

A Guided Tour of Chapter 5: Dynamic Programming

Ashwin Rao

ICME, Stanford University

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP
- (Optimal Policy can be extracted from Optimal Value Function)
- Planning versus Learning: access to the \mathcal{P}_R function ("model")
- Original use of *DP* term: MDP Theory *and* solution methods
- Bellman referred to DP as the *Principle of Optimality*
- Later, the usage of the term DP diffused out to other algorithms
- In CS, it means "recursive algorithms with overlapping subproblems"
- We restrict the term DP to: "Algorithms for Prediction and Control"
- Specifically applied to the setting of FiniteMarkovDecisionProcess
- Later we cover extensions such as Asynchronous DP, Approximate DP

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms
- Each of the 3 algorithms is founded on the Bellman Equations
- Each is an iterative algorithm converging to the true Value Function
- Each algorithm is based on the concept of *Fixed-Point*

Definition

The Fixed-Point of a function $f : \mathcal{X} \rightarrow \mathcal{X}$ (for some arbitrary domain \mathcal{X}) is a value $x \in \mathcal{X}$ that satisfies the equation: $x = f(x)$.

- Some functions have multiple fixed-points, some have none
- DP algorithms are based on functions with a unique fixed-point
- Simple example: $f(x) = \cos(x)$, Fixed-Point: $x^* = \cos(x^*)$
- For any x_0 , $\cos(\cos(\dots \cos(x_0) \dots))$ converges to fixed-point x^*
- Why does this work? How fast does it converge?

Banach Fixed-Point Theorem

Theorem (Banach Fixed-Point Theorem)

Let \mathcal{X} be a non-empty set equipped with a complete metric $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. Let $f : \mathcal{X} \rightarrow \mathcal{X}$ be such that there exists a $L \in [0, 1)$ such that $d(f(x_1), f(x_2)) \leq L \cdot d(x_1, x_2)$ for all $x_1, x_2 \in \mathcal{X}$. Then,

- There exists a unique Fixed-Point $x^* \in \mathcal{X}$, i.e.,

$$x^* = f(x^*)$$

- For any $x_0 \in \mathcal{X}$, and sequence $[x_i | i = 0, 1, 2, \dots]$ defined as $x_{i+1} = f(x_i)$ for all $i = 0, 1, 2, \dots$,

$$\lim_{i \rightarrow \infty} x_i = x^*$$

If you have a complete metric space $\langle \mathcal{X}, d \rangle$ and a contraction f (with respect to d), then you have an algorithm to solve for the fixed-point of f .

Policy Evaluation (for Prediction)

- MDP with $\mathcal{S} = \{s_1, s_2, \dots, s_n\}, \mathcal{N} = \{s_1, s_2, \dots, s_m\}$
- Given a policy π , compute the Value Function of π -implied MRP
- $\mathcal{P}_R^\pi : \mathcal{N} \times \mathcal{D} \times \mathcal{S} \rightarrow [0, 1]$ is given as a data structure
- Extract (from \mathcal{P}_R^π) $\mathcal{P}^\pi : \mathcal{N} \times \mathcal{S} \rightarrow [0, 1]$ and $\mathcal{R}^\pi : \mathcal{N} \rightarrow \mathbb{R}$
- For non-large spaces, we can compute (in vector notation):

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

- Note: $\mathbf{V}^\pi, \mathbf{R}^\pi$ are m -column vectors ($\in \mathbb{R}^m$) and \mathbf{P}^π is $m \times m$ matrix
- So we look for an iterative algorithm to solve MRP Bellman Equation:

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

Bellman Policy Operator and its Fixed-Point

- Define the *Bellman Policy Operator* $\mathbf{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ as:

$$\mathbf{B}^\pi(\mathbf{V}) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \mathbf{V} \text{ for any Value Function vector } \mathbf{V} \in \mathbb{R}^m$$

- \mathbf{B}^π is an affine transformation on vectors in \mathbb{R}^m
- So, the MRP Bellman Equation can be expressed as:

$$\mathbf{V}^\pi = \mathbf{B}^\pi(\mathbf{V}^\pi)$$

- This means $\mathbf{V}^\pi \in \mathbb{R}^m$ is a Fixed-Point of $\mathbf{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$
- Metric $d : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$ defined as L^∞ norm:

$$d(\mathbf{X}, \mathbf{Y}) = \|\mathbf{X} - \mathbf{Y}\|_\infty = \max_{s \in \mathcal{N}} |(\mathbf{X} - \mathbf{Y})(s)|$$

- \mathbf{B}^π is a contraction function under L^∞ norm: For all $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^m$,

$$\begin{aligned} \max_{s \in \mathcal{N}} |(\mathbf{B}^\pi(\mathbf{X}) - \mathbf{B}^\pi(\mathbf{Y}))(s)| &= \gamma \cdot \max_{s \in \mathcal{N}} |(\mathcal{P}^\pi \cdot (\mathbf{X} - \mathbf{Y}))(s)| \\ &\leq \gamma \cdot \max_{s \in \mathcal{N}} |(\mathbf{X} - \mathbf{Y})(s)| \end{aligned}$$

Policy Evaluation Convergence Theorem

Invoking the Banach Fixed-Point Theorem for $\gamma < 1$ gives:

Theorem (Policy Evaluation Convergence Theorem)

For a Finite MDP with $|\mathcal{N}| = m$ and $\gamma < 1$, if $\mathbf{V}^\pi \in \mathbb{R}^m$ is the Value Function of the MDP when evaluated with a fixed policy $\pi : \mathcal{N} \times \mathcal{A} \rightarrow [0, 1]$, then \mathbf{V}^π is the unique Fixed-Point of the Bellman Policy Operator $\mathbf{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$, and

$$\lim_{i \rightarrow \infty} (\mathbf{B}^\pi)^i(\mathbf{V}_0) \rightarrow \mathbf{V}^\pi \text{ for all starting Value Functions } \mathbf{V}_0 \in \mathbb{R}^m$$

Policy Evaluation algorithm

- Start with any Value Function $\mathbf{V}_0 \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \dots$, calculate in each iteration:

$$\mathbf{V}_{i+1} = \mathbf{B}^\pi(\mathbf{V}_i) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \mathbf{V}_i$$

- Stop when $d(\mathbf{V}_i, \mathbf{V}_{i+1}) = \max_{s \in \mathcal{N}} |(\mathbf{V}_i - \mathbf{V}_{i+1})(s)|$ is small enough

Banach Fixed-Point Theorem also assures speed of convergence (dependent on choice of starting Value Function \mathbf{V}_0 and on choice of γ).

Running time of each iteration is $O(m^2)$. Constructing the MRP from the MDP and the policy takes $O(m^2 k)$ operations, where $m = |\mathcal{N}|$, $k = |\mathcal{A}|$.

Greedy Policy

- Now we move on to solving the MDP *Control* problem
- We want to iterate *Policy Improvements* to drive to an *Optimal Policy*
- *Policy Improvement* is based on a “greedy” technique
- The *Greedy Policy Function* $G : \mathbb{R}^m \rightarrow (\mathcal{N} \rightarrow \mathcal{A})$
(interpreted as a function mapping a Value Function vector \mathbf{V} to a deterministic policy $\pi'_D : \mathcal{N} \rightarrow \mathcal{A}$) is defined as:

$$G(\mathbf{V})(s) = \pi'_D(s) = \arg \max_{a \in \mathcal{A}} \{ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \mathbf{V}(s') \}$$

Definition (Value Function Comparison)

We say $X \geq Y$ for Value Functions $X, Y : \mathcal{N} \rightarrow \mathbb{R}$ of an MDP iff:

$$X(s) \geq Y(s) \text{ for all } s \in \mathcal{N}$$

We say π_1 better (“improvement”) than π_2 if $\mathbf{V}^{\pi_1} \geq \mathbf{V}^{\pi_2}$

Policy Improvement Theorem

Theorem (Policy Improvement Theorem)

For a finite MDP, for any policy π ,

$$\mathbf{V}^{\pi'_D} = \mathbf{V}^{G(\mathbf{V}^\pi)} \geq \mathbf{V}^\pi$$

- Note that applying $\mathbf{B}^{\pi'_D} = \mathbf{B}^{G(\mathbf{V}^\pi)}$ repeatedly, starting with \mathbf{V}^π , will converge to $\mathbf{V}^{\pi'_D}$ (Policy Evaluation with policy $\pi'_D = G(\mathbf{V}^\pi)$):

$$\lim_{i \rightarrow \infty} (\mathbf{B}^{\pi'_D})^i(\mathbf{V}^\pi) = \mathbf{V}^{\pi'_D}$$

- So the proof is complete if we prove that:

$$(\mathbf{B}^{\pi'_D})^{i+1}(\mathbf{V}^\pi) \geq (\mathbf{B}^{\pi'_D})^i(\mathbf{V}^\pi) \text{ for all } i = 0, 1, 2, \dots$$

