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**

Discounted Infinite Horizon MDPs

- Defining value as total reward is problematic with infinite horizons (r1 + r2 + r3 + r4 + …..)
  - many or all policies have infinite expected reward
  - some MDPs are ok (e.g., zero-cost absorbing states)
- "Trick": introduce discount factor $0 \le \beta < 1$
  - future rewards discounted by β per time step

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

Bounded Value

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

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

immediate reward

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

Policy Evaluation

- Value equation for fixed policy

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

- How can we compute the value function for a fixed policy?
  - we are given R, T, and B and want to find $V_\pi(s)$ for each s
  - linear system with n variables and n constraints
    - Variables are values of states: V(s1),…,V(sn)
    - Constraints: one value equation (above) per state
  - Use linear algebra to solve for V (e.g. matrix inverse)

Policy Evaluation via Matrix Inverse

V_π and ***R*** are n-dimensional column vector (one element for each state)

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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

Computing an Optimal Value Function

- **Bellman equation** for optimal value function

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

immediate reward

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

Computing an Optimal Value Function

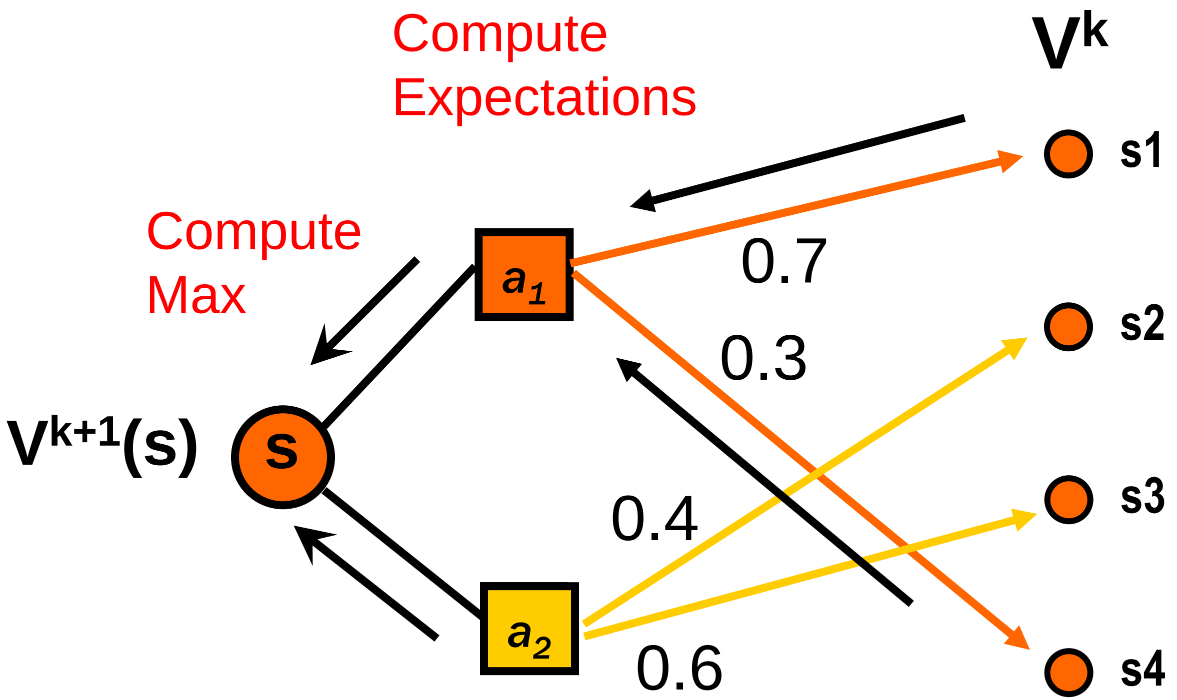
- **Bellman equation** for optimal value function

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

- 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)



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

Value Iteration

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

$$V^0(s) = 0$$;; Could also initialize to R(s)

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

- Will it converge to optimal value function as k gets large?
  - Yes. $\lim_{k\to\infty} V^k = V^*$
- Why?

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

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

- 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

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

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

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

Convergence: Contraction Property

- Let ||V|| denote the max-norm of V, which returns the maximum element 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', $\| B[V] - B[V'] \| \leq \beta \, \| V - V' \|$
  - 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 $\|V^k - V^*\| \leq \beta^k \| V^* - V^0 \|$

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)

$$greedy[V^k](s) = \underset{a}{\arg\max} \sum_{s'} T(s,a,s') \cdot V^k(s')$$

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

How to Act

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

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

- We care about the **value of the greedy policy** which we denote by $\mathbf{V_g}$
  - This is how good the greedy policy will be in practice.

- How close is V_g to V^*?

Value of Greedy Policy

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

- Define V_g 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 $||V_g - V^*|| \leq 2\lambda\beta/(1-\beta)$
  - Thus, V_g 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 $||V_g - V^*|| \leq \Delta$
  - How?

Goal: $||V_g - V^*|| \leq \Delta$

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

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

Answer: If $||V^k - V^{k-1}|| \leq (1-\mathrm{B})^2\Delta/(2\mathrm{B}^2)$ then $||V_g - V^*|| \leq \Delta$

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[π] 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[π] ?
- Since the only valid policy for M[π] is π, $V^* = V_{\pi}$.

Policy Evaluation Revisited

- Running VI on M[π] 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:

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

Iterative Policy Evaluation

- Running VI on M[π] 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

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_π (s')

For each s in S, set $\pi'(s) = \arg\max_a \sum_{s'} T(s,a,s') \cdot V_\pi(s')$

Proposition: $V_{\pi'} \geq V_\pi$ with strict inequality for sub-optimal 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, $V_1(s) \geq V_2(s)$.

$$\pi'(s) = \underset{a}{\arg\max} \sum_{s'} T(s,a,s') \cdot V_\pi(s')$$

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

Useful Properties for Proof:

1) $V_\pi = \mathrm{B}_\pi[\mathrm{V}_\pi]$
2) For any V_1, V_2 and π, if $V_1 \geq V_2$ then $B_\pi[V_1] \geq B_\pi[V_2]$

$$\pi'(s) = \arg\max_{a} \sum_{s'} T(s,a,s') \cdot V_{\pi}(s')$$

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

Proof:

$$\pi'(s) = \arg\max_{a} \sum_{s'} T(s,a,s') \cdot V_{\pi}(s')$$

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

Proof:

Optimization via Policy Iteration

1. Choose a random policy π
2. Loop:
   (a) Evaluate V_π
   (b) For each s in S, set $\pi'(s) = \arg\max_{a} \sum_{s'} T(s,a,s') \cdot V_\pi(s')$
   (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
- Still no polynomial bound on the number of PI iterations (open problem)!
  - But may have been solved recently ????…..

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 $\pi(s) = \underset{a}{\arg\max} \sum_{s'} T(s,a,s') \cdot V(s')$

   (b) Partial Policy Evaluation

   Repeat K times: $V \leftarrow B_{\pi}[V]$ Approx. evaluation

Until change in V is minimal

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?