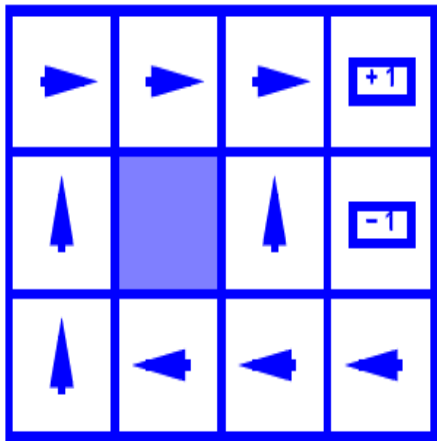
- Optimal plan had minimal cost to reach goal
- Utility or value of a policy π starting in state s is the expected sum of future rewards will receive by following π starting in state s

- Optimal policy has maximal expected sum of rewards from following it

Optimal Policies

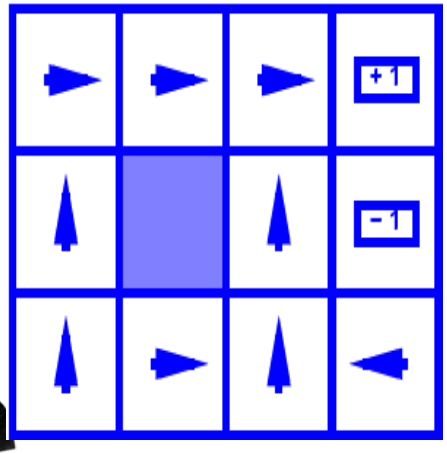
R(s) = -0.01

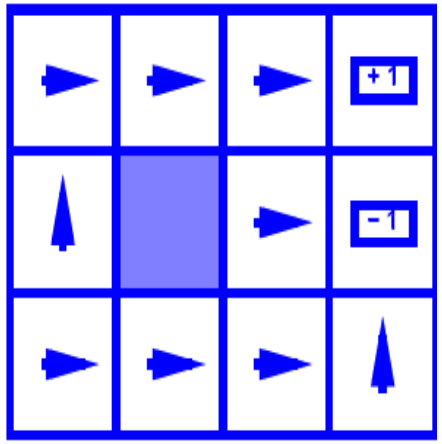




R(s) = -0.03

R(s) = -0.4

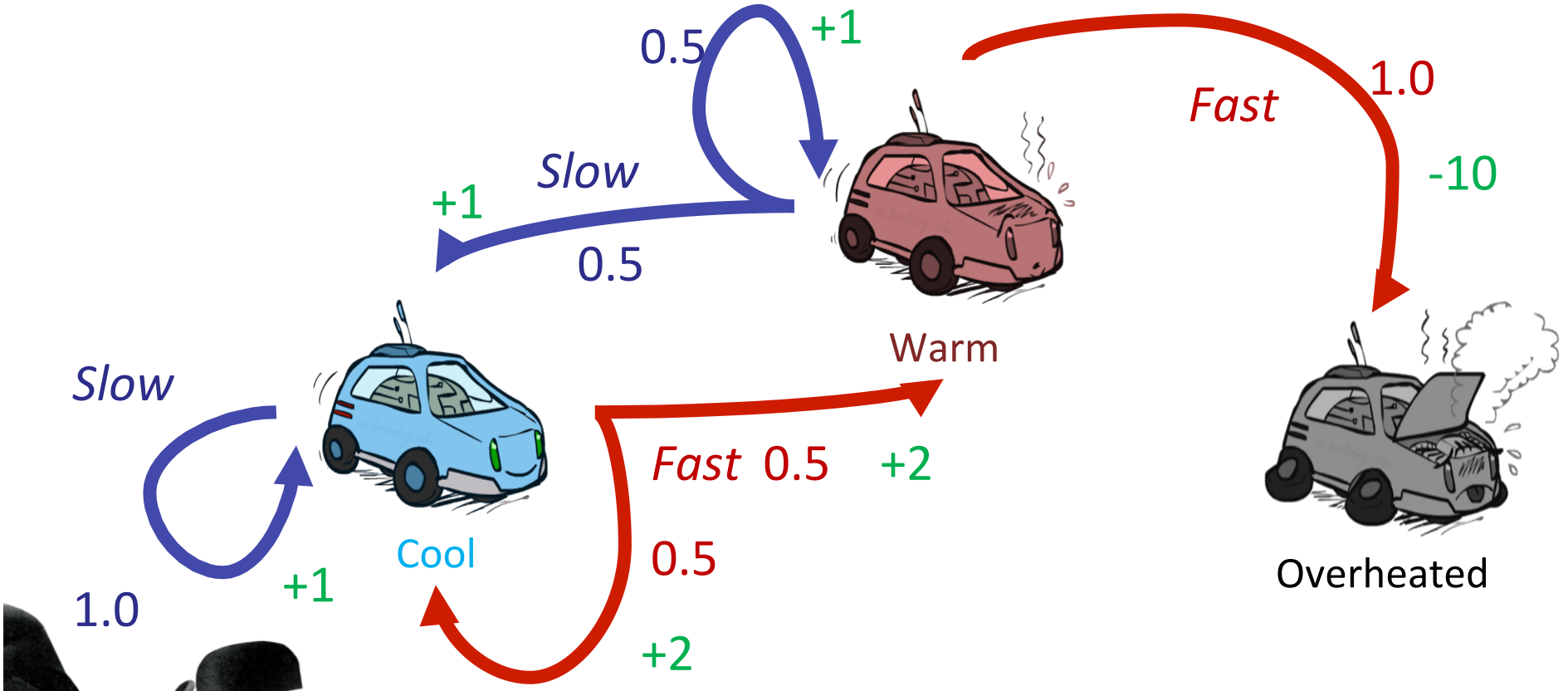




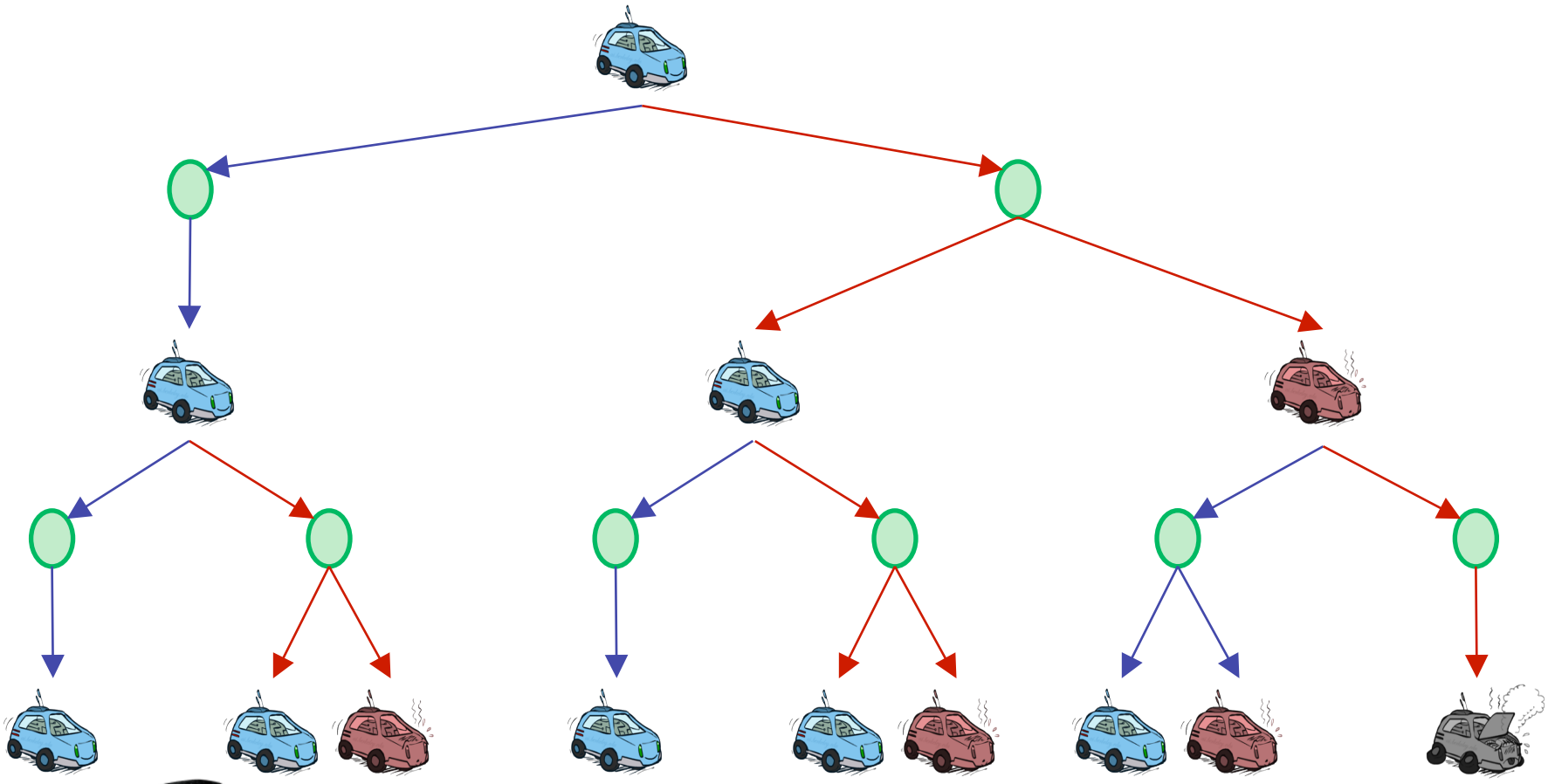
R(s) = -2.0

Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward

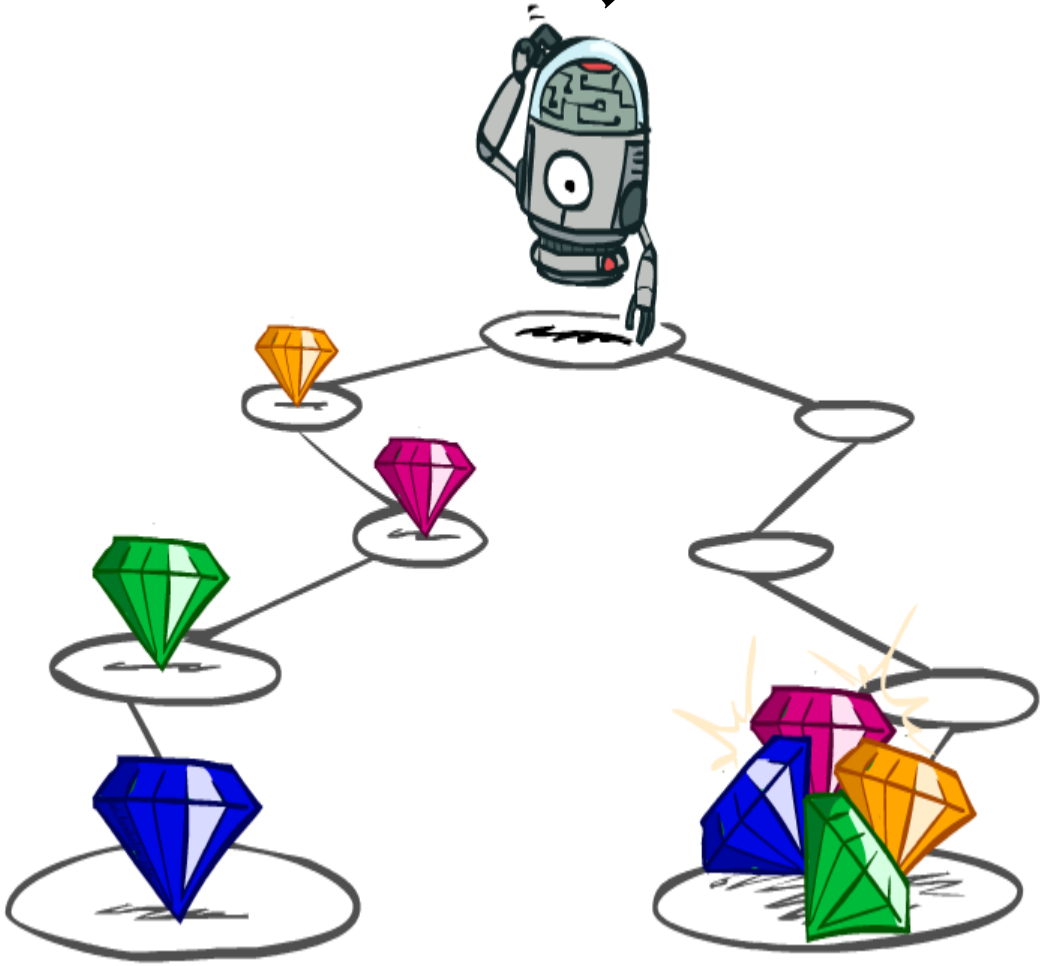


Racing Search Tree



Slide adapted from Klein and Abbeel

Utilities of Sequences



Utilities of Sequences

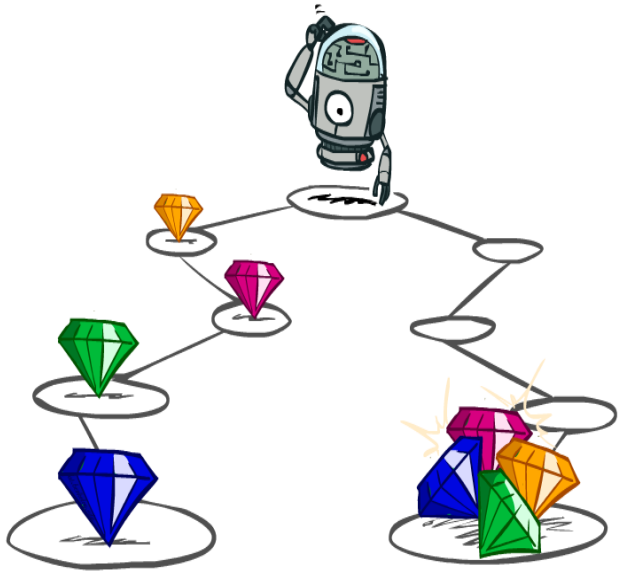
- What preferences should an agent have over reward sequences?

- More or less?

  [1, 2, 2] or [2, 3, 4]

- Now or later?

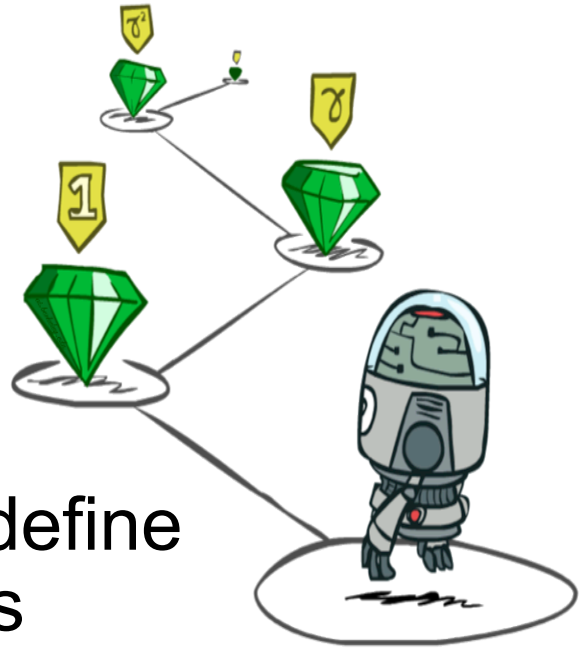
  [0, 0, 1] or [1, 0, 0]



Stationary Preferences

- Theorem: if we assume stationary preferences:

$$[a_1, a_2, \ldots] \succ [b_1, b_2, \ldots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \ldots] \succ [r, b_1, b_2, \ldots]$$



- Then: there are only two ways to define utilities over sequences of rewards
  - Additive utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots$
  - Discounted utility: $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$

What are Discounts?

- It's reasonable to prefer rewards now to rewards later
- Decay rewards exponentially

1

Worth Now

γ

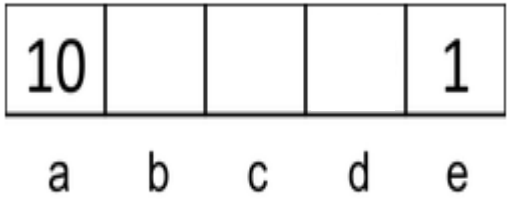
Worth Next Step

γ^2

Worth In Two Steps

Discounting $U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 \cdots$

- Given:



  - Actions: East, West
  - Terminal states: a and e (end when reach one or the other)
  - Transitions: deterministic
  - Reward for reaching a is 10 (regardless of initial state & action, e.g. r(s,action,a) = 10), reward for reaching e is 1, and the reward for reaching all other states is 0
- Quiz 1: For γ = 1, what is the optimal policy?
- Quiz 2: For γ = 0.1, what is the optimal policy for states b, c and d?
- Quiz 3: For which γ are West and East equally good when in state d?

Infinite Utilities?!