- Non-decreasing tower of Value Functions $[(\mathbf{B}^{\pi'_D})^i(\mathbf{V}^\pi) | i = 0, 1, 2, \dots]$ with repeated applications of $\mathbf{B}^{\pi'_D}$

Proof by Induction

- To prove the base case (of proof by induction), note that:

$$\mathbf{B}^{\pi'_D}(\mathbf{V}^\pi)(s) = \max_{a \in \mathcal{A}} \{ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \mathbf{V}^\pi(s') \} = \max_{a \in \mathcal{A}} Q^\pi(s, a)$$

- $\mathbf{V}^\pi(s)$ is weighted average of $Q^\pi(s, \cdot)$ while $\mathbf{B}^{\pi'_D}(\mathbf{V}^\pi)(s)$ is maximum

$$\mathbf{B}^{\pi'_D}(\mathbf{V}^\pi) \geq \mathbf{V}^\pi$$

- Induction step is proved by monotonicity of \mathbf{B}^π operator (for any π):

Monotonicity Property of $\mathbf{B}^\pi : \mathbf{X} \geq \mathbf{Y} \Rightarrow \mathbf{B}^\pi(\mathbf{X}) \geq \mathbf{B}^\pi(\mathbf{Y})$

$$\text{So } (\mathbf{B}^{\pi'_D})^{i+1}(\mathbf{V}^\pi) \geq (\mathbf{B}^{\pi'_D})^i(\mathbf{V}^\pi) \Rightarrow (\mathbf{B}^{\pi'_D})^{i+2}(\mathbf{V}^\pi) \geq (\mathbf{B}^{\pi'_D})^{i+1}(\mathbf{V}^\pi)$$

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\mathbf{B}^{\pi'_D})^i(\mathbf{V}^\pi) | i = 0, 1, 2, \dots]$
- Each stage of further application of $\mathbf{B}^{\pi'_D}$ improves the Value Function
- Stage 0: Value Function \mathbf{V}^π means execute policy π throughout
- Stage 1: VF $\mathbf{B}^{\pi'_D}(\mathbf{V}^\pi)$ means execute improved policy π'_D for the 1st time step, then execute policy π for all further time steps
- Improves the VF from Stage 0: \mathbf{V}^π to Stage 1: $\mathbf{B}^{\pi'_D}(\mathbf{V}^\pi)$
- Stage 2: VF $(\mathbf{B}^{\pi'_D})^2(\mathbf{V}^\pi)$ means execute improved policy π'_D for first 2 time steps, then execute policy π for all further time steps
- Improves the VF from Stage 1: $\mathbf{B}^{\pi'_D}(\mathbf{V}^\pi)$ to Stage 2: $(\mathbf{B}^{\pi'_D})^2(\mathbf{V}^\pi)$
- Each stage applies policy π'_D instead of π for an extra time step
- These stages are the iterations of *Policy Evaluation* (using policy π'_D)
- Builds non-decreasing tower of VFs that converge to VF $\mathbf{V}^{\pi'_D} (\geq \mathbf{V}^\pi)$

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:
 - Start with Value Function \mathbf{V}^π (for policy π)
 - Perform a “greedy policy improvement” to create policy $\pi'_D = G(\mathbf{V}^\pi)$
 - Perform Policy Evaluation (for policy π'_D) with starting VF \mathbf{V}^π
 - This results in VF $\mathbf{V}^{\pi'_D} \geq$ starting VF \mathbf{V}^π
- We can repeat this process starting with $\mathbf{V}^{\pi'_D}$
- Creating an improved policy π''_D and improved VF $\mathbf{V}^{\pi''_D}$.
- ... and we can keep going to create further improved policies/VFs
- ... until there is no further improvement
- This in fact is the *Policy Iteration* algorithm

Policy Iteration algorithm

- Start with any Value Function $\mathbf{V}_0 \in \mathbb{R}^m$
- Iterating over $j = 0, 1, 2, \dots$, calculate in each iteration:

$$\text{Deterministic Policy } \pi_{j+1} = G(\mathbf{V}_j)$$

$$\text{Value Function } \mathbf{V}_{j+1} = \lim_{i \rightarrow \infty} (\mathbf{B}^{\pi_{j+1}})^i(\mathbf{V}_j)$$

- Stop when $d(\mathbf{V}_j, \mathbf{V}_{j+1}) = \max_{s \in \mathcal{N}} |(\mathbf{V}_j - \mathbf{V}_{j+1})(s)|$ is small enough
At termination: $\mathbf{V}_j = (\mathbf{B}^{G(\mathbf{V}_j)})^i(\mathbf{V}_j) = \mathbf{V}_{j+1}$ for all $i = 0, 1, 2, \dots$

