# Markov Decision Processes (MDPs)

Ron Parr
CompSci 370
Department of Computer Science
Duke University

With thanks to Kris Hauser for some slides

# The Winding Path to Reinforcement Learning

- Decision Theory
  - Descriptive theory of optimal behavior
- Markov Decision Processes
  - Mathematical/Algorithmic realization of Decision Theory
- Reinforcement Learning
  - Application of learning techniques to challenges of MDPs with numerous or unknown parameters

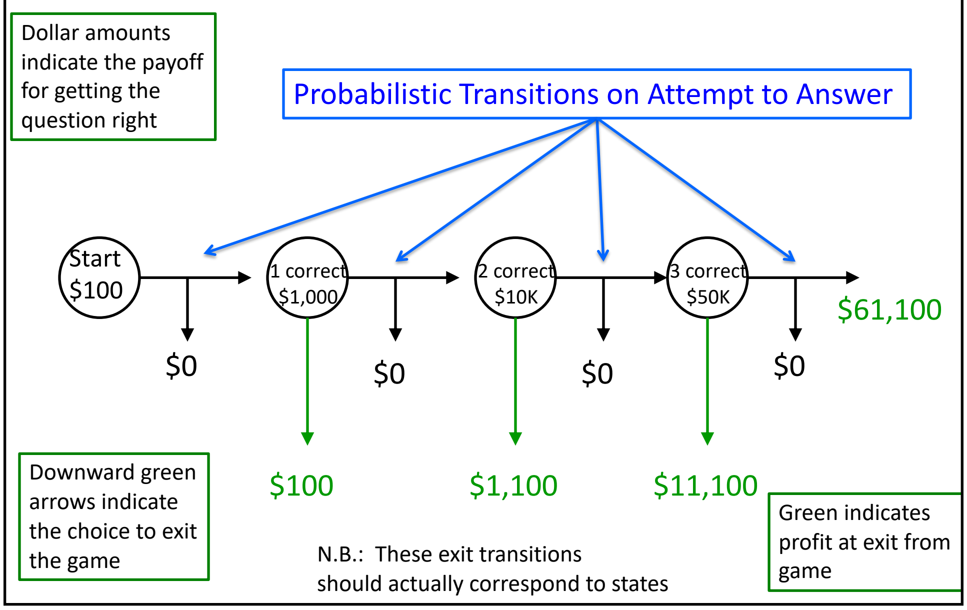
## Swept under the rug today

- Utility of money (assumed 1:1)
- How to determine costs/utilities
- How to determine probabilities

## Playing a Game Show

- Assume series of questions
  - Increasing difficulty
  - Increasing payoff
- Choice:
  - Accept accumulated earnings and quit
  - Continue and risk losing everything
- "Who wants to be a millionaire?"

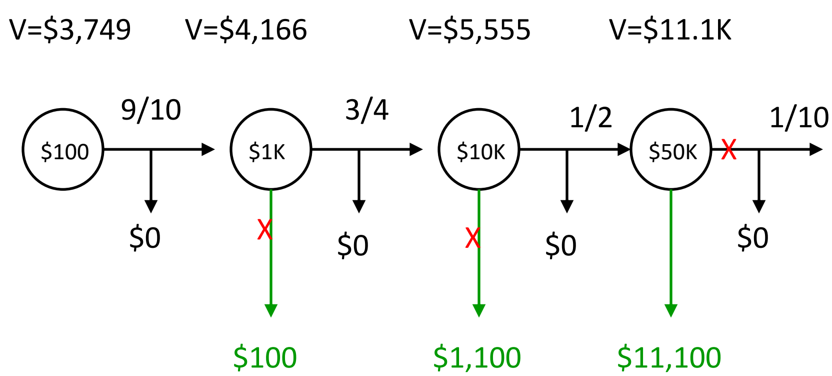
# State Representation




# Making Optimal Decisions

- Work *backwards* from future to present
- Consider $50,000 question
  - Suppose P(correct) = 1/10
  - V(stop)=$11,100
  - V(continue) = 0.9*$0 + 0.1*$61.1K = $6.11K
- Optimal decision stops

