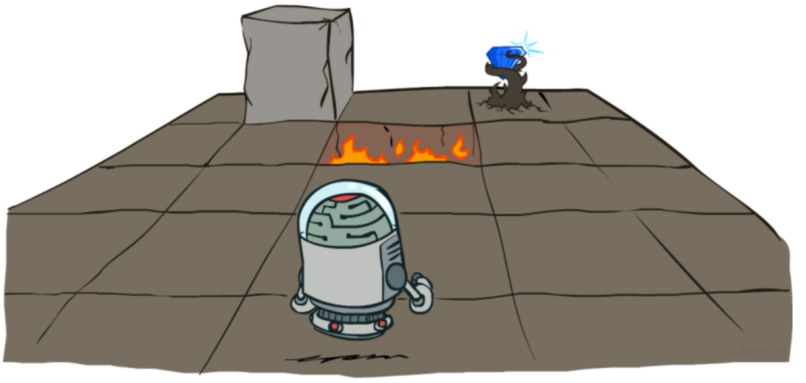
CS 573: Artificial Intelligence

Markov Decision Processes



Dan Weld

University of Washington

Many slides by Dan Klein & Pieter Abbeel / UC Berkeley. (http://ai.berkeley.edu) and by Mausam & Andrey Kolobov

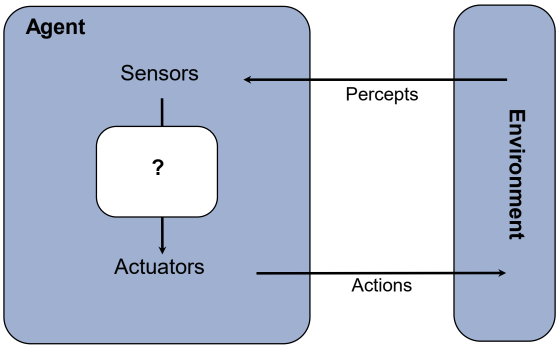
Outline

Stochastic Environments

- Expectimax
- Markov Decision Processes
  - Value iteration
  - Policy iteration
- Reinforcement Learning

Agent *vs.* Environment

- An **agent** is an entity that *perceives* and *acts*.
- A **rational agent** selects actions that ***maximize its utility function***.



Deterministic *vs.* ***stochastic***

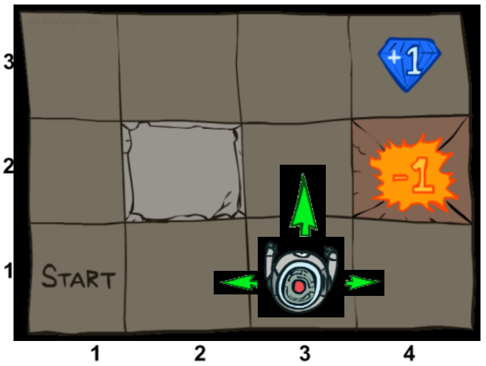
Fully observable *vs.* partially observable

Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function $T(s, a, s')$
    - Probability that a from s leads to s', i.e., $P(s' | s, a)$
    - Also called the model or the dynamics
  - A reward function $R(s, a, s')$

…
$R(s_{32}, N, s_{33}) = -0.01$
…
$R(s_{32}, N, s_{42}) = -1.01$
…
$R(s_{33}, E, s_{43}) = 0.99$

Cost of breathing

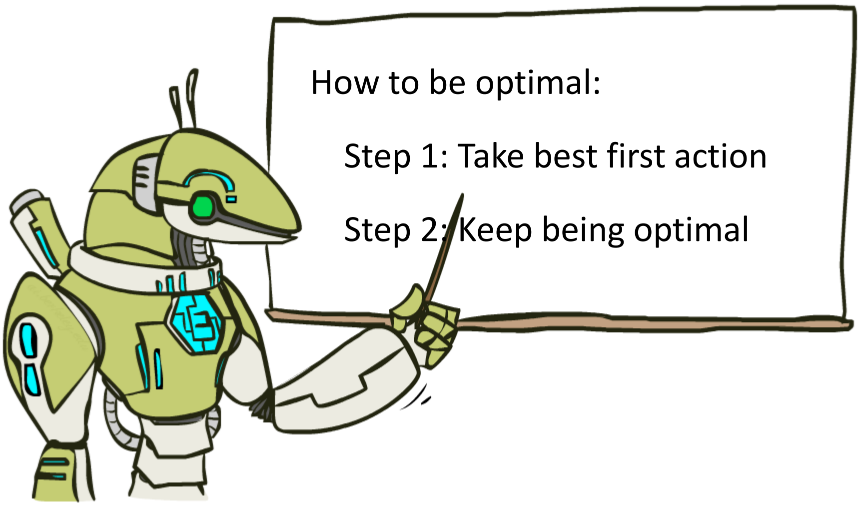


R & T are big tables!