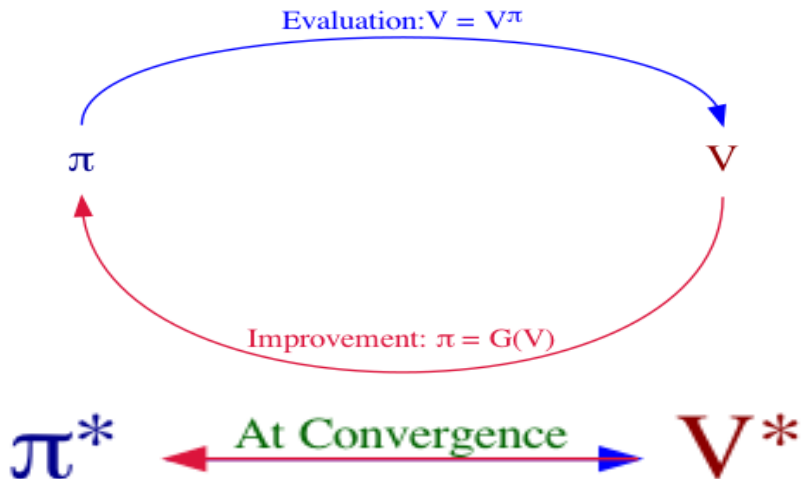
Specializing this to $i = 1$, we have for all $s \in \mathcal{N}$:

$$\mathbf{V}_j(s) = \mathbf{B}^{G(\mathbf{V}_j)}(\mathbf{V}_j)(s) = \max_{a \in \mathcal{A}} \{ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \mathbf{V}_j(s') \}$$

This means \mathbf{V}_j satisfies the MDP Bellman Optimality Equation and so,

$$\mathbf{V}_j = \mathbf{V}^{\pi_j} = \mathbf{V}^*$$

Policy Iteration algorithm



Policy Iteration Convergence Theorem

Theorem (Policy Iteration Convergence Theorem)

For a Finite MDP with $|\mathcal{N}| = m$ and $\gamma < 1$, Policy Iteration algorithm converges to the Optimal Value Function $\mathbf{V}^ \in \mathbb{R}^m$ along with a Deterministic Optimal Policy $\pi_D^* : \mathcal{N} \rightarrow \mathcal{A}$, no matter which Value Function $\mathbf{V}_0 \in \mathbb{R}^m$ we start the algorithm with.*

Running time of Policy Improvement is $O(m^2k)$ where $|\mathcal{N}| = m, |\mathcal{A}| = k$

Running time of each iteration of Policy Evaluation is $O(m^2k)$

Bellman Optimality Operator

- Tweak the definition of Greedy Policy Function (arg max to max)
- *Bellman Optimality Operator* $\mathbf{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$ defined as:

$$\mathbf{B}^*(\mathbf{V})(s) = \max_{a \in \mathcal{A}} \{ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \mathbf{V}(s') \}$$

- Think of this as a non-linear transformation of a VF vector $\mathbf{V} \in \mathbb{R}^m$
- The action a producing the max is the action prescribed by $G(\mathbf{V})$. So,

$$\mathbf{B}^{G(\mathbf{V})}(\mathbf{V}) = \mathbf{B}^*(\mathbf{V}) \text{ for all } \mathbf{V} \in \mathbb{R}^m$$

- Specializing \mathbf{V} to be the Value Function \mathbf{V}^π for a policy π , we get:

$$\mathbf{B}^{G(\mathbf{V}^\pi)}(\mathbf{V}^\pi) = \mathbf{B}^*(\mathbf{V}^\pi)$$

- This is the 1st stage of Policy Evaluation with improved policy $G(\mathbf{V}^\pi)$

Fixed-Point of Bellman Optimality Operator

- \mathbf{B}^π was motivated by the MDP Bellman Policy Equation
- Similarly, \mathbf{B}^* is motivated by the MDP Bellman Optimality Equation:

$$\mathbf{V}^*(s) = \max_{a \in \mathcal{A}} \{ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \mathbf{V}^*(s') \} \text{ for all } s \in \mathcal{N}$$

- So we can express the MDP Bellman Optimality Equation neatly as:

$$\mathbf{V}^* = \mathbf{B}^*(\mathbf{V}^*)$$

- Therefore, $\mathbf{V}^* \in \mathbb{R}^m$ is a Fixed-Point of $\mathbf{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$
- We want to prove that \mathbf{B}^* is a contraction function (under L^∞ norm)
- So we can take advantage of Banach Fixed-Point Theorem
- And solve the Control problem by iterative applications of \mathbf{B}^*

