Lecture 3: Planning by Dynamic Programming

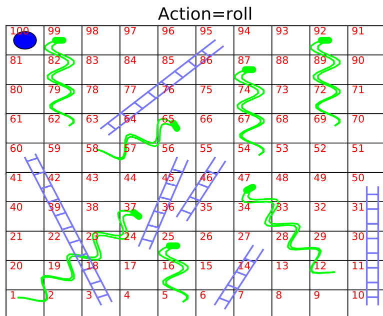
Joseph Modayil

Outline

1. Introduction
2. Policy Evaluation
3. Policy Iteration
4. Value Iteration
5. Extensions to Dynamic Programming
6. Contraction Mapping

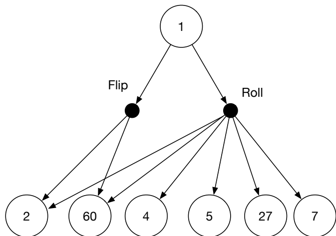
Reference: Sutton & Barto, chapter 4

Motivation: Solving an MDP



- Consider a modified game of Snakes and Ladders.
- States: Squares of the board (1=Start 100=Terminal)
- Two actions:
  - Flip a coin (move 1 or 2)
  - Roll a die (move 1–6)
- Reward is -1 per step, γ is 1
- Transitions: Move # squares, climb up ladders, slide down snakes
- What is the meaning of value here?
- What can happen from state 1?
- What action is best in each square?

Motivation: Solving an MDP (2)



100	99	98 roll (-2.9)	97 roll (-3.4)	96 roll (-4.0)	95 roll (-4.7)	94 flip (-5.3)	93 flip (-6.0)	92	91 roll (-7.7)
81 roll (-12.9)	82 roll (-12.6)	83 roll (-12.1)	84 roll (-11.5)	85 roll (-11.0)	86 roll (-11.8)	87	88 flip (-9.7)	89 roll (-9.1)	90 roll (-8.4)
80 roll (-13.0)	79 roll (-13.3)	78 roll (-13.6)	77 roll (-14.0)	76 roll (-14.3)	75 roll (-14.5)	74 roll (-14.8)	73 roll (-15.0)	72 roll (-15.5)	71 roll (-15.8)
61 flip (-9.0)	62 flip (-11.1)	63	64 flip (-15.5)	65	66 roll (-17.3)	67 roll (-17.0)	68 roll (-16.7)	69 roll (-16.3)	70 roll (-16.0)
60 flip (-11.3)	59 flip (-11.3)	58 roll (-11.6)	57 roll (-10.9)	56 roll (-12.0)	55 roll (-12.1)	54 roll (-12.6)	53 roll (-12.9)	52 roll (-12.9)	51 roll (-13.3)
41 roll (-18.7)	42 roll (-18.1)	43 roll (-16.9)	44 roll (-17.5)	45 roll (-15.7)	46 roll (-15.6)	47	48 flip (-14.1)	49 roll (-13.4)	50 roll (-13.7)
40 roll (-18.7)	39 flip (-19.7)	38 roll (-18.6)	37	36	35 flip (-14.5)	34 flip (-14.5)	33 flip (-15.6)	32 roll (-16.0)	31 roll (-15.8)
21 roll (-17.2)	22 flip (-15.2)	23 flip (-16.2)	24	25	26 roll (-17.4)	27 roll (-17.1)	28 roll (-16.8)	29 flip (-16.9)	30 roll (-15.9)
20 roll (-16.9)	19 roll (-17.1)	18 roll (-16.9)	17 roll (-17.6)	16 roll (-17.9)	15 roll (-18.2)	14 roll (-18.6)	13 roll (-18.9)	12 roll (-19.2)	11 roll (-19.5)
1 flip (-14.6)	2 flip (-15.7)	3	4 roll (-18.1)	5 roll (-18.2)	6	7 flip (-18.4)	8 flip (-16.7)	9 flip (-17.6)	10

What is Dynamic Programming?