- Problem: What if the game lasts forever? Do we get infinite rewards?
- Solutions:
  - Finite horizon: (similar to depth-limited search)
    - Terminate episodes after a fixed T steps (e.g. life)
    - Gives nonstationary policies (π depends on time left)
  - Discounting: use $0 < \gamma < 1$

$$U([r_0, \ldots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{\max}/(1-\gamma)$$

    - Smaller γ means smaller "horizon" – shorter term focus
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached (like "overheated" for racing)

Recap: Defining MDPs

- Markov decision processes:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions P(s'|s,a) (or T(s,a,s'))
  - Rewards R(s,a,s') (and discount γ)
- MDP quantities so far:
  - Policy = Choice* of action for each state
  - Utility/Value = sum of (discounted) rewards

Value of a Policy in Each State

- Expected immediate reward for taking action prescribed by policy π for that state
- And expected future reward get after taking that action from that state and following π

$$V^{\pi}(s) = \sum_{s' \in S} p(s' \mid s, \pi(s)) \Big[R(s, \pi(s), s') + \gamma V^{\pi}(s') \Big]$$

- Future reward depends on horizon (how many more steps get to act). For now assume infinite

Q: State-Action Value

- Expected immediate reward for taking action
- And expected future reward get after taking that action from that state and following π

$$Q^{\pi}(s,a) = \sum_{s' \in S} p(s' \mid s,a)\left[R(s,a,s') + \gamma V^{\pi}(s')\right]$$

Optimal Value V* and π*

- Optimal value: Highest possible value for each s
- Satisfies the **Bellman Equation**

$$V^*(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s_i, a, s') + \gamma V^*(s_j) \right] \right)$$

- Optimal policy

$$\pi^*(s_i) = \operatorname*{argmax}_a Q(s_i, a)$$

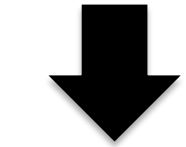
$$= \operatorname*{argmax}_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s_i, a, s') + \gamma V^*(s_j) \right] \right)$$

- Want to find these optimal values!

Value Iteration

- Bellman equation inspires an update rule

$$V^*(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s, a, s') + \gamma V^*(s_j) \right] \right)$$



$$V_k(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s, a, s') + \gamma V_{k-1}(s_j) \right] \right)$$

- Form of *dynamic programming*

Also Called a Bellman Backup

$$V_k(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s, a, s') + \gamma V_{k-1}(s_j) \right] \right)$$

- In shorthand, for performing the above computation for all states,

$$V_k = BV_{k-1}$$

Value Iteration Algorithm

1. Initialize $V_0(s_i)=0$ for all states s_i, Set K=1
2. While k < desired horizon or (if infinite horizon) values have converged
   - For all s,

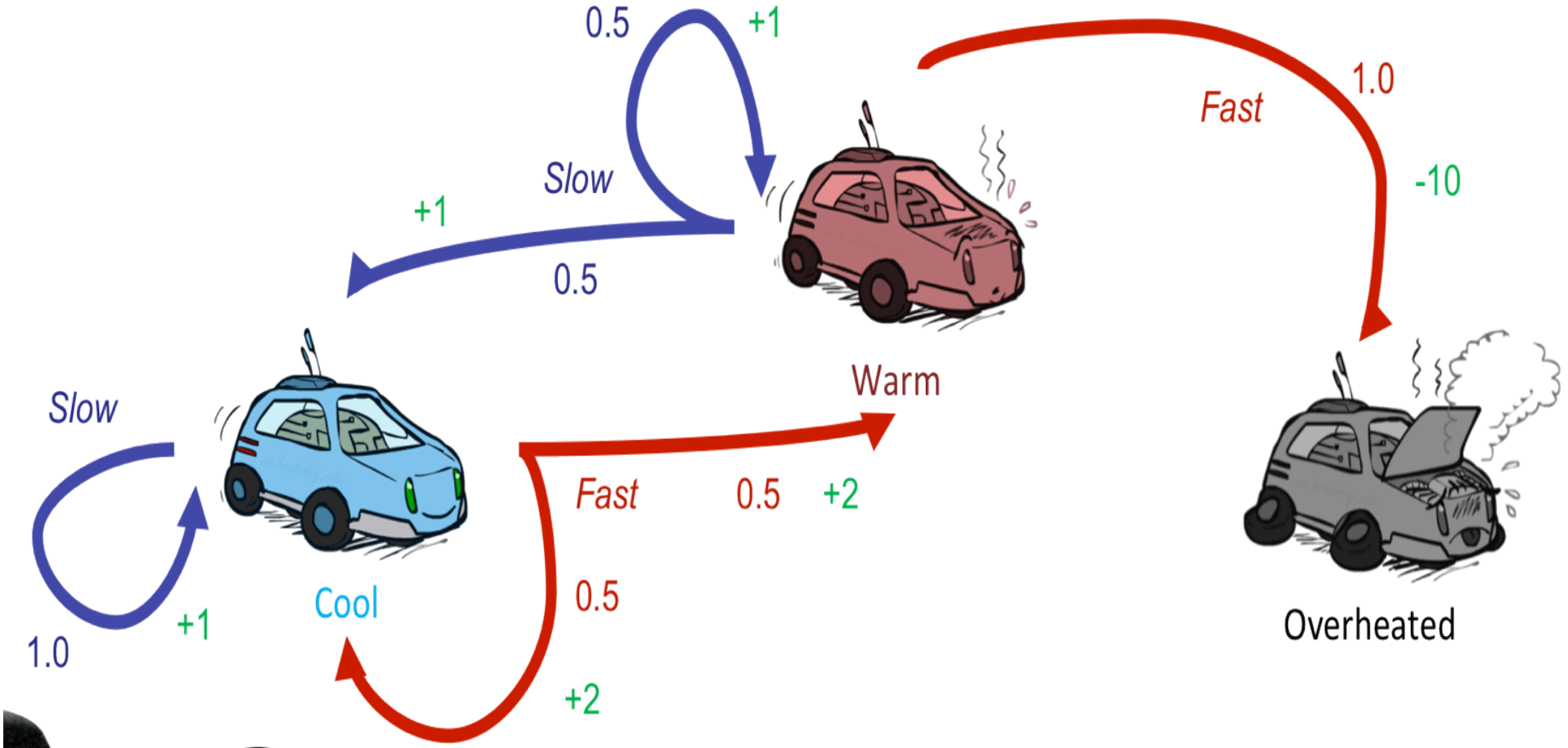
$$V_k(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s,a,s') + \gamma V_{k-1}(s_j) \right] \right)$$

3. Extract Policy

$$\pi_k(s_i) = \operatorname{argmax}_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s,a,s') + \gamma V_{k-1}(s_j) \right] \right)$$

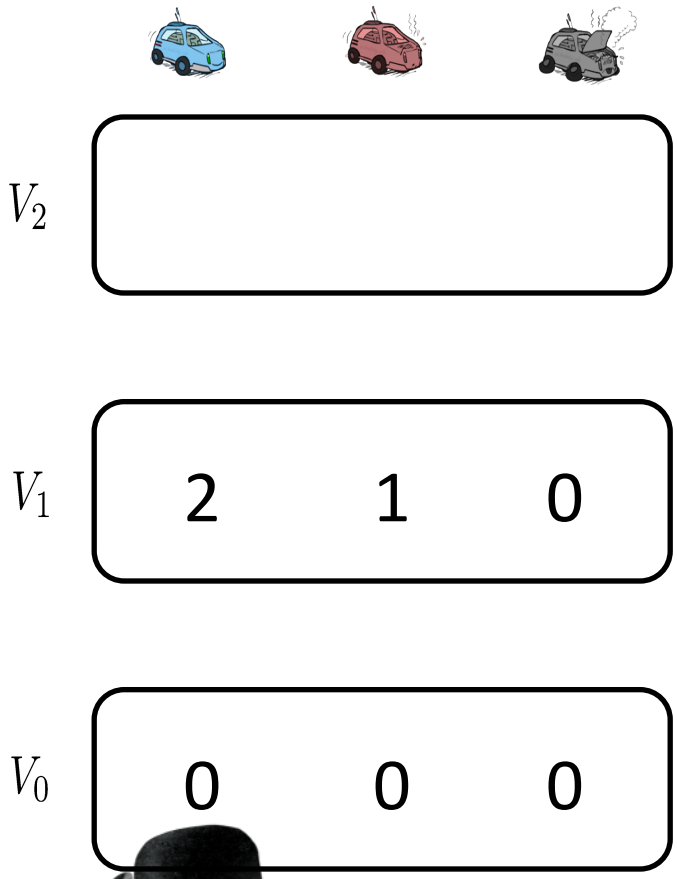
Calculate V_2(warmCar)

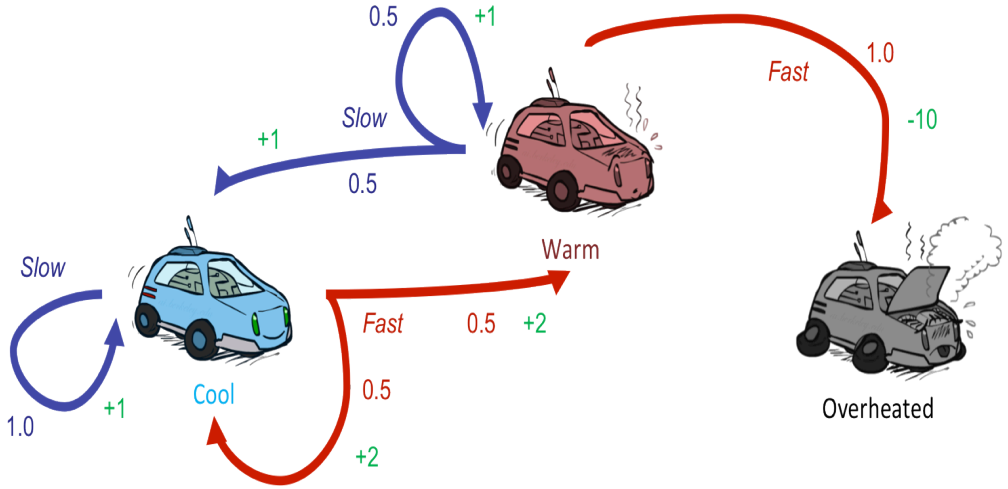
Assume γ=1



Slide adapted from Klein and Abbeel

For General Practice, check can calculate V_2(warmCar)





Assume γ=1

$$V_k(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s,a,s') + \gamma V_{k-1}(s_j) \right] \right)$$

Computational Cost for 1 Update of V(s) for all s in Value Iteration?

- For all s,

$$V_k(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s,a,s') + \gamma V_{k-1}(s_j) \right] \right)$$

Will Value Iteration Converge for Infinite Horizon Problems?

Contraction Operator

- Let O be an operator
- If |OV – OV'| <= |V-V|'
- Then O is a contraction operator

Will Value Iteration Converge?

- Yes, if discount factor $\gamma < 1$ or end up in a terminal state with probability 1
- Bellman equation is a contraction if discount factor, $\gamma < 1$
- If apply it to two different value functions, distance between value functions shrinks after apply Bellman equation to each

Properties of Contraction

- Only has 1 fixed point (the point reach if apply a contraction operator many times)
  - If had two, then would not get closer when apply contraction function, violating definition of contraction
- When apply contraction function to any argument, value must get closer to fixed point
  - Fixed point doesn't move
  - Repeated function applications yield fixed point

VI Converges

- Value iteration converges to unique solution which is optimal value function

- Proof: $\lim_{k\to\infty} V_k = V^*$

$$\|V_{k+1} - V^*\|_\infty = \|BV_k - V^*\|_\infty \le \gamma \|V_k - V^*\|_\infty \le \ldots$$
$$\le \gamma^{k+1} \|V_0 - V^*\|_\infty \to 0$$

Discuss and Report Back: Does Initialization Impact Final Value?

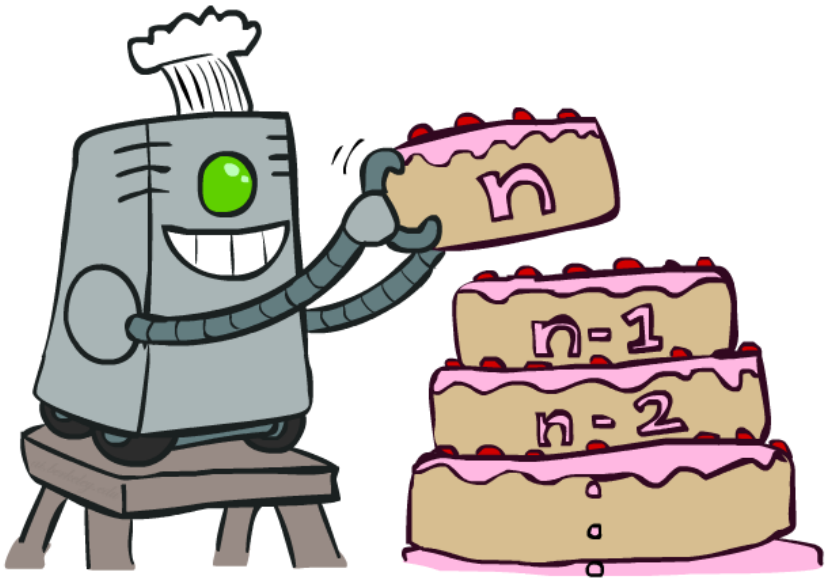
Value Iteration Algorithm

1. Init $V_0(s_i)$ for all states s_i
2. k=1
3. While k < desired horizon or (if infinite horizon) values have converged
   - For all s,