Proof that B^* is a contraction

- We need to utilize two key properties of B^*

Monotonicity Property: $\mathbf{X} \geq \mathbf{Y} \Rightarrow B^*(\mathbf{X}) \geq B^*(\mathbf{Y})$

Constant Shift Property: $B^*(\mathbf{X} + c) = B^*(\mathbf{X}) + \gamma c$

- With these two properties, we can prove that:

$$\max_{s \in \mathcal{N}} |(B^*(\mathbf{X}) - B^*(\mathbf{Y}))(s)| \leq \gamma \cdot \max_{s \in \mathcal{N}} |(\mathbf{X} - \mathbf{Y})(s)|$$

Theorem (Value Iteration Convergence Theorem)

For a Finite MDP with $|\mathcal{N}| = m$ and $\gamma < 1$, if $\mathbf{V}^ \in \mathbb{R}^m$ is the Optimal Value Function, then \mathbf{V}^* is the unique Fixed-Point of the Bellman Optimality Operator $B^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$, and*

$$\lim_{i \rightarrow \infty} (B^*)^i(\mathbf{V}_0) \rightarrow \mathbf{V}^* \text{ for all starting Value Functions } \mathbf{V}_0 \in \mathbb{R}^m$$

Value Iteration algorithm

- Start with any Value Function $\mathbf{V}_0 \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \dots$, calculate in each iteration:

$$\mathbf{V}_{i+1}(s) = \mathbf{B}^*(\mathbf{V}_i)(s) \text{ for all } s \in \mathcal{N}$$

- Stop when $d(\mathbf{V}_i, \mathbf{V}_{i+1}) = \max_{s \in \mathcal{N}} |(\mathbf{V}_i - \mathbf{V}_{i+1})(s)|$ is small enough

Running time of each iteration of Value Iteration is $O(m^2k)$ where $|\mathcal{N}| = m$ and $|\mathcal{A}| = k$

Optimal Policy from Optimal Value Function

- Note that Value Iteration does not deal with any policy (only VFs)
- Extract Optimal Policy π^* from Optimal VF V^* such that $V^{\pi^*} = V^*$
- Use Greedy Policy function G . We know:

$$B^{G(V)}(V) = B^*(V) \text{ for all } V \in \mathbb{R}^m$$

- Specializing V to V^* , we get:

$$B^{G(V^*)}(V^*) = B^*(V^*)$$

- But we know V^* is the Fixed-Point of B^* , i.e., $B^*(V^*) = V^*$. So,

$$B^{G(V^*)}(V^*) = V^*$$

- So V^* is the Fixed-Point of the Bellman Policy Operator $B^{G(V^*)}$
- But we know $B^{G(V^*)}$ has a unique Fixed-Point ($= V^{G(V^*)}$). So,

$$V^{G(V^*)} = V^*$$

- Evaluating MDP with greedy policy extracted from V^* achieves V^*
- So, $G(V^*)$ is a (Deterministic) Optimal Policy

Value Function Progression in Policy Iteration

$$\pi_1 = G(\mathbf{V}_0) : \mathbf{V}_0 \rightarrow \mathbf{B}^{\pi_1}(\mathbf{V}_0) \rightarrow \dots (\mathbf{B}^{\pi_1})^i(\mathbf{V}_0) \rightarrow \dots \mathbf{V}^{\pi_1} = \mathbf{V}_1$$

$$\pi_2 = G(\mathbf{V}_1) : \mathbf{V}_1 \rightarrow \mathbf{B}^{\pi_2}(\mathbf{V}_1) \rightarrow \dots (\mathbf{B}^{\pi_2})^i(\mathbf{V}_1) \rightarrow \dots \mathbf{V}^{\pi_2} = \mathbf{V}_2$$

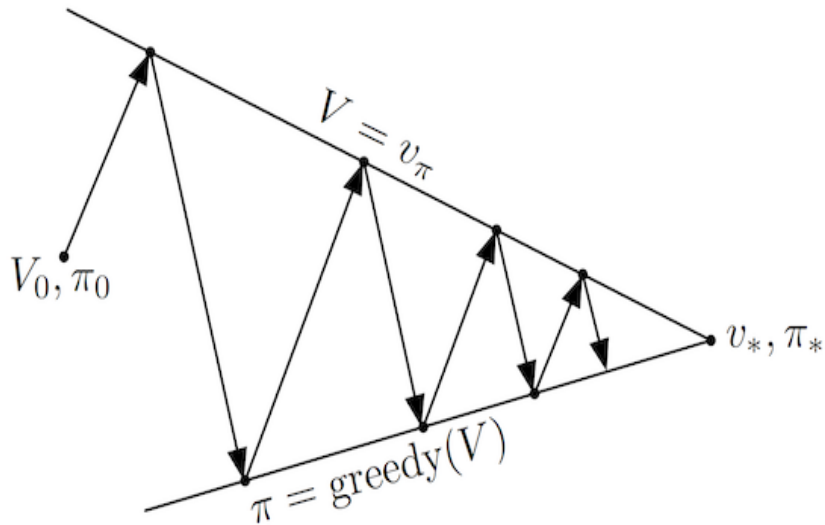
...

