Markov Decision Processes

Infinite Horizon Problems

Alan Fern *

* Based in part on slides by Craig Boutilier and Daniel Weld

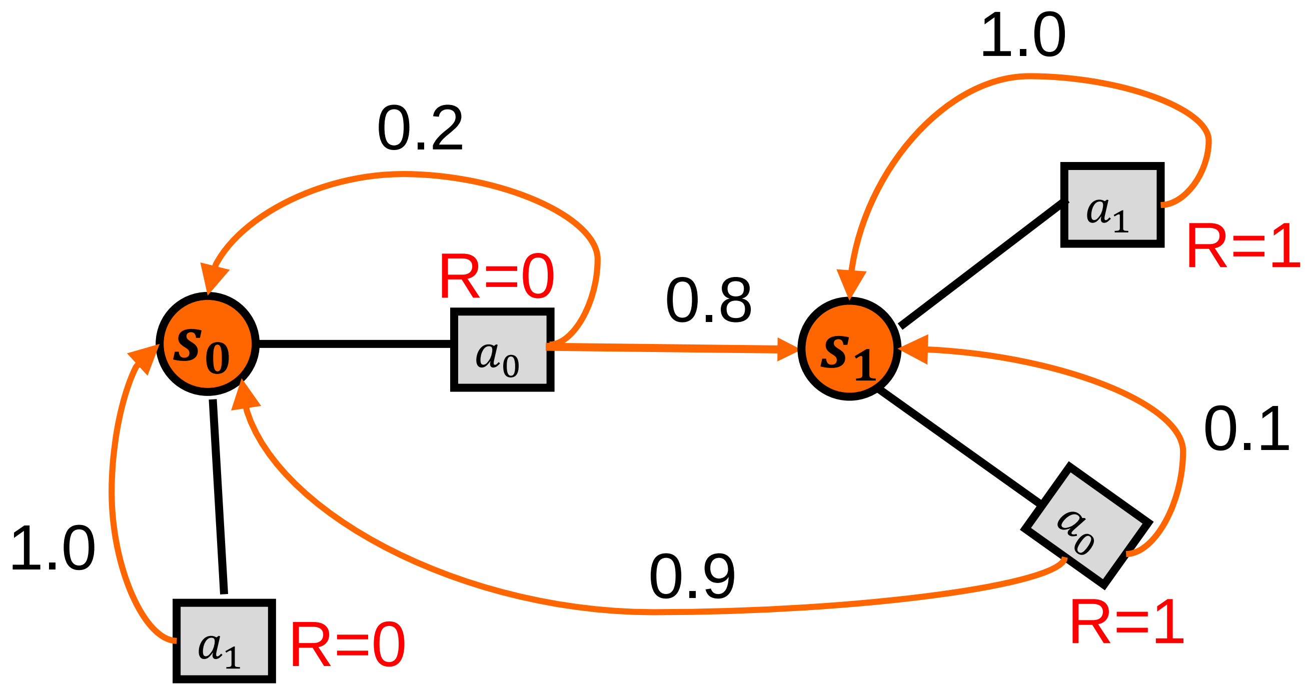
What is a solution to an MDP?

MDP Planning Problem:

Input: an MDP (S,A,R,T)
Output: a policy that achieves an "optimal value"

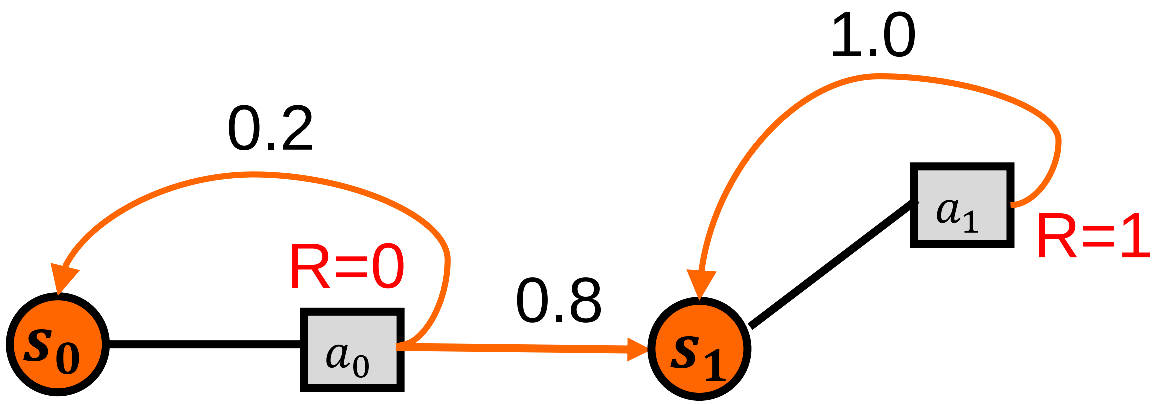
- This depends on how we define the value of a policy
- There are several choices and the solution algorithms depend on the choice
- We will consider two common choices
  - Finite-Horizon Value
  - **Infinite Horizon Discounted Value**

Infinite Horizons



Consider accumulating reward over an infinite horizon?

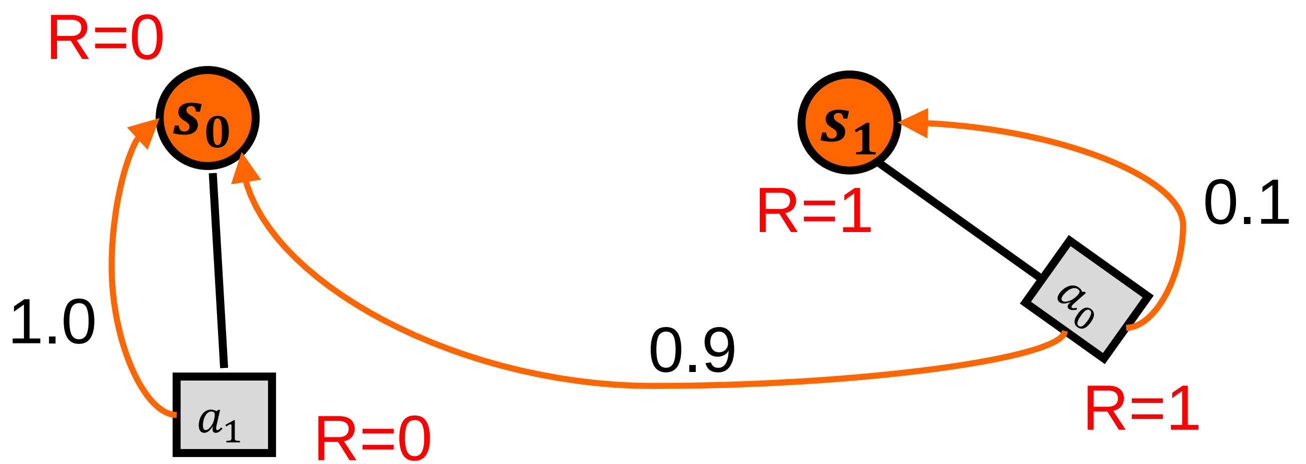
Infinite Horizon



Example policy: $\pi(s_0) = a_0,\ \pi(s_1) = a_1$

Do we have any problems here for infinite horizon?

Infinite Horizon



Example policy: $\pi(s_0) = a_1,\ \pi(s_1) = a_0$

Do we have any problems here for infinite horizon?

Discounted Infinite Horizon MDPs

- Defining value as total reward is problematic with infinite horizons $r_0 + r_1 + r_2 + r_3 + \cdots$
  - many or all policies have infinite expected reward
  - some MDPs are ok (e.g., zero-cost absorbing states)
- Why is this bad?
  - Consider π_1 that gets R=1 per step and π_2 that gets R=2 per step
  - π_2 is clearly better, but infinite total reward can't distinguish between them (both get infinite value)
- "Trick": introduce discount factor $0 \leq \beta < 1$
  - future rewards discounted by β per time step
  - $r_0 + \beta r_1 + \beta^2 r_2 + \beta^3 r^3 + \cdots$

Discounted Infinite Horizon MDPs

- Expected infinite horizon discounted reward

$$V_\pi(s) = E\left[\sum_{t=0}^{\infty} \beta^t R^t \mid \pi, s\right]$$

- We avoid infinite values (consider getting max absolute reward each step)

Maximum absolute reward

$$V_\pi(s) \le E\left[\sum_{t=0}^{\infty} \beta^t R^{\max}\right] = \frac{1}{1-\beta} R^{\max}$$

- Motivation: economic? prob of death? convenience?

Notes: Discounted Infinite Horizon

- Optimal policies guaranteed to exist (Howard, 1960)
  - I.e. there is a policy that maximizes value at each state
- Furthermore there is always an optimal stationary policy
  - Intuition: why would we change action at s at a new time when there is always forever ahead
- We define $V^*(s)$ to be the optimal value function.
  - That is, $V^*(s) = V_\pi(s)$ for some optimal stationary π

Computational Problems

- Policy Evaluation
  - Given π and an MDP compute V_π

- Policy Optimization
  - Given an MDP, compute an optimal policy π^* and V^*.
  - We'll cover two algorithms for doing this: value iteration and policy iteration

Policy Evaluation

- Value equation for fixed policy

$$\underbrace{V_\pi(s) = R(s,\pi(s))}_{\text{immediate reward}} + \underbrace{\beta \sum_{s'} T(s,\pi(s),s') \cdot V_\pi(s')}_{\text{discounted expected value of following policy in the future}}$$

- Equation can be derived from original definition of infinite horizon discounted value

Sutton & Barto Notation vs. Ours

- Recall that Sutton & Barto define MDPs via $\boldsymbol{p(s', r \mid s, a)}$ rather than **R(s,a)** and **T(s,a,s')** as in our slides
- Define $\boldsymbol{R(s,a) = \sum_{s'} \sum_{r} p(s', r \mid s, a) \cdot r = E_p[r \mid s, a]}$ and

  $$\boldsymbol{T(s,a,s') = \sum_{r} p(s', r \mid s, a) = p(s' \mid s, a)}$$

- By defining R and T this way, our value function defined via R and T is equivalent to definition via Sutton & Barto's p

$$V_\pi(s) = \sum_{s'} \sum_{r} p(s', r \mid s, \pi(s)) \cdot (r + \beta V_\pi(s'))$$ **;; definition via p**

$$= \sum_{s'} \sum_{r} p(s', r \mid s, \pi(s)) \cdot r + \sum_{s'} \sum_{r} p(s', r \mid s, \pi(s)) \cdot \beta V_\pi(s'))$$

$$= R(s, \pi(s)) + \beta \sum_{s'} T(s, \pi(s), s') \cdot V_\pi(s')$$ **;; definition via R and T**

Policy Evaluation

- Value equation for fixed policy

$$V_\pi(s) = R(s,\pi(s)) + \underbrace{\beta \sum_{s'} T(s,\pi(s),s') \cdot V_\pi(s')}_{\text{discounted expected value of following policy in the future}}$$

immediate reward

- Equation can be derived from original definition of infinite horizon discounted value

Policy Evaluation

- Value equation for fixed policy

$$V_{\pi}(s) = R(s, \pi(s)) + \beta \sum_{s'} T(s, \pi(s), s') \cdot V_{\pi}(s')$$

- How can we compute the value function for a fixed policy?
  - we are given R, T, π, β and want to find $V_{\pi}(s)$ for each s
  - linear system with n variables and n constraints
    - Variables are values of states: $V(s_1), \ldots, V(s_n)$
    - Constraints: one value equation (above) per state
  - Use linear algebra to solve for V (e.g. matrix inverse)

Policy Evaluation via Matrix Inverse

$\boldsymbol{S} = \{\boldsymbol{s_1}, \boldsymbol{s_2}, \dots, \boldsymbol{s_n}\}$

$\boldsymbol{V_\pi}$ is n-dim column vector, where $V_\pi(i) = \mathrm{V}_\pi(\mathrm{s_i})$

R is n-dim column vector, where $R(i) = \mathrm{R}(\mathrm{s_i}, \pi(s_i))$

T is an nxn matrix s.t. $T(i,j) = T(s_i, \pi(s_i), s_j)$

$$V_\pi = R + \beta \mathrm{T} V_\pi$$

$$\Downarrow$$

$$(I - \beta \mathrm{T}) V_\pi = R$$

$$\Downarrow$$

$$\boxed{V_\pi = (I - \beta \mathrm{T})^{-1} R}$$

Computational Problems

- Policy Evaluation
  - Given π and an MDP compute V_π
- Policy Optimization
  - Given an MDP, compute an optimal policy π^* and V^*.
  - We'll cover two algorithms for doing this: value iteration and policy iteration

Optimizing Value Functions

- Our first algorithm will compute an arbitrarily close approximation to optimal value function V*.
- If we are given just V*, do we know which action to take in a state?
- What if we are also given the transition function T?
- Use *greedy* policy: (one step lookahead)

$$gr[V^*](s) = \arg\max_{a} R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V^*(s')$$

Computing an Optimal Value Function

- **Bellman equation** for optimal value function

$$V^*(s) = \underbrace{\max_a R(s,a)}_{\text{immediate reward}} + \underbrace{\beta \sum_{s'} T(s,a,s') \cdot V^*(s')}_{\text{discounted expected value of best action assuming we we get optimal value in future}}$$

- Bellman proved this is always true for an optimal value function

Computing an Optimal Value Function

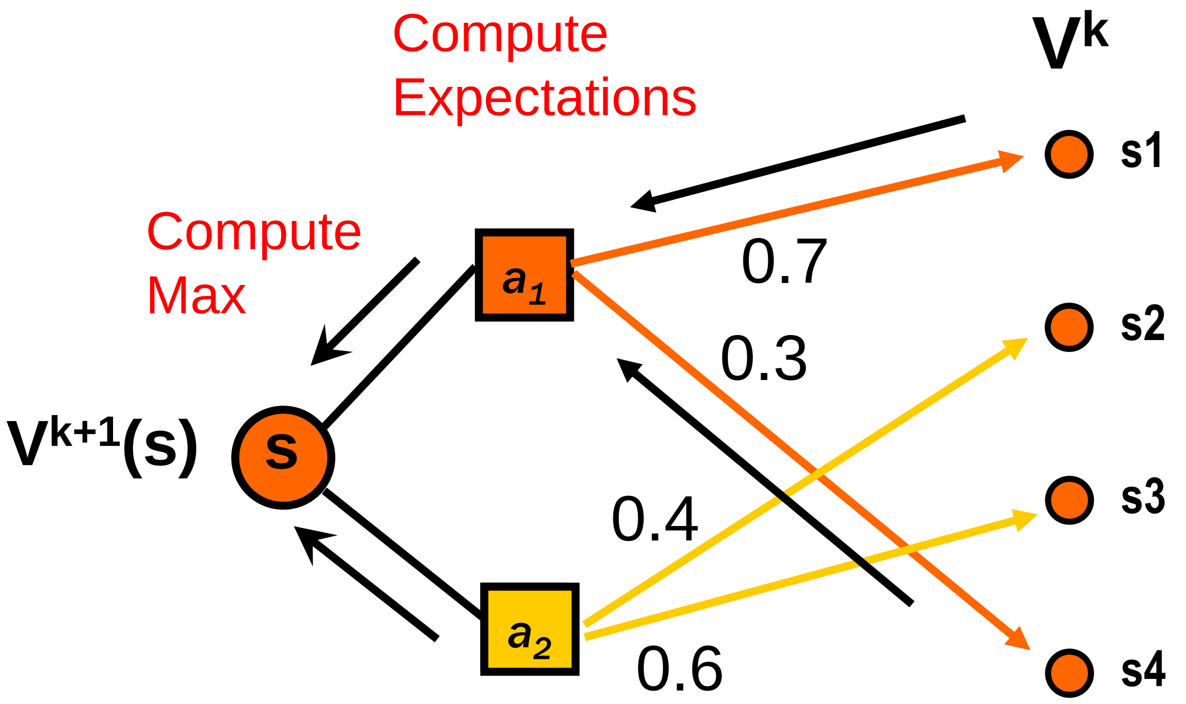
- **Bellman equation** for optimal value function

$$V^*(s) = \max_a R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V^*(s')$$

- How can we solve this equation for V*?
  - The MAX operator makes the system non-linear, so the problem is more difficult than policy evaluation

- **Idea:** lets pretend that we have a finite, but very, very long, horizon and apply finite-horizon value iteration
  - Adjust Bellman Backup to take discounting into account.

Bellman Backups (Revisited)



$$V^{k+1}(s) = \max_a R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V^k(s')$$

Value Iteration

- Can compute optimal policy using value iteration based on Bellman backups, just like finite-horizon problems (but include discount term)

$$V^0(s) = 0$$

$$V^k(s) = \max_a R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

- Do we need to store all of the V^k in memory?
  - No. We only need to store the latest value function V^{k-1}, to compute the updated V^k

Value Iteration

- Can compute optimal policy using value iteration based on Bellman backups, just like finite-horizon problems (but include discount term)

$$V^0(s) = 0$$

$$V^k(s) = \max_a R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

- Will it converge to optimal value function as k gets large?
  - Yes. $\lim_{k \to \infty} V^k = V^*$
- Why? When should we stop iterating in practice?

Convergence of Value Iteration

- **Bellman Backup Operator:** define B to be an operator that takes a value function V as input and returns a new value function after a Bellman backup
  - Think of V and $B[V]$ as vectors indexed by states

$$B[V](s) = \max_{a} R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V(s')$$

- Value iteration is just the iterative application of B:

$$V^0 = 0$$

$$V^k = B[V^{k-1}]$$

Convergence: Fixed Point Property

- **Bellman equation** for optimal value function

$$V^*(s) = \max_a R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V^*(s')$$

- **Fixed Point Property:** The optimal value function is a **fixed-point** of the Bellman Backup operator B.
  - That is B[V*]=V*

$$B[V^*](s) = \max_a R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V^*(s') = V^*$$

Convergence: Contraction Property

- Let ||V|| denote the max-norm of V, which returns the maximum absolute value of the vector.
  - E.g. ||(0.1 -100 5 6)|| = 100

- $B[V]$ is a **contraction operator** wrt max-norm

  For any V and V', $\|\boldsymbol{B}[\boldsymbol{V}] - \boldsymbol{B}[\boldsymbol{V'}]\| \leq \beta \|\boldsymbol{V} - \boldsymbol{V'}\|$
  - You will prove this.

- That is, applying B to any two value functions causes them to get closer together in the max-norm sense!

Convergence

- Using the properties of B we can prove convergence of value iteration.
- Proof:
  1. For any V: $\| V^* - B[V] \| = \| B[V^*] - B[V] \| \leq \beta \| V^* - V \|$
  2. So applying Bellman backup to any value function V brings us closer to V* by a constant factor β

     $\|V^* - V^{k+1}\| = \|V^* - B[V^k]\| \leq \beta \| V^* - V^k \|$
  3. This means that $\|V^* - V^k\| \leq \beta^k \| V^* - V^0 \|$
  4. Thus $\lim_{k\to\infty} \left\| V^* - V^k \right\| = 0$

Value Iteration: Stopping Condition

- Want to stop when we can guarantee the value function is near optimal.

- Key property: (not hard to prove)

If $||V^k - V^{k-1}|| \leq \varepsilon$ then $||V^k - V^*|| \leq \varepsilon\beta/(1-\beta)$

- Continue iteration until $||V^k - V^{k-1}|| \leq \varepsilon$
  - Select small enough ε for desired error guarantee

How to Act

- Given a V^k from value iteration that closely approximates V^*, what should we use as our policy?

- Use *greedy* policy: (one step lookahead)

$$gr[V^k](s) = \arg\max_{a} R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V^k(s')$$

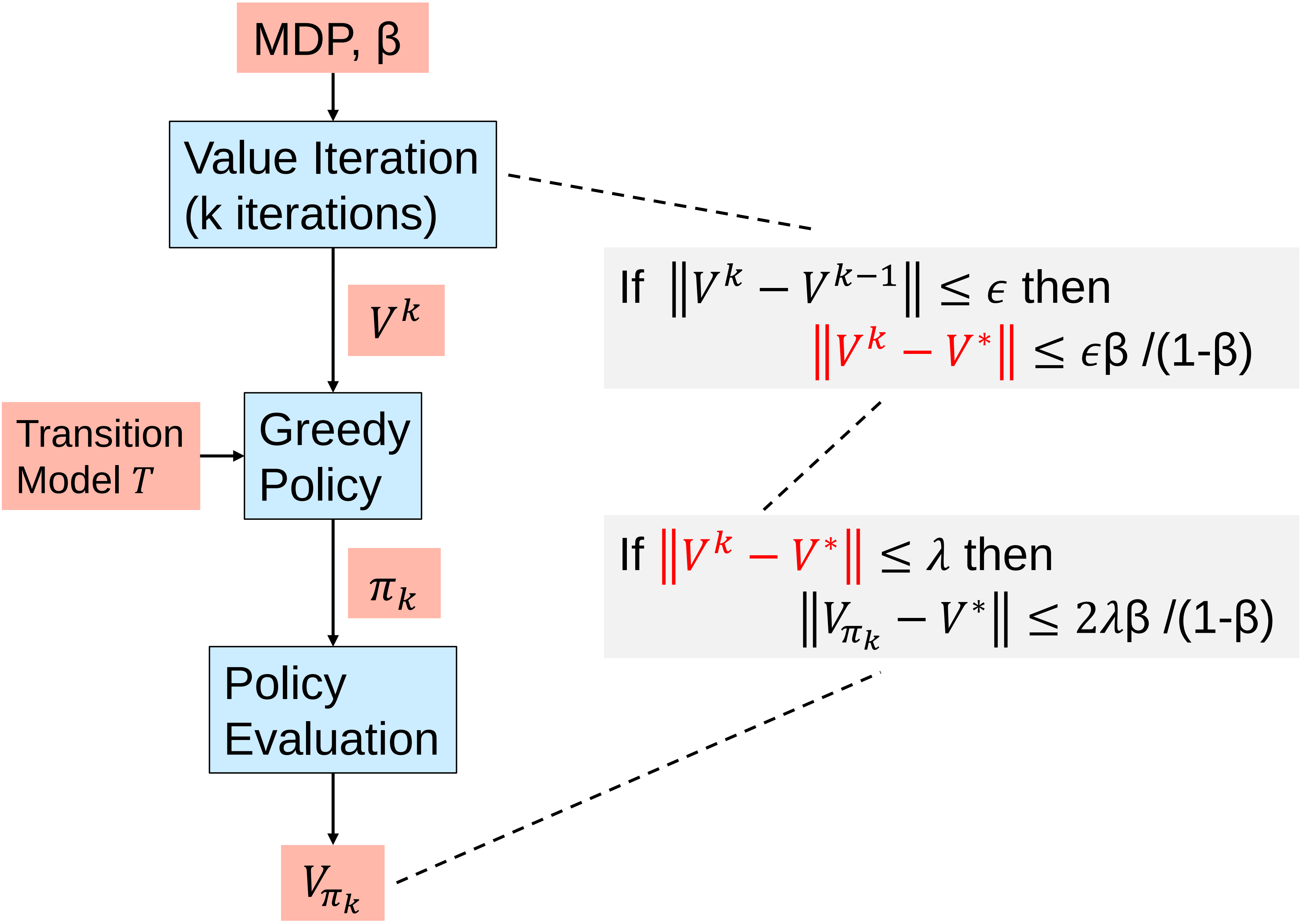
- Note that the value of greedy policy may not be exactly equal to V^k
  - Why?

How to Act

- Use *greedy* policy: (one step lookahead)

$$gr[V^k](s) = \arg\max_a R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V^k(s')$$

- For simplicity, define $\pi_k(s) = gr[V^k](s)$
- We care about the **value of the greedy policy** which we denote by V_{π_k}
  - This is how good the greedy policy will be in practice.
- How close is V_{π_k} to V^*?
  - What is the price for acting greedily with respect to a close approximation to V* compared to V*?


MDP, β
Value Iteration (k iterations)
V^k
Transition Model T
Greedy Policy
π_k
Policy Evaluation
V_{π_k}
If $\|V^k - V^{k-1}\| \leq \epsilon$ then
$\|V^k - V^*\| \leq \epsilon\beta/(1-\beta)$
If $\|V^k - V^*\| \leq \lambda$ then
$\|V_{\pi_k} - V^*\| \leq 2\lambda\beta/(1-\beta)$

Value of Greedy Policy

$$\pi_k(s) = gr[V^k](s)$$

- Define V_{π_k} to be the value of this greedy policy
  - This is likely not the same as V^k

- **Property:** If $||V^k – V^*|| \leq \lambda$ then $||V_{\pi_k}$ - $V^*|| \leq 2\lambda\beta /(1-\beta)$
  - Thus, V_{π_k} is not too far from optimal if V^k is close to optimal

- Our previous stopping condition allows us to bound λ based on $||V^{k+1} – V^k||$

- Set stopping condition so that $||V_{\pi_k}$ - $V^*|| \leq \Delta$
  - How?

Goal: $||V_{\pi_k} - V^*|| \leq \Delta$

Property: If $||V^k - V^*|| \leq \lambda$ then $||V_{\pi_k} - V^*|| \leq 2\lambda\beta/(1-\beta)$

Property: If $||V^k - V^{k-1}|| \leq \varepsilon$ then $||V^k - V^*|| \leq \varepsilon\beta/(1-\beta)$

Answer: If $||V^k - V^{k-1}|| \leq (1-\beta)^2\Delta/(2\beta^2)$ then $||V_{\pi_k} - V^*|| \leq \Delta$

Asynchronous Value Iteration

We just considered synchronous value iteration:

At iteration k perform Bellman Backup at ALL states.

$$V^k(s) = \max_a R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

Asynchronous Value Iteration:

At iteration k perform a Bellman Backup on a random state s (V^k only differs from V^{k-1} at a single state)

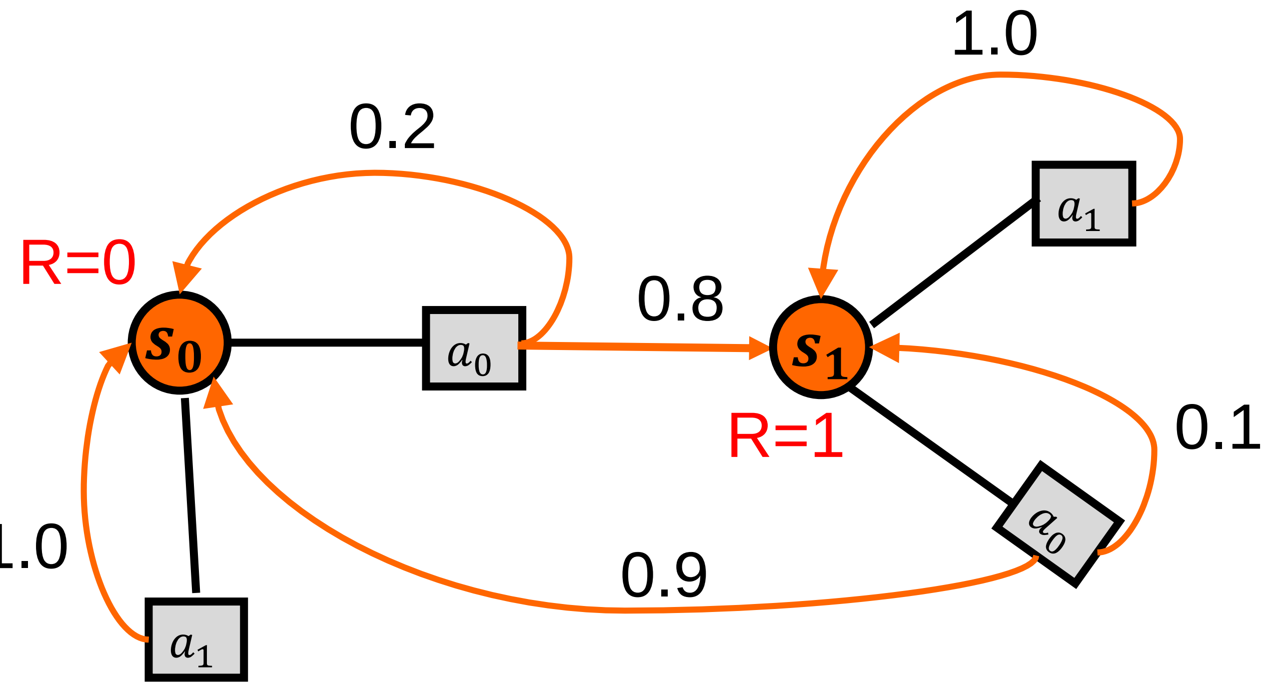
Asynchronous Value Iteration converges as long as all states are updated infinitely often. Order of updates does not mater.

Do we need to store full copies of both V^k and V^{k+1} as in VI?