Dynamic sequential or temporal component to the problem

Programming optimising a "program", i.e. a policy
- c.f. linear programming

- A method for solving complex problems
- By breaking them down into subproblems
  - Solve the subproblems
  - Combine solutions to subproblems

Requirements for Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructures
  - Optimal solution to a problem composed from optimal solutions to subproblems
- Overlapping subproblems
  - Subproblems recur many times
  - Solutions can be cached and reused
- Markov decision processes satisfy both properties
  - Bellman equation gives recursive decomposition
  - Value function stores and reuses solutions

Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for *planning* in an MDP
- For prediction:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
  - or: MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
  - Output: value function v^π
- Or for control:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: optimal value function v^*
  - and: optimal policy π^*

Other Applications of Dynamic Programming

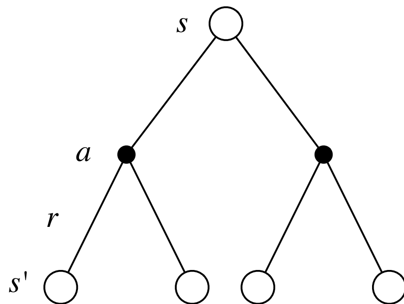
Dynamic programming is used to solve many other problems, e.g.

- Scheduling algorithms
- String algorithms (e.g. sequence alignment)
- Graph algorithms (e.g. shortest path algorithms)
- Graphical models (e.g. Viterbi algorithm)
- Bioinformatics (e.g. lattice models)

Iterative Policy Evaluation

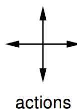
- Problem: evaluate a given fixed policy π
- Solution: iterative application of Bellman expectation backup
- $V_1 \rightarrow V_2 \rightarrow ... \rightarrow v^{\pi}$
- Using *synchronous* backups,
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $V_{k+1}(s)$ from $V_k(s')$
  - where s' is a successor state of s
- Convergence to v^{π} will be proven at the end of the lecture

Iterative Policy Evaluation (2)



$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

Small Gridworld



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

$r = -1$
on all transitions

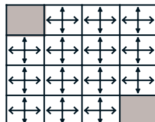
- Undiscounted episodic MDP
  - $\gamma = 1$
  - All episodes terminate in absorbing terminal state
- Nonterminal states $1, \dots, 14$
- One terminal state (shown twice as shaded squares)
- Actions that would take agent off the grid leave state unchanged
- Reward is -1 until the terminal state is reached

Iterative Policy Evaluation in Small Gridworld

V_k for the Random Policy | Greedy Policy w.r.t. V_k

$k = 0$

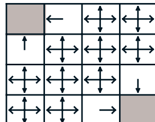
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

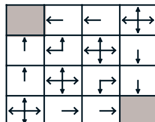
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

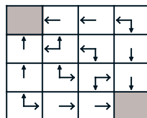
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Iterative Policy Evaluation in Small Gridworld (2)

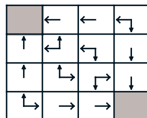
$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



$k = 10$

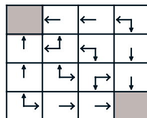
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



optimal policy

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can *improve* the policy by acting greedily

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\ q^{\pi}(s, a)$$

- This improves the value from any state s over one step,

$$q^{\pi}(s, \pi'(s)) = \max_{a \in \mathcal{A}} q^{\pi}(s, a) \geq q^{\pi}(s, \pi(s)) = v^{\pi}(s)$$

- It therefore improves the value function, $v^{\pi'}(s) \geq v^{\pi}(s)$

$$\begin{aligned} v^{\pi}(s) \leq q^{\pi}(s, \pi'(s)) &= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v^{\pi}(S_{t+1}) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q^{\pi}(A_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q^{\pi}(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v^{\pi'}(s) \end{aligned}$$

Policy Improvement (2)

- If improvements stop,

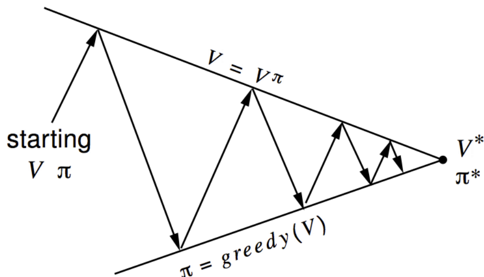
$$q^{\pi}(s, \pi'(s)) = \max_{a \in \mathcal{A}} q^{\pi}(s, a) = q^{\pi}(s, \pi(s)) = v^{\pi}(s)$$