For now, we give them to the agent

- We want to output the optimal policy, $\pi^*: S \rightarrow A$
  - One that maximizes expected value of the discounted sequence of rewards.

The Bellman Equations



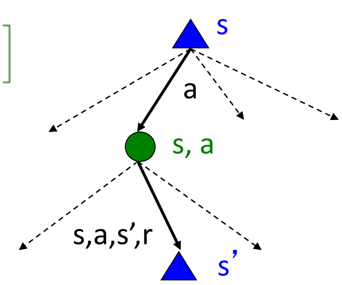
The Bellman Equations


(1920-1984)

Definition of “optimal utility” via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$



These are the Bellman equations, and they characterize optimal values in a way we’ll use over and over

Value Iteration

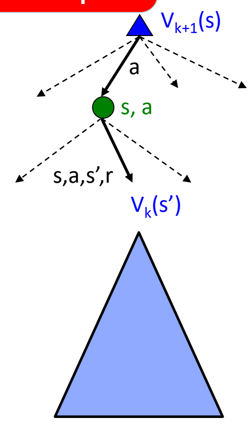
- **Forall s, Initialize $V_0(s) = 0$** *no time steps left means an expected reward of zero*
- **Repeat**

Called a "Bellman Backup"

$K \mathrel{+}= 1$

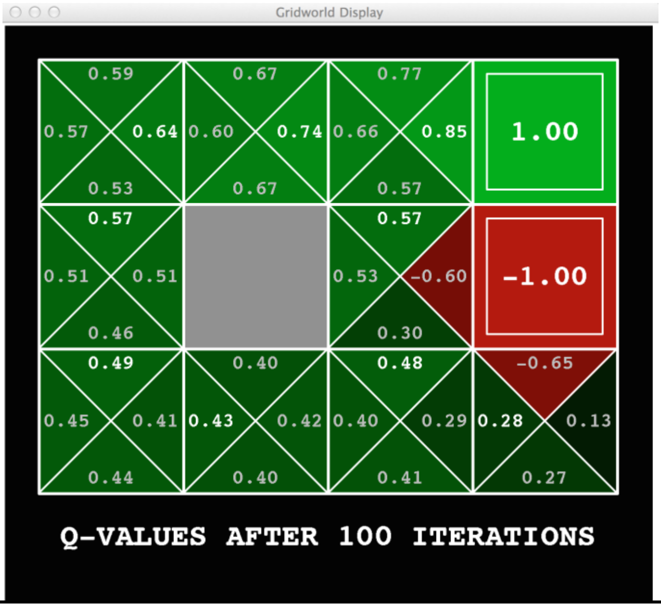
$$\left.\begin{aligned} Q_{k+1}(s, a) &= \Sigma_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')] \\ V_{k+1}(s) &= \text{Max}_a\, Q_{k+1}(s, a) \end{aligned}\right\} \text{do } \forall s, a$$

- **Repeat until $|V_{k+1}(s) - V_k(s)| < \varepsilon$, forall s** *"convergence to V*"*



Successive approximation; dynamic programming

Gridworld: Q*



Gridworld Values V*

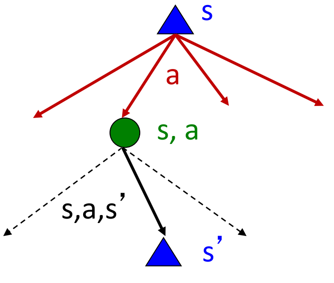
$$V^*(s) = \max_a Q^*(s,a)$$

Problems with Value Iteration

- Value iteration repeats the Bellman updates:

  $Q_{k+1}(s, a) = \Sigma_{s'}\ T(s, a, s')\ [\ R(s, a, s') + \gamma\ V_k(s')]$

  $V_{k+1}(s) = \text{Max}_{\ a}\ Q_{k+1}(s, a)$



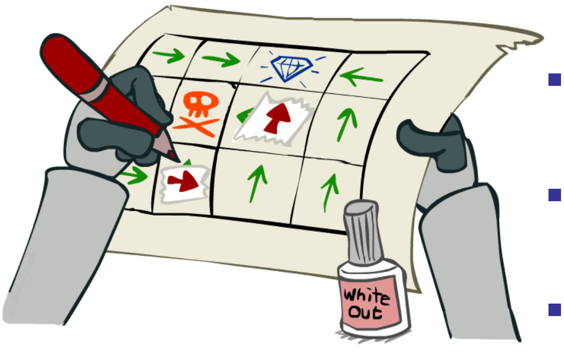
- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "max" at each state rarely changes
- Problem 3: The policy often converges long before the values

VI → Asynchronous VI

- Is it essential to back up ***all*** states in each iteration?
  - No!
- States may be backed up
  - many times or not at all
  - in any order
  - As long as no state gets starved… convergence properties still hold!!
- Random Order …works! Easy to parallelize [Dyna, Sutton 91]
- On-Policy Order Simulate the states that the system actually visits.
- Efficient Order e.g. Prioritized Sweeping [Moore 93]
  Q-Dyna [Peng & Williams 93]

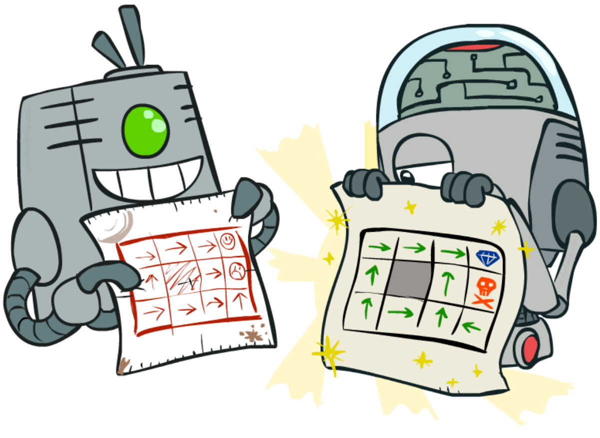
12

Solving MDPs



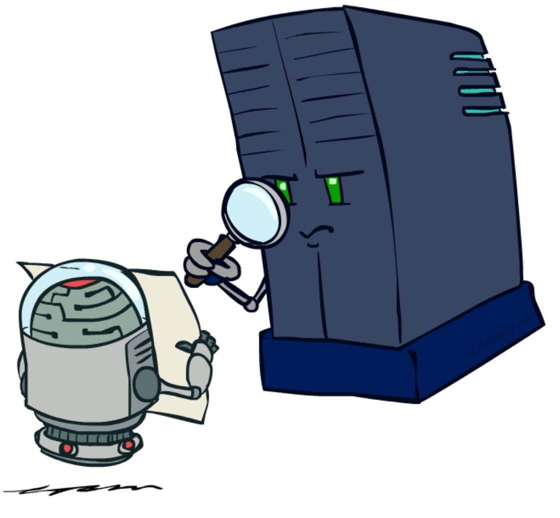
- Value Iteration
- Policy Iteration
- Heuristic Search Methods
- Real-Time Dynamic programming
- Reinforcement Learning

Policy Iteration

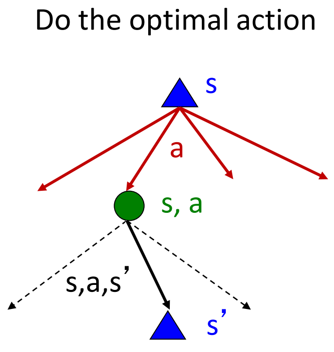


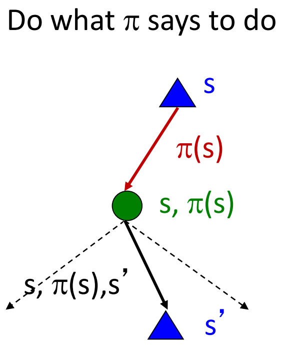
1. Policy Evaluation
2. Policy Improvement

Part 1 - Policy Evaluation



Fixed Policies



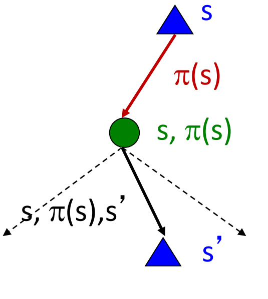


- Expectimax trees ***max*** over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler – only ***one action per state***
  - … though the tree's value would depend on which policy we fixed