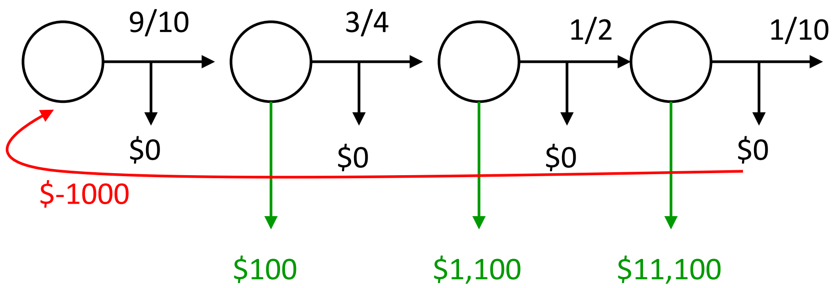
# Working Backwards




Red X indicates bad choice

# Dealing with Loops

Suppose you can pay $1000 (from any losing state) to play again

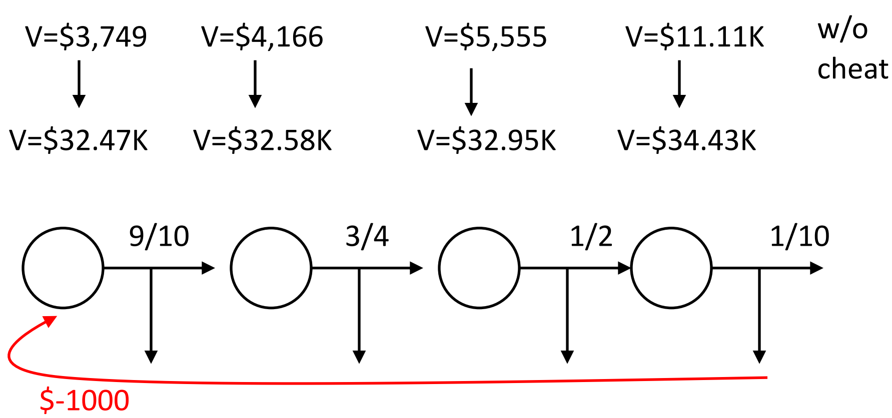


## From Policies to Linear Systems

- Suppose we always pay until we win.
- What is value of following this policy?

$$
\begin{aligned}
V(s_0) &= 0.10(-1000 + V(s_0)) + 0.90V(s_1) \\
V(s_1) &= 0.25(-1000 + V(s_0)) + 0.75V(s_2) \\
V(s_2) &= 0.50(-1000 + V(s_0)) + 0.50V(s_3) \\
V(s_3) &= \underbrace{0.90(-1000 + V(s_0))}_{\text{Return to Start}} + \underbrace{0.10(61100)}_{\text{Continue}}
\end{aligned}
$$

## And the solution is…




Is this optimal?
How do we find the optimal policy?

## The MDP Framework

- State space: S
- Action space: A
- Transition function: P
- Reward function: R(s,a,s') or R(s,a) or R(s)
- Discount factor: $\gamma$
- Policy: $\pi(s) \rightarrow a$

Objective: *Maximize expected, discounted return*
(decision theoretic optimal behavior)

## Applications of MDPs

- AI/Computer Science
  - Robotic control
    (Koenig & Simmons, Thrun et al., Kaelbling et al.)
  - Air Campaign Planning (Meuleau et al.)
  - Elevator Control (Barto & Crites)
  - Computation Scheduling (Zilberstein et al.)
  - Control and Automation (Moore et al.)
  - Spoken dialogue management (Singh et al.)
  - Cellular channel allocation (Singh & Bertsekas)

## Applications of MDPs

- Economics/Operations Research
  - Fleet maintenance (Howard, Rust)
  - Road maintenance (Golabi et al.)
  - Packet Retransmission (Feinberg et al.)
  - Nuclear plant management (Rothwell & Rust)
  - Debt collection strategies (Abe et al.)
  - Data center management (DeepMind)