$$V_k(s_i) = \max_a \left(\sum_{s_j \in S} p(s_j \mid s_i, a) \left[R(s,a,s') + \gamma V_{k-1}(s_j) \right] \right)$$

Value Iteration in Infinite Horizon

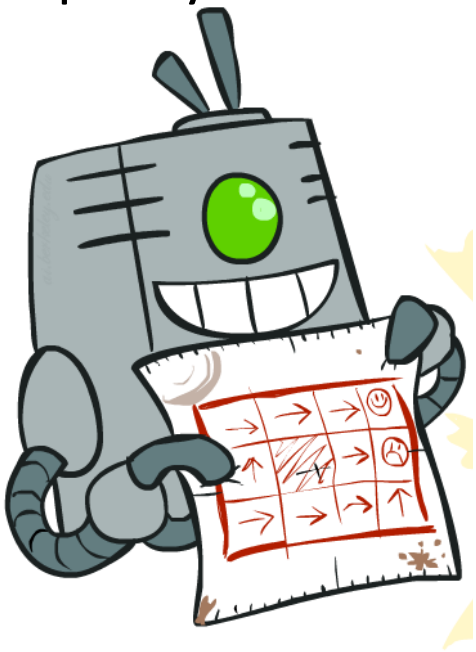
- Always have optimal values as if had only t more decisions to make
- Extracting optimal policy for t-th step yields optimal action should take, if have t more steps to act
  - But not for (t-1), (t-2)… steps
- Before convergence, these are approximations (because actually get to act forever!)
  - After convergence, value is always the same if do another update, and so is the policy



Drawing by Ketrina Yim

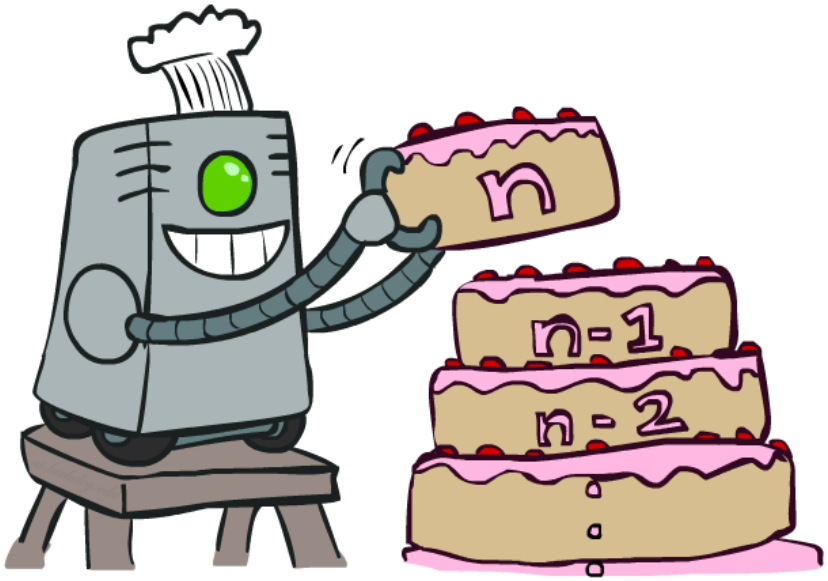
Policy Iteration

Maintain value of following a particular policy forever



Value Iteration

Maintain optimal values if have n more actions

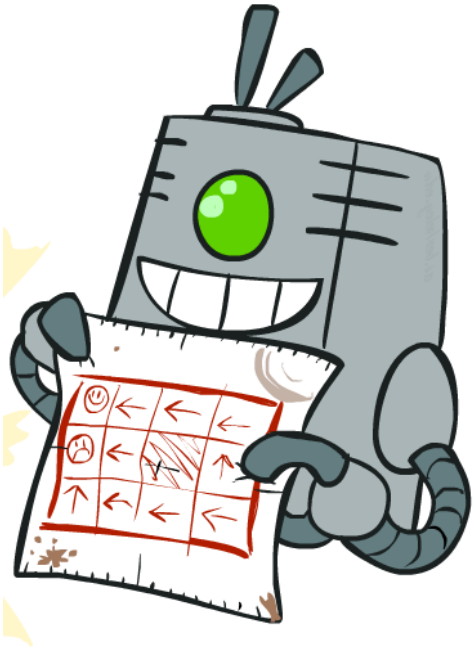


Drawings by Ketrina Yim

Policy Iteration for Infinite Horizon

- Instead of maintaining optimal value if have t steps left…

1. Calculate exact value of acting in infinite horizon for a particular policy
2. Then try to improve the policy
3. Repeat 1 & 2 until policy doesn't change

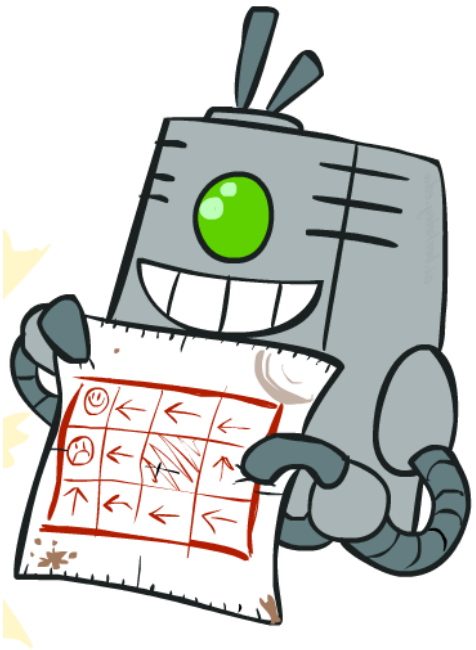


Drawing by Ketrina Yim

Policy Iteration for Infinite Horizon

- Instead of maintaining optimal value if have t steps left…

1. Calculate exact value of acting in infinite horizon for a particular policy **(evaluation)**
2. Then try to improve the policy **(improvement)**
3. Repeat 1 & 2 until policy doesn't change