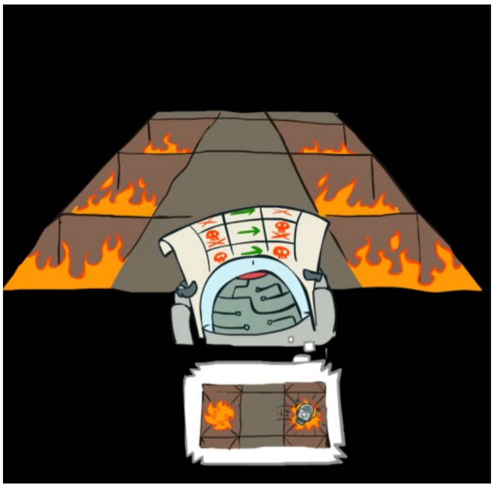
Computing Utilities for a Fixed Policy

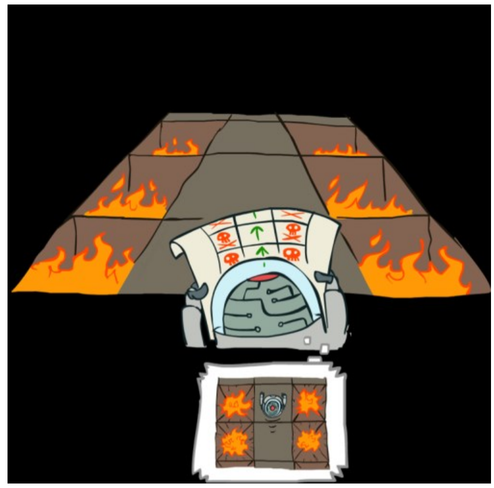
- A new basic operation: compute the utility of a state s under a fixed (generally non-optimal) policy
- Define the utility of a state s, under a fixed policy π:
  $V^{\pi}(s)$ = expected total discounted rewards starting in s and following π
- Recursive relation (variation of Bellman equation):

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$



Example: Policy Evaluation

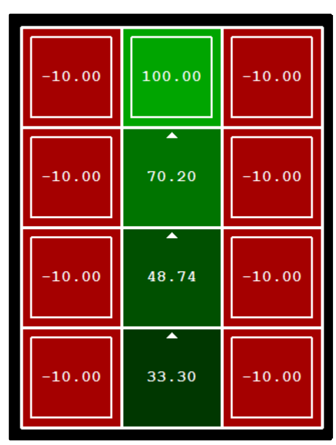




Example: Policy Evaluation

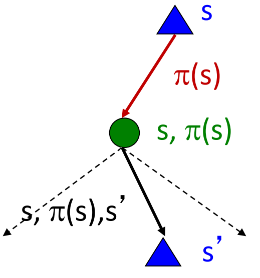
Always Go Right

Always Go Forward



Iterative Policy Evaluation Algorithm

- How do we calculate the V's for a fixed policy π?
- Idea 1: Turn recursive Bellman equations into updates (like value iteration)



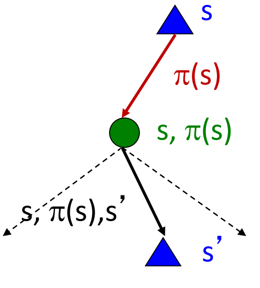
$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$

- Efficiency: $O(S^2)$ per iteration
  - Often converges in much smaller number of iterations compared to VI

Linear Policy Evaluation Algorithm

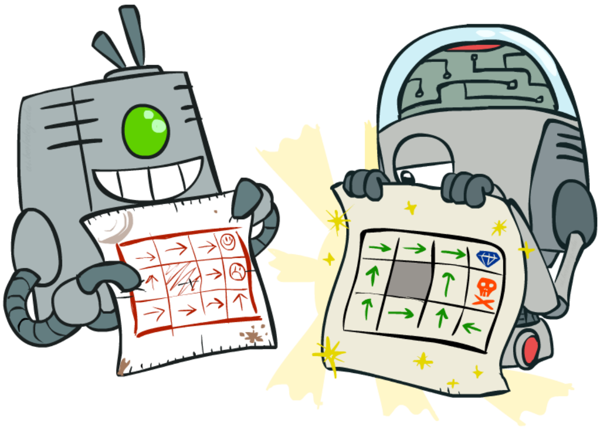
- Another way to calculate the V's for a fixed policy π?
- Idea 2: Without the maxes, the Bellman equations are just a linear system of equations



$$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') \;+\; \gamma V^\pi(s')]$$

- Solve with Matlab (or your favorite linear system solver)
  - S equations, S unknowns = $O(S^3)$ and EXACT!
  - In large spaces, probably too expensive