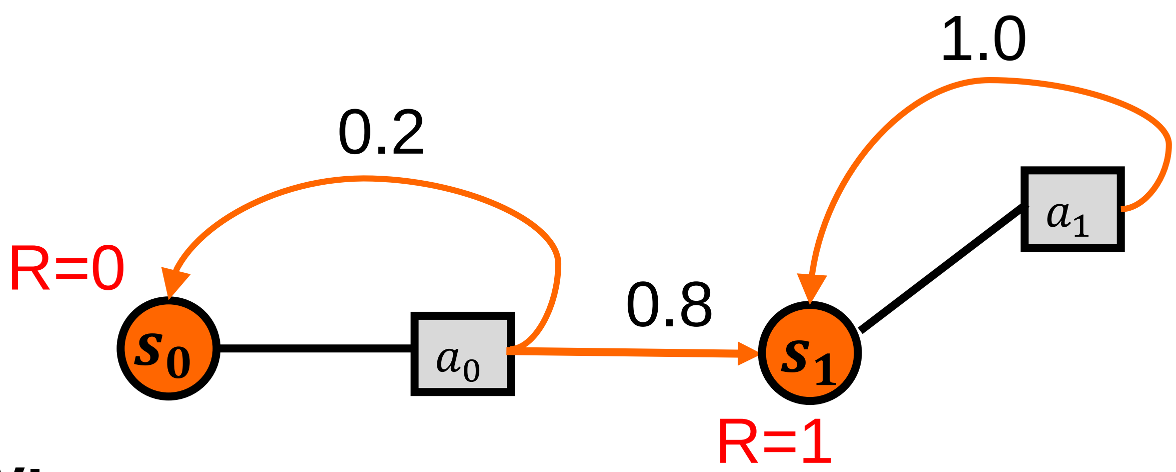
Policy Evaluation Revisited

- Sometimes policy evaluation is expensive due to matrix operations
- Can we have an iterative algorithm like value iteration for policy evaluation?
- **Idea:** Given a policy π and MDP M, create a new MDP $M[\pi]$ that is identical to M, except that in each state s we only allow a single action π(s)

MDP M



MDP $M[\pi]$



Example policy:

$\pi(s_0) = a_0, \pi(s_1) = a_1$

What is V^* for $M[\pi]$?

Policy Evaluation Revisited

- Sometimes policy evaluation is expensive due to matrix operations
- Can we have an iterative algorithm like value iteration for policy evaluation?
- **Idea:** Given a policy π and MDP M, create a new MDP $M[\pi]$ that is identical to M, except that in each state s we only allow a single action **π**(s)
  - What is the optimal value function V* for $M[\pi]$?
- Since the only valid policy for $M[\pi]$ is π, $\mathrm{V}^* = V_{\pi}$.

Policy Evaluation Revisited

- Running VI on $M[\pi]$ will converge to $V^* = V_\pi$.
  - What does the Bellman backup look like here?

- The Bellman backup now only considers one action in each state, so there is no max
  - We are effectively applying a backup restricted by π

Restricted Bellman Backup:

$$B_\pi[V](s) = R(s,\pi(s)) + \beta \sum_{s'} T(s,\pi(s),s') \cdot V(s')$$

Iterative Policy Evaluation

- Running VI on $M[\pi]$ is equivalent to iteratively applying the restricted Bellman backup.

Iterative Policy Evaluation:

$$V^0 = 0$$

$$V^k = B_\pi[V^{k-1}]$$

Convergence: $\lim_{k\to\infty} V^k = V_\pi$

- Often become close to V_π for small k

Computational Problems

- Policy Evaluation
  - Given π and an MDP compute V_π
- Policy Optimization
  - Given an MDP, compute an optimal policy π^* and V^*.
  - We'll cover two algorithms for doing this: value iteration and policy iteration

Optimization via Policy Iteration

- Policy iteration uses policy evaluation as a sub routine for optimization
- It iterates steps of policy evaluation and policy improvement

1. Choose a random policy π
2. Loop:
   (a) Evaluate V_π
   (b) π' = ImprovePolicy(V_π)
   (c) Replace π with π'

Until no improving action possible at any state

Given V_π returns a strictly better policy if π isn't optimal

Policy Improvement

- Given V_π how can we compute a policy π' that is strictly better than a sub-optimal π?
- **Idea:** given a state s, take the action that looks the best assuming that we following policy π thereafter
  - That is, assume the next state s' has value $V_\pi(s')$

Action Values for Policy Improvement

- The **Q-function** is widely used in MDP literature for assigning values to actions

- $Q_\pi(s,a)$ is expected discounted cumulative reward of taking action a in s and then following π thereafter

$$Q_\pi(s,a) = R(s,a) + \beta \sum_{s'} T(s,a,s') V_\pi(s')$$

- **Improved Policy π':** act greedily according to Q_π

For each s in S, $\pi'(s) = \arg\max_{a \in A} Q_\pi(s,a) = gr[V_\pi](s)$

Policy Improvement Theorem

If π is not optimal then π' is strictly better than π !

For any two value functions V_1 and V_2, we write $V_1 \geq V_2$ to indicate that for all states s, $V_1(s) \geq V_2(s)$.

Proposition: $V_{\pi'} \geq V_{\pi}$ with strict inequality for sub-optimal π.

Useful Properties for Proof:

1) $V_\pi = \mathrm{B}_\pi[\mathrm{V}_\pi]$;; fixed point property
2) $B[V_\pi] = B_{\pi'}[V_\pi]$;; by the definition of π'
3) For any V_1, V_2 and π, if $V_1 \geq V_2$ then $B_\pi[V_1] \geq B_\pi[V_2]$

$$\pi'(s) = \arg\max_{a \in A} Q_\pi(s, a)$$

Proposition: $V_{\pi'} \geq V_\pi$ with strict inequality for sub-optimal π.

Proof: (first part, non-strict inequality)

We know that $V_\pi = B_\pi[V_\pi] \leq B[V_\pi] = B_{\pi'}[V_\pi]$

So we have that $V_\pi \leq B_{\pi'}[V_\pi]$.

Now by monotonicity we get $B_{\pi'}[V_\pi] \leq B_{\pi'}^2[V_\pi]$ where $B_{\pi'}^k$ denotes k applications of $B_{\pi'}$.