Drawing by Ketrina Yim

Review: Value of a Policy

- Expected immediate reward for taking action prescribed by policy
- And expected future reward get after taking policy from that state and following π

$$V^{\pi}(s) = \sum_{s' \in S} p(s' \mid s, \pi(s)) \Big[R(s, \pi(s), s') + \gamma V^{\pi}(s') \Big]$$

Policy Evaluation

- Goal: compute $V^{\pi}(s)$ for all s

Idea 1: Use Small Variant of Value Iteration

- Initialize $V_0(s)$ to 0 for all s
- For k=1… convergence

$$V_k^{\pi}(s_i) = \sum_{s_j \in S} p(s_j \mid s_i, \pi(s_i)) \left[R(s, \pi(s), s') + \gamma V_{k-1}^{\pi}(s_j) \right]$$

Idea 2: No max in eqn so linear set of equations… Analytic Solution!

$$V^{\pi}(s) = \sum_{s' \in S} p(s' \mid s, \pi(s)) \left[R(s, \pi(s), s') + \gamma V^{\pi}(s') \right]$$

Let T^{π} be a S x S matrix where the (i,j) entry is:

$$T^{\pi}(s_i, s_j) = p(s_j \mid s_i, \pi(s_i))$$

$$\vec{V} = T^{\pi}\vec{R} + \gamma T^{\pi}\vec{V}$$

$$\vec{V} - \gamma T^{\pi}\vec{V} = T^{\pi}\vec{R}$$

$$\vec{V} = (1 - \gamma T^{\pi})^{-1} T^{\pi}\vec{R}$$

Requires taking an inverse of a S by S matrix $O(S^3)$

Policy Improvement

- Have $V^{\pi}(s)$ for all s
- Want a better policy
- Idea:
  - Find the state-action Q value of doing an action followed by following π forever, for each state
  - Then take argmax of Qs

Policy Improvement

- Have $V^{\pi}(s)$ for all s
- First compute

$$Q^{\pi}(s,a) = \sum_{s' \in S} p(s' \mid s,a)\left[R(s,a,s') + \gamma V^{\pi}(s')\right]$$

- Then extract new policy.
  For each s,

$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$

Delving Deeper Into Improvement Step

$$Q^{\pi}(s,a) = \sum_{s' \in S} p(s' \mid s,a)\left[R(s,a,s') + \gamma V^{\pi}(s')\right]$$

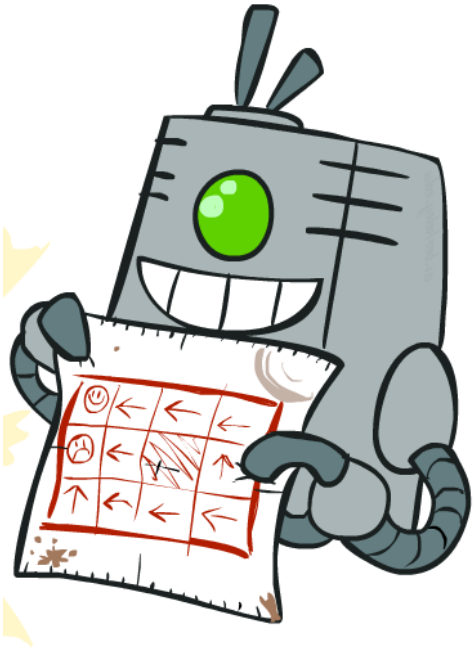
$$\max_a Q^{\pi}(s,a) \geq V^{\pi}(s)$$

$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$

- So if, starting at any state, we followed π'(s), transitioned to a new state s', and then followed π forever, our expected sum of rewards would be at least as good as if we had always followed π
- But we're not doing that, we're getting a new policy and proposing always following that policy π'…

Policy Iteration for Infinite Horizon

1. **Policy Evaluation:**
   Calculate exact value of acting in infinite horizon for a particular policy
2. **Policy Improvement**
3. Repeat 1 & 2 until policy doesn't change



Drawing by Ketrina Yim

Discuss and Report Back: If Policy Doesn't Change (π'(s) =π(s) for all s), Can It Ever Change Again in More Iterations?

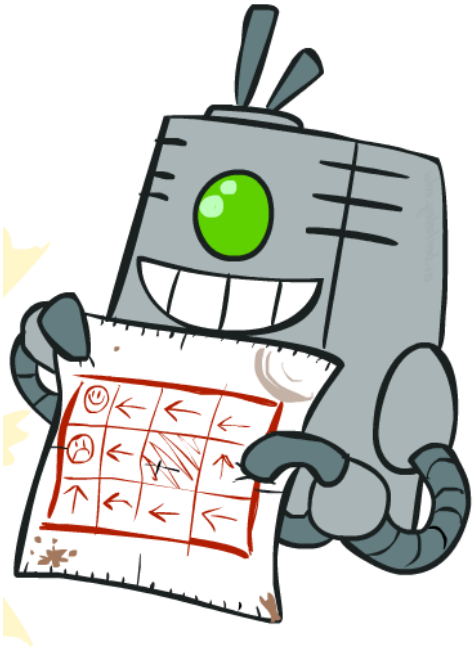
- Given $V^{\pi}(s)$ for all s

$$Q^{\pi}(s,a) = \sum_{s' \in S} p(s' \mid s,a)\left[R(s,a,s') + \gamma V^{\pi}(s')\right]$$

$$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$$

Policy Iteration for Infinite Horizon

1. **Policy Evaluation:**
   Calculate exact value of acting in infinite horizon for a particular policy
2. **Policy Improvement**
3. Repeat 1 & 2 until policy doesn't change