Policy Iteration



1. Policy Evaluation
2. Policy Improvement

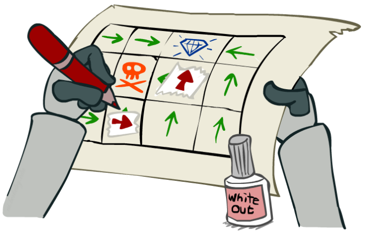
Policy Iteration

- Initialize $\pi(s)$ to random actions
- Repeat
  - Step 1: Policy evaluation: calculate $V^{\pi}(s)$ for each s   *% like we just discussed*
  - Step 2: Policy improvement: update policy using ***one-step look-ahead***

    For each s, what's the ***best action*** to execute, ***assuming agent then follows π?***

    Let $\pi'(s)$ = this best action.

    $\pi = \pi'$
- Until policy doesn't change



Policy Iteration Details

- Initialize $\pi(s)$ to random actions
- Repeat
  - Step 1: Policy evaluation:
    - Initialize k=0; Forall s, $V_0^\pi(s) = 0$
    - Repeat until V^π converges
      - For each state s, $V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')]$
      - Increment k
  - Step 2: Policy improvement:
    - For each state, s, $\pi'(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi}(s')\right]$
    - If $\pi == \pi'$ then it's optimal; return it.
    - Else set $\pi := \pi'$ and loop.

Example

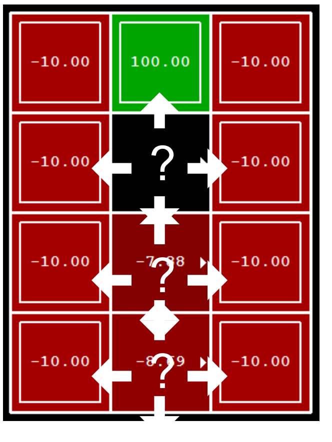
Initialize π_0 to "always go right"

Perform policy evaluation

Perform policy improvement
Iterate through states

Has policy changed?

Yes! i += 1



Example

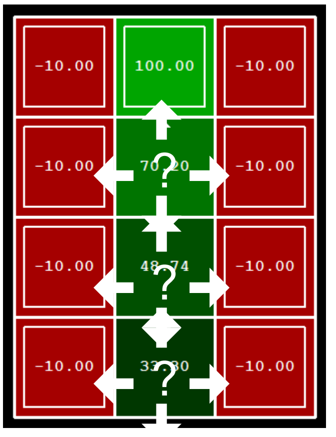
π_1 says "always go up"

Perform policy evaluation

Perform policy improvement
Iterate through states

Has policy changed?

No! We have the optimal policy



Policy Iteration Properties

- Can we view PI as search?
  - Space of …?
  - Search algorithm?
- Policy iteration finds the optimal policy, guaranteed (assuming exact evaluation)!
  - Why does hill-climbing yield optimum?!?
- Often converges (much) faster than VI

VI & PI Comparison

- Changing the search space.

- Policy Iteration
  - Search over the space of possible ***policies***
  - Compute the resulting value

- Value Iteration
  - Search over the space of possible Real-valued ***value functions***
  - Compute the resulting policy

How Big?

41

VI & PI Comparison Part II

- Both compute the same thing (V^* and π^*) using Bellman Equations
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do fewer iterations
  - Each one is slower (must update all V^π and then choose new best π)
  - Modified policy iteration is faster per iteration, since approximate V^π
- Which is better?
  - Lots of actions? Choose Policy Iteration
  - Already got a good policy? Policy Iteration
  - Few actions, acyclic? Value Iteration
  - Best of both worlds – Modified Policy Iteration

Modified Policy Iteration [van Nunen 76]

- initialize π_0 as a random policy
- Repeat

  Approximate Policy Evaluation: Compute $V^{\pi_{n-1}}$

  by running only few iterations of iterative policy eval.

  Policy Improvement: Construct π_n greedy wrt $V^{\pi_{n-1}}$
- Until convergence
- return π_n

43

Modified Policy Iteration as Search

- Can we view MPI as search?
  - Space of … policies
  - Search algorithm… hill climbing.
  - Heuristic?
- In practice, usually faster than VI or PI.

What’s Next Part II? Reinforcement Learning!

- So far we’ve assumed agent knows T(s,a,s’) and R(s,a,s’)
- Often one doesn’t know them, must interact to learn them!
  - PS4 (after midterm) will cover this