...

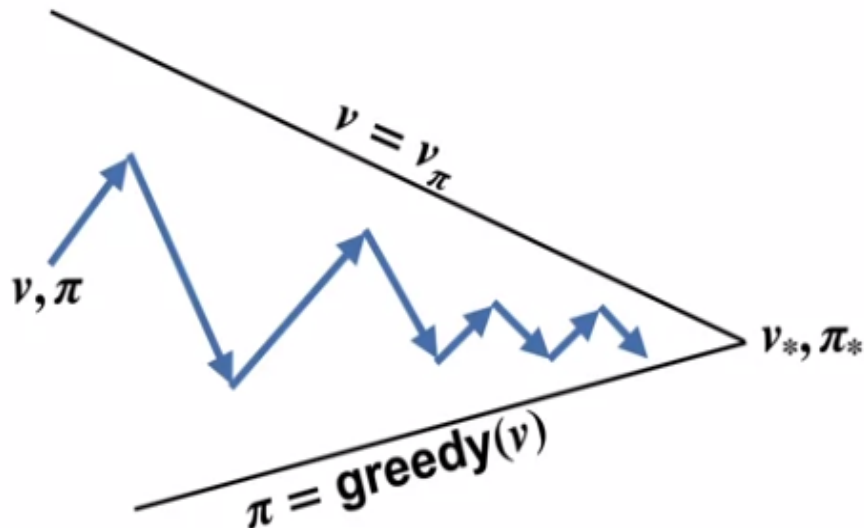
$$\pi_{j+1} = G(\mathbf{V}_j) : \mathbf{V}_j \rightarrow \mathbf{B}^{\pi_{j+1}}(\mathbf{V}_j) \rightarrow \dots (\mathbf{B}^{\pi_{j+1}})^i(\mathbf{V}_j) \rightarrow \dots \mathbf{V}^{\pi_{j+1}} = \mathbf{V}^*$$

- Policy Evaluation and Policy Improvement alternate until convergence
- In the process, they simultaneously compete and try to be consistent
- There are actually two notions of consistency:
 - VF \mathbf{V} being consistent with/close to VF \mathbf{V}^π of the policy π .
 - π being consistent with/close to Greedy Policy $G(\mathbf{V})$ of VF \mathbf{V} .

Policy Iteration



Generalized Policy Iteration (GPI)



Value Iteration and Reinforcement Learning as GPI

- Value Iteration takes only one step of Policy Evaluation

$$\pi_1 = G(\mathbf{V}_0) : \mathbf{V}_0 \rightarrow \mathbf{B}^{\pi_1}(\mathbf{V}_0) = \mathbf{V}_1$$

$$\pi_2 = G(\mathbf{V}_1) : \mathbf{V}_1 \rightarrow \mathbf{B}^{\pi_2}(\mathbf{V}_1) = \mathbf{V}_2$$

...

...

$$\pi_{j+1} = G(\mathbf{V}_j) : \mathbf{V}_j \rightarrow \mathbf{B}^{\pi_{j+1}}(\mathbf{V}_j) = \mathbf{V}^*$$

- RL updates either a subset of states or just one state at a time
- Large-scale RL updates function approximations of a VF
- These can be thought of as *partial* Policy Evaluation/Improvement

Asynchronous Dynamic Programming

The DP algorithms we've covered are qualified as *Synchronous DP*:

- All states' values are updated in each iteration
- "Simultaneous" state updates implemented by updating a copy of VF

Asynchronous DP can update subset of states, or update in any order

- ① *In-place* updates enable updated values to be used immediately
- ② *Prioritized Sweeping* keeps states sorted by their Value Function *gaps*

$$\text{Gaps } g(s) = |V(s) - \max_{a \in \mathcal{A}} \{ \mathcal{R}(s, a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot V(s') \}|$$