## Applications of MDPs

- EE/Control
  - Missile defense (Bertsekas et al.)
  - Inventory management (Van Roy et al.)
  - Football play selection (Patek & Bertsekas)
- Agriculture
  - Herd management (Kristensen, Toft)
- Other
  - Sports strategies
  - Board games
  - Video games

## The Markov Assumption

- Let $S_t$ be a random variable for the state at time t
- $P(S_t|A_{t-1}S_{t-1},\ldots,A_0S_0) = P(S_t|A_{t-1}S_{t-1})$
- Markov is special kind of conditional independence
- Future is independent of past given current state, ***action***

## Understanding Discounting

- Mathematical motivation
  - Keeps values bounded
  - What if I promise you $0.01 every day you visit me?
- Economic motivation
  - Discount comes from inflation
  - Promise of $1.00 in future is worth $0.99 today
- Probability of dying (losing the game)
  - Suppose $\varepsilon$ probability of dying at each decision interval
  - Transition w/prob $\varepsilon$ to state with value 0
  - Equivalent to 1- $\varepsilon$ discount factor

## Discounting in Practice

- Often chosen unrealistically low
  - Faster convergence of the algorithms we'll see later
  - Leads to slightly myopic policies

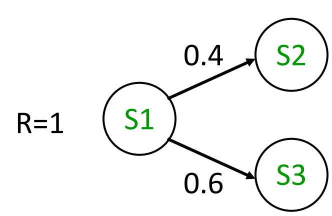
- Can reformulate most algs. for avg. reward
  - Mathematically uglier
  - Somewhat slower run time

## Value Determination

Determine the value of each state under policy π

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

Bellman Equation for a fixed policy π




$$V^{\pi}(s_1) = 1 + \gamma(0.4 V^{\pi}(s_2) + 0.6 V^{\pi}(s_3))$$

## Matrix Form

$$\mathbf{P}^{\pi} = \begin{pmatrix} P(s_1 \mid s_1, \pi(s_1)) & P(s_2 \mid s_1, \pi(s_1)) & P(s_3 \mid s_1, \pi(s_1)) \\ P(s_1 \mid s_2, \pi(s_2)) & P(s_2 \mid s_2, \pi(s_2)) & P(s_3 \mid s_2, \pi(s_2)) \\ P(s_1 \mid s_3, \pi(s_3)) & P(s_2 \mid s_3, \pi(s_3)) & P(s_3 \mid s_3, \pi(s_3)) \end{pmatrix}$$

$$\mathbf{V}^{\pi} = \gamma \mathbf{P}^{\pi} \mathbf{V}^{\pi} + \mathbf{R}^{\pi}$$

This is a generalization of the game show example from earlier

How do we solve this system efficient? Does it even have a solution?

## Solving for Values

$$\mathbf{V}^{\pi} = \gamma \mathbf{P}^{\pi} \mathbf{V}^{\pi} + \mathbf{R}^{\pi}$$

For moderate numbers of states we can solve this system exacty:

$$\mathbf{V}^{\pi} = \underbrace{(\mathbf{I} - \gamma \mathbf{P}^{\pi})}_{}{}^{-1} \mathbf{R}^{\pi}$$

Guaranteed invertible because $\gamma \mathbf{P}^{\pi}$ has spectral radius <1

# Iteratively Solving for Values

$$\mathbf{V}^{\pi} = \gamma \mathbf{P}^{\pi} \mathbf{V}^{\pi} + \mathbf{R}^{\pi}$$

For larger numbers of states we can solve this system indirectly:

$$\mathbf{V}^{\pi}{}_{i+1} = \gamma \mathbf{P}^{\pi} \mathbf{V}^{\pi}{}_{i} + \mathbf{R}^{\pi}$$

Guaranteed convergent because $\gamma \mathbf{P}_{\pi}$ has spectral radius <1

# Establishing Convergence

- Eigenvalue analysis
- Monotonicity
  - Assume all values start pessimistic
  - One value must always increase
  - Can never overestimate
  - Easy to prove
- Contraction analysis…

# Contraction Analysis

- Define maximum norm

$$\|V\|_\infty = \max_i |V[i]|$$

- Consider two value functions $V^a$ and $V^b$ each at iteration 1:

$$\left\|V_1^a - V_1^b\right\|_\infty = \varepsilon$$

- WLOG say

$$V_1^a \leq V_1^b + \vec{\varepsilon} \quad \text{(Vector of all } \varepsilon\text{'s)}$$

# Contraction Analysis Contd.

- At next iteration for $V^b$:

$$V_2^b = R + \gamma P V_1^b$$

- For $V^a$

$$V_2^a = R + \gamma P(V_1^a) \leq R + \gamma P\underbrace{(V_1^b + \vec{\varepsilon})}_{\text{Distribute}} = R + \gamma P V_1^b + \gamma P\vec{\varepsilon} = R + \gamma P V_1^b + \gamma\vec{\varepsilon}$$

- Conclude:

$$\left\|V_2^a - V_2^b\right\|_\infty \leq \gamma\varepsilon$$

# Importance of Contraction

- Any two value functions get closer
- True value function V* is a fixed point
(value doesn't change with iteration)
- Max norm distance from V* decreases *dramatically* quickly with iterations

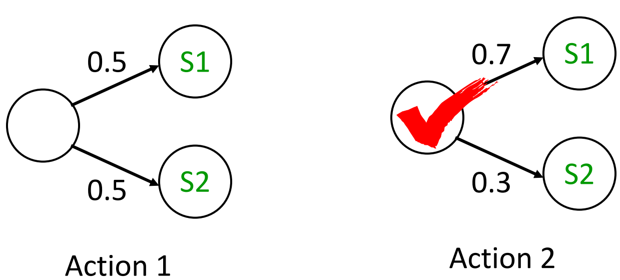
$$\left\| V_0 - V^* \right\|_\infty = \varepsilon \rightarrow \left\| V_n - V^* \right\|_\infty \le \gamma^n \varepsilon$$

# Finding Good Policies

Suppose an expert told you the "true value" of each state:

V(S1) = 10

V(S2) = 5



## Improving Policies

- How do we get the optimal policy?
- If we knew the values under the optimal policy, then just take the optimal action in every state
- How do we define these values?
- Fixed point equation with choices (Bellman equation):

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

Decision theoretic optimal choice given V*
If we know V*, picking the optimal action is easy
If we know the optimal actions, computing V* is easy
How do we compute both at the same time?

## Value Iteration

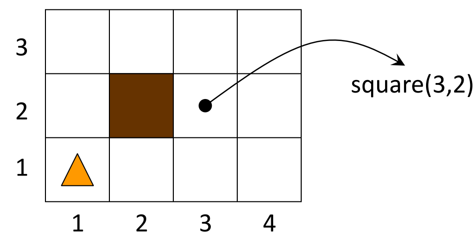
We can't solve the system directly with a max in the equation
Can we solve it by iteration?

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

- Called *value iteration* or simply *successive approximation*
- Same as value determination, but we can *change* actions

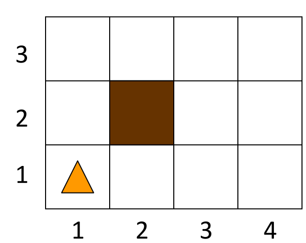
- Convergence:
  - Can't do eigenvalue analysis (not linear)
  - Still monotonic
  - Still a contraction in max norm (exercise)
  - Converges quickly

# Robot Navigation Example




- The robot (shown) lives in a world described by a 4x3 grid of squares with square (2,2) occupied by an obstacle
- A state is defined by the square in which the robot is located: (1,1) in the above figure
  → 11 states

# Action (Transition) Model

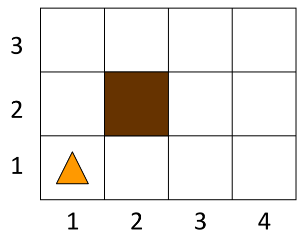



U brings the robot to:
- (1,2) with probability 0.8
- (2,1) with probability 0.1
- (1,1) with probability 0.1

- In each state, the robot's possible actions are {U, D, R, L}
- For each action:
  - With probability 0.8 the robot does the right thing (moves up, down, right, or left by one square)
  - With probability 0.1 it moves in a direction perpendicular to the intended one
  - If the robot can't move, it stays in the same square

  [This model satisfies the Markov condition]

# Action (Transition) Model




L brings the robot to:
- (1,1) with probability 0.8 + 0.1 = 0.9
- (1,2) with probability 0.1

- In each state, the robot's possible actions are {U, D, R, L}
- For each action:
  - With probability 0.8 the robot does the right thing (moves up, down, right, or left by one square)
  - With probability 0.1 it moves in a direction perpendicular to the intended one
  - If the robot can't move, it stays in the same square

  [This model satisfies the Markov condition]

# Terminal States, Rewards, and Costs

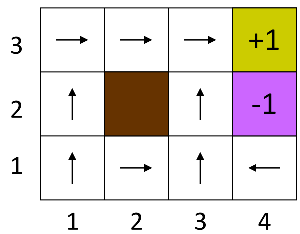
| | | | | |
|---|---|---|---|---|
| 3 | -.04 | -.04 | -.04 | +1 |
| 2 | -.04 | | -.04 | -1 |
| 1 | -.04 | -.04 | -.04 | -.04 |
| | 1 | 2 | 3 | 4 |

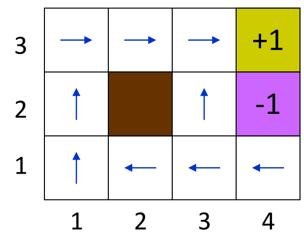
"terminal" states
Not part of formal MDP specification. Usually handled by forcing state to have a fixed value, e.g. +1

- Two terminal states: (4,2) and (4,3)
- Rewards:
  - R(4,3) = +1 [The robot finds gold]
  - R(4,2) = -1 [The robot gets trapped in quicksand]
  - R(s) = -0.04 in all other states

- This example (from the Russell & Norvig text) assumes no discounting ($\gamma$=1)
- Discussion: Is this a good modeling decision?

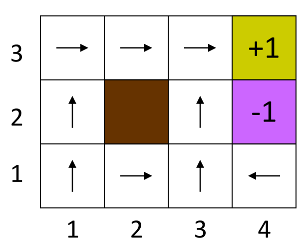
# (Stationary) Policy

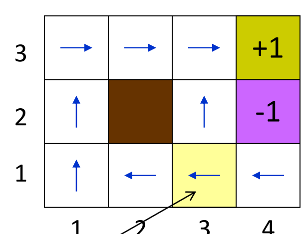



- A stationary policy is a complete map $\pi$: state $\rightarrow$ action
- For each non-terminal state it recommends an action, independent of when and how the state is reached
- Under the Markov and infinite horizon assumptions, the optimal policy $\pi^*$ is necessarily a stationary policy
  [The best action in a state does not depends on the past]

# (Stationary) Policy



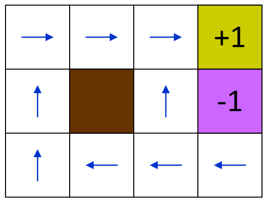




- A stationary policy is a complete map $\pi$: state $\rightarrow$ action
- For each non-terminal state it recommends an action, independent of when and how the

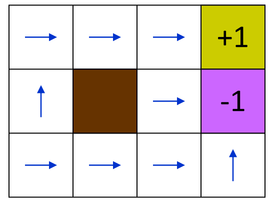
The optimal policy tries to avoid “dangerous” state (3,2)

- Under the M nal policy $\pi^*$ is necessarily a stationary policy
  [The best action in a state does not depends on the past]

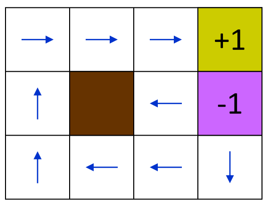
# Optimal Policies for Various R(s)



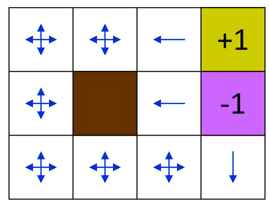

R(s) = -0.04




R(s) = -2

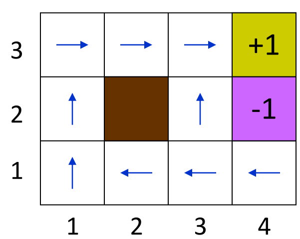



R(s) = -0.01




R(s) > 0

# Bellman Equation




The utility of s depends on the utility of other states s' (possibly, including s), and vice versa

- If s is terminal:

$$V(s) \;=\; R(s)$$

Appl(s) used if not all actions are defined in all states

- If s is non-terminal:

$$V(s) = R(s) + \max_{a \in Appl(s)} \sum_{s' \in Succ(s,a)} P(s'|s,a)V(s')$$

[Bellman equation]

- $\pi^*(s) = \arg\max_{a \in Appl(s)} \sum_{s' \in Succ(s,a)} P(s'|s,a)V(s')$

# Value Iteration Applied

| | | | | |
|---|---|---|---|---|
| 3 | 0 | 0 | 0 | +1 |
| 2 | 0 | | 0 | -1 |
| 1 | 0 | 0 | 0 | 0 |
| | 1 | 2 | 3 | 4 |

$\longrightarrow$

| | | | | |
|---|---|---|---|---|
| 3 | 0.81 | 0.87 | 0.92 | +1 |
| 2 | 0.76 | | 0.66 | -1 |
| 1 | 0.71 | 0.66 | 0.61 | 0.39 |
| | 1 | 2 | 3 | 4 |

1. Initialize the utility of each non-terminal states to $V_0(s) = 0$
2. For t = 0, 1, 2, ... do

$$V_{t+1}(s) = R(s) + \max_{a \in Appl(s)} \sum_{s' \in Succ(s,a)} P(s'|s,a)V_t(s')$$

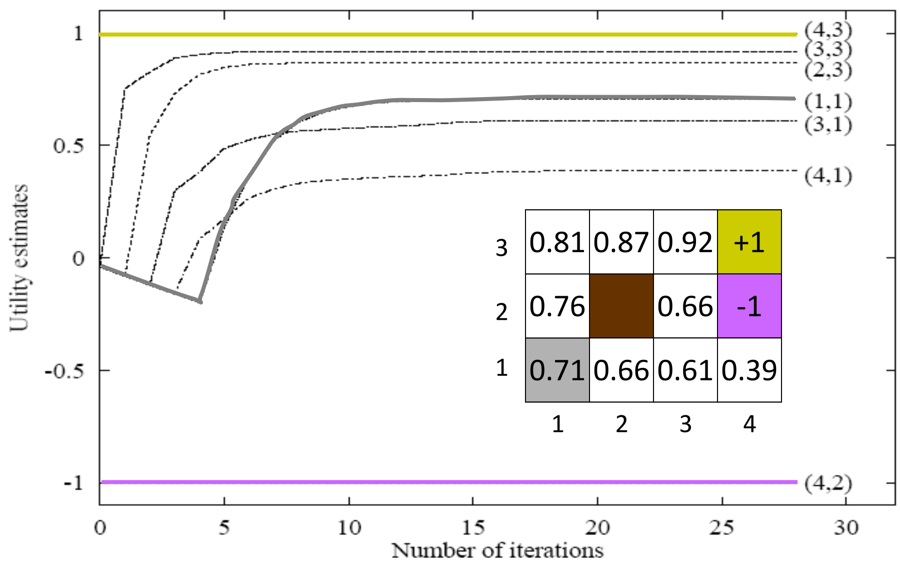
for each non-terminal state s

# State Utilities/Values

| | | | | |
|---|---|---|---|---|
| 3 | 0.81 | 0.87 | 0.92 | +1 |
| 2 | 0.76 | | 0.66 | -1 |
| 1 | 0.71 | 0.66 | 0.61 | 0.39 |
| | 1 | 2 | 3 | 4 |

- The utility of a state s is the maximal expected amount of reward that the robot will collect from s and future states by executing some action in each encountered state, until it reaches a terminal state (infinite horizon)
- Under the Markov and infinite horizon assumptions, the utility of s is independent of when and how s is reached
  [It only depends on the possible sequences of states after s, not on the possible sequences before s]

# Convergence of Value Iteration




# Properties of Value Iteration

- VI converges to V* ($||.||_\infty$ from V* shrinks by $\gamma$ factor each iteration)
- Converges to optimal policy
- Why? (Because we figure out V*, optimal policy is argmax)
- Optimal policy is stationary (i.e. Markovian – depends only on current state)
- Why? (Because we are summing utilities. Thought experiment: Suppose you think it's better to change actions the second time you visit a state. Why didn't you just take the best action the first time?)

## Policy Iteration

## Greedy Policy Construction

Let's name the action that looks best WRT V:

$$\pi_v(s) = \arg\max_a R(s,a) + \gamma \sum_{s'} \underbrace{P(s' \mid s,a)V(s')}_{\text{Expectation over next-state values}}$$

$$\pi_v = \text{greedy}(V)$$

# Bootstrapping: Policy Iteration

Idea: Greedy selection is useful even with suboptimal V

Guess $\pi_V = \pi_0$

$V_\pi$ = value of acting on $\pi$

(solve linear system)

$\pi_V \leftarrow \text{greedy}(V_\pi)$

Repeat until policy doesn't change

Guaranteed to find optimal policy

Usually takes very small number of iterations

Computing the value functions is the expensive part

# Comparing VI and PI

- VI
  - Value changes at every step
  - Policy *may* change before exact value of policy is computed
  - Many relatively cheap iterations
- PI
  - Alternates policy/value updates
  - Solves for value of each policy *exactly*
  - Fewer, slower iterations (need to invert matrix)
- Convergence
  - Both are contractions in max norm
  - PI is *shockingly* fast (small number of iterations) in practice

## Computational Complexity

- VI and PI are both contraction mappings w/rate $\gamma$
  (we didn't prove this for PI in class)
- VI costs less per iteration
- For n states, a actions PI tends to take O(n) iterations in practice
  - Recent results indicate ~$O(n^2a/1-\gamma)$ worst case
  - Interesting aside: Biggest insight into PI came ~50 years after the algorithm was introduced

## A Unified View of Value Iteration and Policy Iteration

# Notation

- Update for for a fixed policy – definition of $T^{\pi}$ operator (matrix-vector form):

$$T^{\pi}V \equiv R_{\pi} + \gamma P^{\pi}V$$

- Update with policy improvement – definition of the T operator:

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

# Value Determination

- For 0 steps $V_0 = R^{\pi}$
- For i steps $V_i = T^{\pi}V_{i-1} = (T^{\pi})^i R^{\pi}$
- Infinite horizon $\lim_{i\to\infty} V_i = (T^{\pi})^{\infty}R^{\pi} = (1-\gamma P^{\pi})^{-1}R^{\pi} = V^{\pi}$

## Value Iteration

- For 0 steps $V_0 = R$ (If R depends on a, pick a with the highest immediate reward)

- For i steps $V_i = TV_{i-1} = T^i R$

- Infinite horizon $\lim_{i\to\infty} V_i = T^{\infty} R = TV^* = V^*$

## Modified Policy Iteration

- Guess $V_0$ (usually just R), and $\pi$
- i=1
- Repeat until convergence*
  - For j=1 to n
    - $V_i = T^{\pi} V_{i-1}$
    - i = i+1

    n steps of iterative policy evaluation
  - $\pi$ =greedy($V_{i-1}$)

- Special cases: n=1 (VI), n⟶∞ (PI)

# MDP Limitations → Reinforcement Learning

- MDP operate at the level of *states*
  - States = atomic events
  - We usually have exponentially (or infinitely) many of these
- We assume P and R are known

- Machine learning to the rescue!
  - Infer P and R (implicitly or explicitly from data)
  - Generalize from small number of states/policies