We can continue and derive that in general for any k, $B_{\pi'}^k[V_\pi] \leq B_{\pi'}^{k+1}[V_\pi]$, which also implies that $V_\pi \leq B_{\pi'}^k[V_\pi]$ for any k.

Thus $V_\pi \leq \lim_{k \to \infty} B_{\pi'}^k[V_\pi] = V_{\pi'}$

$$\pi'(s) = \arg\max_{a \in A} Q_\pi(s, a)$$

Proposition: $V_{\pi'} \geq V_\pi$ with strict inequality for sub-optimal π.

Proof: (part two, strict inequality)

We want to show that ***if π is sub-optimal then*** $\boldsymbol{V_{\pi'} > V_\pi}$.

We prove the contrapositive ***if*** $\boldsymbol{\neg(V_{\pi'} > V_\pi)}$ ***then π is optimal.***

Since we already showed that $V_{\pi'} \geq V_\pi$ we know that the condition of the contrapositive $\neg(V_{\pi'} > V_\pi)$ is equivalent to $V_{\pi'} = V_\pi$.

Now assume that $V_{\pi'} = V_\pi$. Combining this with $V_{\pi'} = B_{\pi'}[V_{\pi'}]$ yields $V_\pi = B_{\pi'}[V_\pi] = B[V_\pi]$.

Thus V_π satisfies the Bellman Equation and must be optimal.

Optimization via Policy Iteration

1. Choose a random policy π
2. Loop:
   (a) Evaluate V_π
   (b) For each s in S, set $\pi'(s) = \arg\max_{a \in A} Q_\pi(s,a)$
   where $Q_\pi(s,a) = R(s,a) + \beta \sum_{s'} T(s,a,s') V_\pi(s')$
   (c) Replace π with π'

Until no improving action possible at any state

Proposition: $V_{\pi'} \geq V_\pi$ with strict inequality for sub-optimal π.

Policy iteration goes through a sequence of improving policies

Policy Iteration: Convergence

- Convergence assured in a finite number of iterations
  - Since finite number of policies and each step improves value, then must converge to optimal
- Gives exact value of optimal policy

Policy Iteration Complexity

- Each iteration runs in polynomial time in the number of states and actions
- There are at most $|A|^n$ policies and PI never repeats a policy
  - So at most an exponential number of iterations
  - Not a very good complexity bound
- Empirically O(n) iterations are required often it seems like O(1)
  - **Challenge:** try to generate an MDP that requires more than that n iterations
- Recent theoretical progress

Policy Iteration Complexity

- Recently it has been shown that for a fixed discount factor β, the max number of iterations of PI is $O\left(\frac{|A|}{1-\beta}\log\left(\frac{|S|}{1-\beta}\right)\right)$
  - So it is polynomial in the # of states and actions for a fixed β
  - But this bound is horrible for $\beta \approx 1$
- In general if we do not treat β as a constant, PI has been shown to run for an exponential number of iterations for some MDPs
  - That is, there are MDPs and values of the discount factor that will cause PI to take exponential time
  - Of course these are quite pathological MDPs

Value Iteration vs. Policy Iteration

- Which is faster? VI or PI
  - It depends on the problem
- VI takes more iterations than PI, but PI requires more time on each iteration
  - PI must perform policy evaluation on each iteration which involves solving a linear system
- VI is easier to implement since it does not require the policy evaluation step
  - But see next slide
- We will see that both algorithms will serve as inspiration for more advanced algorithms

Modified Policy Iteration

- **Modified Policy Iteration:** replaces exact policy evaluation step with inexact iterative evaluation
  - Uses a small number of restricted Bellman backups for evaluation

- Avoids the expensive policy evaluation step
- Perhaps easier to implement.
- Often is faster than PI and VI
- Still guaranteed to converge under mild assumptions on starting points

Modified Policy Iteration

Policy Iteration

1. Choose initial value function V
2. Loop:
   (a) For each s in S, set $\pi(s) = gr[V](s)$

   (b) Partial Policy Evaluation
      Repeat K times: $V \leftarrow B_{\pi}[V]$

   Approx. evaluation

Until change in V is minimal

Generalized Policy Iteration

If we make MPI asynchronous then we get what Sutton & Barto refer to as **Generalized Policy Iteration**.

- Each iteration selects a state s to update and then selects whether to do policy eval update or VI update to s.

Choose initial value function V and initial policy π

Loop:

Select a state s ;; non-zero prob for all states

Choose one of the following ;; *non-zero prob of either*

i) **policy eval:** $V(s) = B_\pi[V](s)$

ii) **policy improve:** $\pi(s) = \arg\max_a R(s,a) + \beta \sum_{s'} T(s,a,s') \cdot V(s')$

Until change in V is minimal

Many DP and RL algorithms can be put in this framework.

Recap: things you should know

- What is an MDP?
- What is a policy?
  - Stationary and non-stationary
- What is a value function?
  - Finite-horizon and infinite horizon
- How to evaluate policies?
  - Finite-horizon and infinite horizon
  - Time/space complexity?
- How to optimize policies?
  - Finite-horizon and infinite horizon
  - Time/space complexity?
  - Why they are correct?