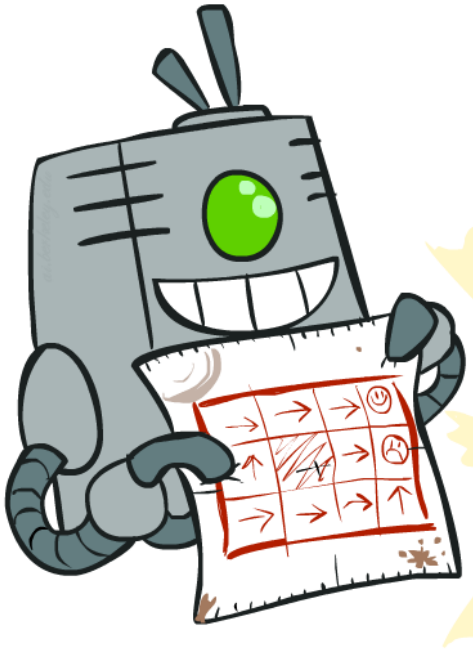
Drawing by Ketrina Yim

- **Discuss and Report Back Policy Iteration is Guaranteed to Converge. Think About Why, and How Many Iterations Could it Take (How many policies could we have to sort through)?**

Policy Iteration

Maintain value of policy

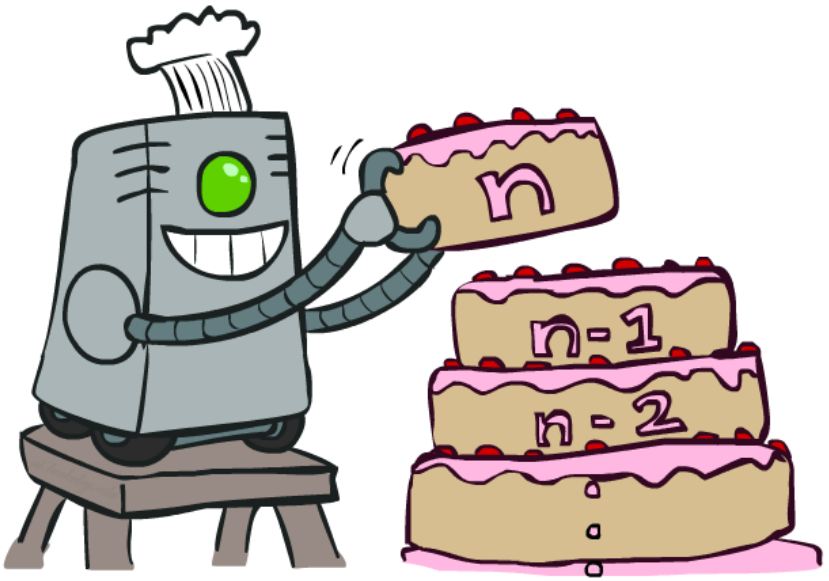
Improve policy



Value Iteration

Keep optimal value for

finite steps, increase steps

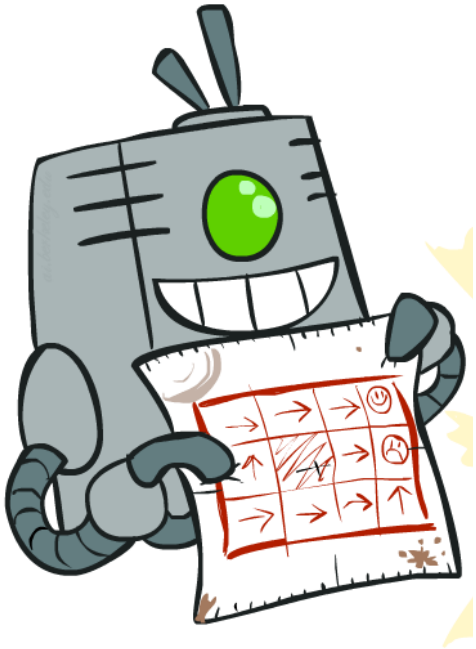


Drawings by Ketrina Yim

Policy Iteration

Fewer Iterations

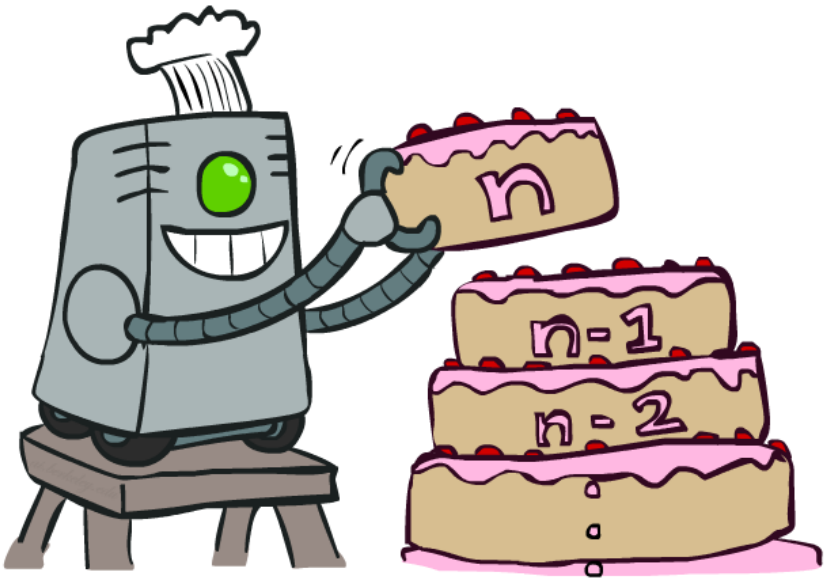
More expensive per iteration



Value Iteration

More iterations

Cheaper per iteration



Drawings by Ketrina Yim

MDPs: What You Should Know

- Definition
- How to define for a problem
- Value iteration and policy iteration
  - How to implement
  - Convergence guarantees
  - Computational complexity