But this requires us to know the reverse transitions to resort queue

- ③ *Real-Time Dynamic Programming (RTDP)* runs DP *while* the agent is experiencing real-time interaction with the environment
 - A state is updated when it is visited during the real-time interaction
 - The choice of action is governed by real-time VF-extracted policy

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:
 - All random sequences terminate within fixed time steps (episodic MDP)
 - A state is encountered at most once in an episode
- This can be conceptualized as a Directed Acyclic Graph (DAG)
- Each node in the DAG is a (state, action) pair
- Prediction/Control solved by “backwards walk” from terminal nodes
- Bellman Equation enables simply *setting the VF* of a visited node
- Avoids the expensive “iterate to convergence” method of classical DP
- States visited (and VFs set) in order of reverse Topological Sort
- Next we cover a special case of DAG MDPs: finite-horizon MDPs

Finite-Horizon MDPs

- Finite-Horizon Markov Decision Processes are characterized by:
 - Each sequence terminates within a finite number of time steps T
 - Each time step has it's own state space
- Denote states at time t as \mathcal{S}_t , terminal states as \mathcal{T}_t , non-terminal states as $\mathcal{N}_t = \mathcal{S}_t - \mathcal{T}_t$ (note: $\mathcal{N}_T = \emptyset$), actions as \mathcal{A}_t , rewards as \mathcal{D}_t
- Augment each state to include time-index: augmented state is (t, s_t)

Entire MDP's States $\mathcal{S} = \{(t, s_t) | t = 0, 1, \dots, T, s_t \in \mathcal{S}_t\}$

- Each t gets its own state-reward transition probability function

$$(\mathcal{P}_R)_t : \mathcal{N}_t \times \mathcal{A}_t \times \mathcal{D}_{t+1} \times \mathcal{S}_{t+1} \rightarrow [0, 1]$$

- Likewise, each t gets its own policy $\pi_t : \mathcal{N}_t \times \mathcal{A}_t \rightarrow [0, 1]$
- An overall policy $\pi : \mathcal{N} \times \mathcal{A} \rightarrow [0, 1]$ composed of $(\pi_0, \pi_1, \dots, \pi_{T-1})$

Backward Induction for Finite MRP with Finite-Horizon

- VF for a given policy π can be represented by time-sequenced VFs

$$V_t^\pi : \mathcal{N}_t \rightarrow \mathbb{R}$$

- So Bellman Equation for π -implied Finite-Horizon MRP becomes:

$$V_t^\pi(s_t) = \sum_{s_{t+1} \in \mathcal{S}_{t+1}} \sum_{r_{t+1} \in \mathbb{D}_{t+1}} (\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^\pi(s_{t+1}))$$

$$(\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) = \sum_{a_t \in \mathcal{A}_t} \pi_t(s_t, a_t) \cdot (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1})$$

- “Backward Induction” algorithm for *finite MRP* with Finite-Horizon
- Decrementing t from $T - 1$ to 0, and calculating V_t^π from V_{t+1}^π
- Running time is $O(m^2 T)$ where $|\mathcal{N}_t|$ is $O(m)$
- $O(m^2 k T)$ to convert MDP to π -implied MRP ($|\mathcal{A}_t|$ is $O(k)$)

Backward Induction for Finite MDP with Finite-Horizon

- Optimal VF V^* can be represented by time-sequenced Optimal VFs

$$V_t^* : \mathcal{N}_t \rightarrow \mathbb{R}$$

- So MDP Bellman Optimality Equation becomes:

$$V_t^*(s_t) = \max_{a_t \in \mathcal{A}_t} \sum_{s_{t+1}} \sum_{r_{t+1}} (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^*(s_{t+1}))$$

- “Backward Induction” (Control) for *finite MDP* with Finite-Horizon
- Decrementing t from $T - 1$ to 0, and calculating V_t^* from V_{t+1}^*
- (Associated) Optimal (Deterministic) Policy $(\pi_D^*)_t : \mathcal{N}_t \rightarrow \mathcal{A}_t$ is

$$(\pi_D^*)_t(s_t) = \arg \max_{a_t \in \mathcal{A}_t} \sum_{s_{t+1}} \sum_{r_{t+1}} (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^*(s_{t+1}))$$

- Running time is $O(m^2 k T)$ where $|\mathcal{N}_t|$ is $O(m)$ and $|\mathcal{A}_t|$ is $O(k)$