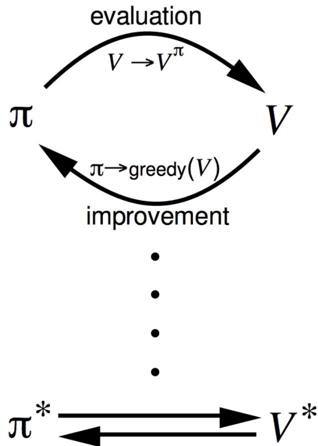
- Then the Bellman optimality equation has been satisfied

$$v^{\pi}(s) = \max_{a \in \mathcal{A}} q^{\pi}(s, a)$$

- Therefore $v^{\pi}(s) = v^{*}(s)$ for all $s \in \mathcal{S}$
- so π is an optimal policy

Policy Iteration



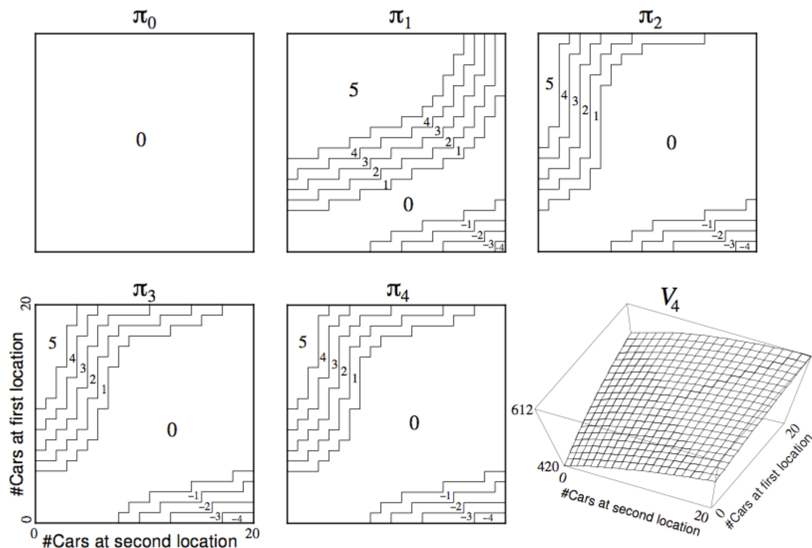


- Policy evaluation Estimate v^{π}
  - Iterative policy evaluation
- Policy improvement Generate $\pi' \geq \pi$
  - Greedy policy improvement

Jack's Car Rental

- States: Two locations, maximum of 20 cars at each
- Actions: Move up to 5 cars overnight (-$2 each)
- Reward: $10 for each car rented (must be available), $\gamma = 0.9$
- Transitions: Cars returned and requested randomly
  - Poisson distribution, n returns/requests with prob $\frac{\lambda^n}{n!}e^{-\lambda}$
  - 1st location: average requests = 3, average returns = 3
  - 2nd location: average requests = 4, average returns = 2

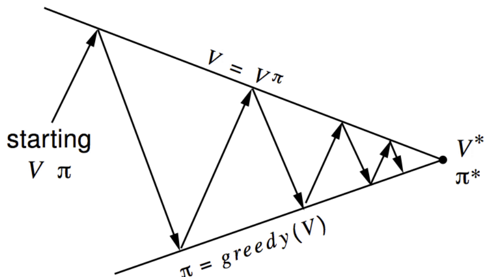
Policy Iteration in Jack's Car Rental

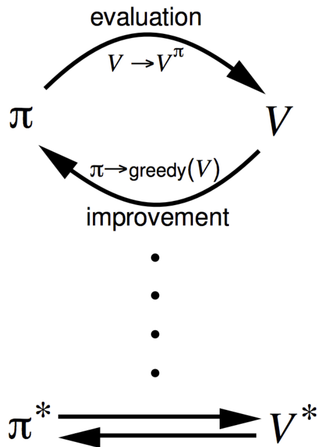


Modified Policy Iteration

- Does policy evaluation need to converge to v^π?
- