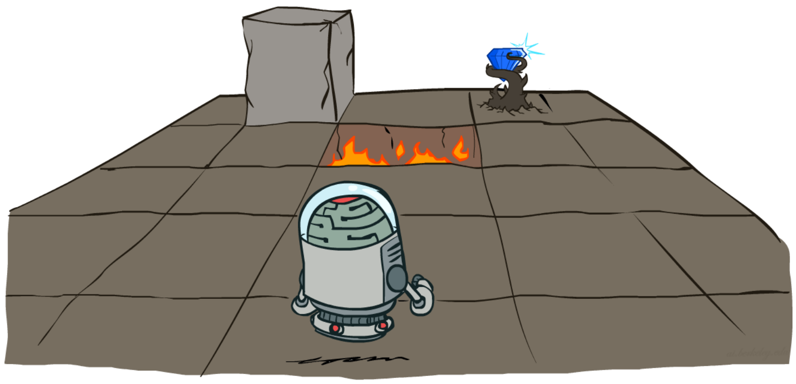
CS 473: Artificial Intelligence

Markov Decision Processes



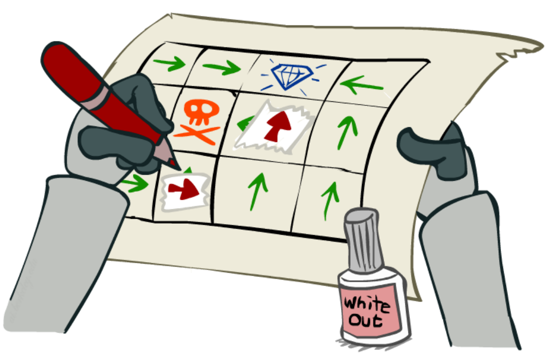
Dan Weld

University of Washington

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

1

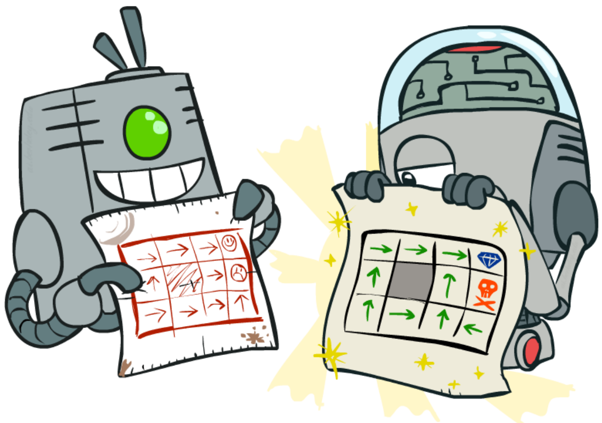
Solving MDPs



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

4

Policy Iteration



1. Policy Evaluation
2. Policy Improvement

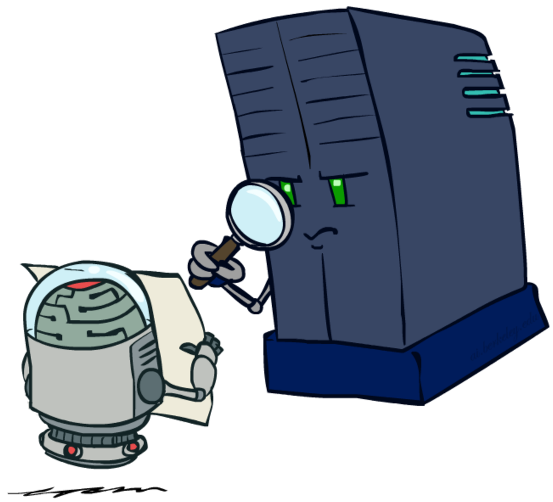
5

Policy Iteration Overview

- Initialize π(s) to random actions
- Repeat
  - Step 1: Policy evaluation: calculate $V^{\pi}(s)$ for each s
  - Step 2: Policy improvement: update policy using ***one-step look-ahead***
- Until policy doesn't change

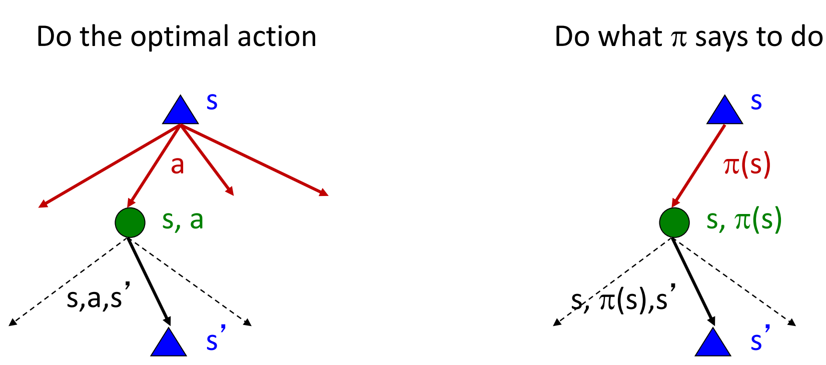
6

Part 1 - Policy Evaluation



7

Fixed Policies

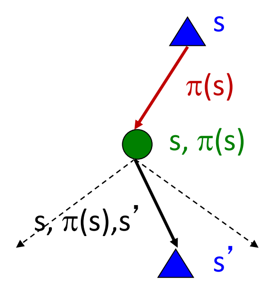


- Expectimax trees ***max*** over all actions to compute the optimal values
- If we fixed some policy $\pi(s)$, then the tree would be simpler – only ***one action per state***
  - … though the tree's value would depend on which policy we fixed

8

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):

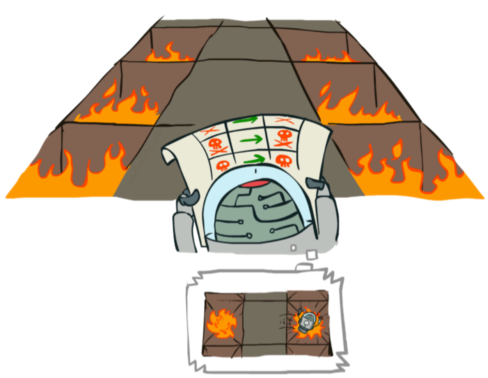


$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$$

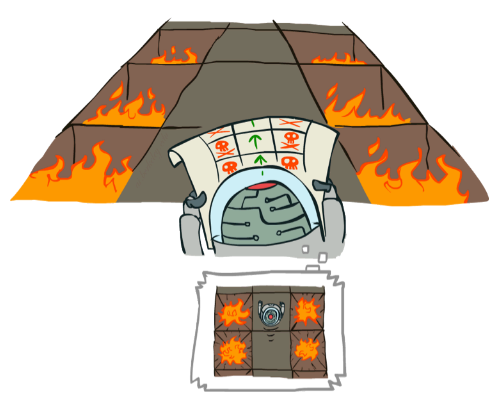
9

Example: Policy Evaluation

Always Go Right



Always Go Forward

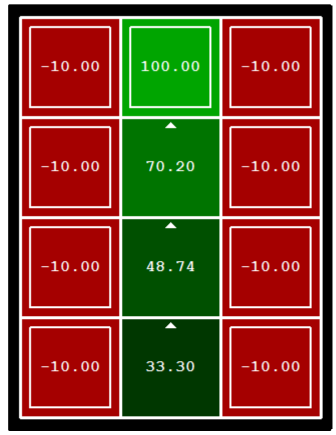


10

Example: Policy Evaluation

Always Go Right

Always Go Forward



11

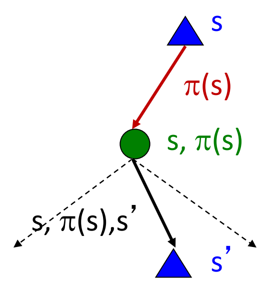
Iterative Policy Evaluation Algorithm

- 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(S²) per iteration
  - Often converges in much smaller number of iterations compared to VI

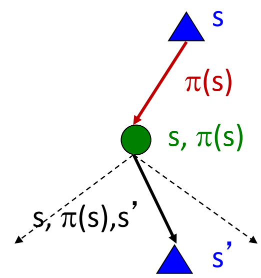
12

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

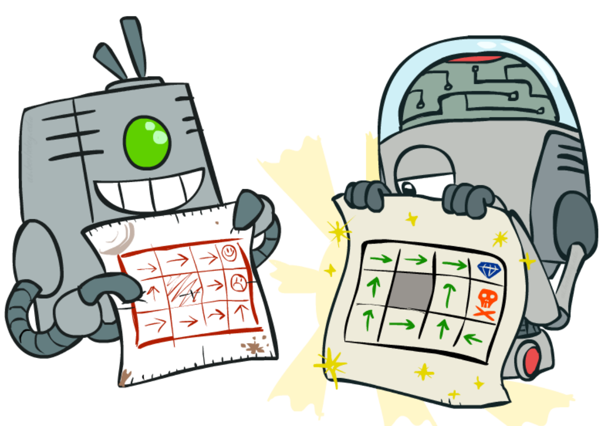
$$V^{\pi}(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$



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

13

Policy Iteration



1. Policy Evaluation
2. Policy Improvement

14

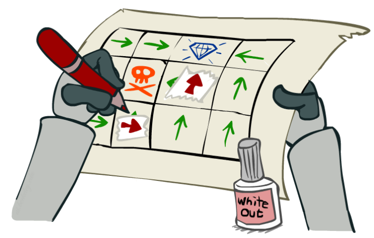
Policy Iteration

- 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



15

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.

16

Example

Initialize π_0 to "always go right"

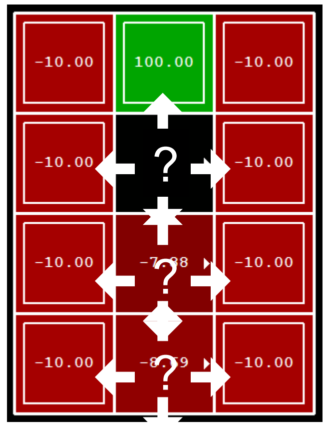
Perform policy evaluation

Perform policy improvement

Iterate through states

Has policy changed?

Yes! i += 1



17

Example

π_1 says "always go up"

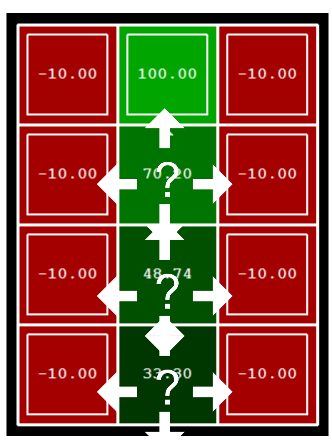
Perform policy evaluation

Perform policy improvement

Iterate through states

Has policy changed?

No! We have the optimal policy



18

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

19

VI & PI Comparison

- Changing the search space.
- Policy Iteration
  - Search over the space of possible ***policies***
  - Hill-climbing search
- Value Iteration
  - Search over the space of possible Real-valued ***value functions***
  - Compute the resulting policy

How Big?

23

23

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

24

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

25

25

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

29