@abstractclass MarkovDecisionProcess

```
def optimal_vf_and_policy(  
    steps: Sequence[StateActionMapping[S, A]],  
    gamma: float  
) -> Iterator[Tuple[  
    Mapping[NonTerminal[S], float],  
    FiniteDeterministicPolicy[S, A]  
]]  
  
def unwrap_finite_horizon_MDP(  
    process: FiniteMarkovDecisionProcess[WithTime[S], A]  
) -> Sequence[StateActionMapping[S, A]]  
  
def finite_horizon_MDP(  
    process: FiniteMarkovDecisionProcess[S, A],  
    limit: int  
) -> FiniteMarkovDecisionProcess[WithTime[S], A]
```

Dynamic Pricing for End-of-Life/End-of-Season

- Dynamic Pricing: Core to many businesses, flexing to supply/demand
- We consider special case of products being sold at end of life/season
- Assume we are T days from season-end and our inventory is M units
- Assume no more incoming inventory during these final T days
- Set prices daily to max *Expected Total Sales Revenue* over T days
- Price for a given day picked from prices $P_1, P_2, \dots, P_N \in \mathbb{R}$
- Customer daily demand is $Poisson(\lambda_i)$ if Price P_i is picked for the day
- Note that demand can exceed inventory on any day, $Sales \leq Inventory$

Dynamic Pricing Model for End-of-Life/End-of-Season

$$\mathcal{S}_t = \{(t, l_t) | l_t \in \mathbb{Z}, 0 \leq l_t \leq M\} \text{ for all } 0 \leq t \leq T$$

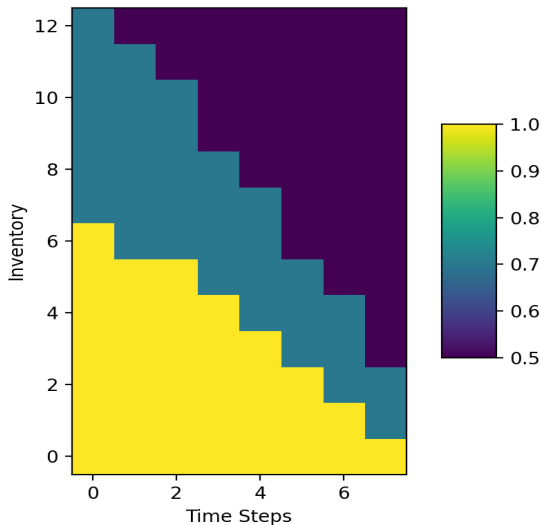
$$\mathcal{N}_t = \mathcal{S}_t \text{ and } \mathcal{A}_t = \{1, 2, \dots, N\} \text{ for all } 0 \leq t < T, \text{ and } \mathcal{N}_T = \emptyset$$

$$l_0 = M \text{ and } l_{t+1} = \max(0, l_t - d_t) \text{ where } d_t \sim \text{Poisson}(\lambda_i) \text{ if } a_t = i$$

Sales Revenue on day t is equal to $\min(l_t, d_t) \cdot P_i$

$$(\mathcal{P}_R)_t(l_t, i, r_{t+1}, l_t - k) = \begin{cases} \frac{e^{-\lambda_i} \lambda_i^k}{k!} & \text{if } k < l_t \text{ and } r_{t+1} = k \cdot P_i \\ \sum_{j=l_t}^{\infty} \frac{e^{-\lambda_i} \lambda_i^j}{j!} & \text{if } k = l_t \text{ and } r_{t+1} = k \cdot P_i \\ 0 & \text{otherwise} \end{cases}$$

Optimal Dynamic Pricing



Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as “tabular” algorithms
- “Tabular” means MDP is specified as a finite data structure
- More importantly, Value Function represented as a “table”
- These algorithms typically sweep through all states in each iteration
- Cannot do this for large finite spaces or for infinite spaces
- Requires us to generalize to function approximation of Value Function
 - Sample an appropriate subset of states
 - Calculate the Value Function for those states (Bellman calculation)
 - Create/Update a func approx with the sampled states’ calculated values
- The fundamental structure of the algorithms is still the same
- Fundamental principles (Fixed-Point/Bellman Operators) still same
- These generalizations known as *Approximate Dynamic Programming*

Key Takeaways from this Chapter

- Fixed-Point of Functions and Fixed-Point Theorem: Enables iterative algorithms to solve a variety of problems cast as Fixed-Point.
- Generalized Policy Iteration: Powerful idea of alternating between *any* method for Policy Evaluation and *any* method for Policy Improvement, including methods that are partial applications of Policy Evaluation or Policy Improvement. This generalized perspective unifies almost all of the algorithms that solve MDP Control problems.
- Backward Induction: A straightforward method to solve finite-horizon MDPs by simply walking backwards and *setting* the Value Function from the horizon-end to the start.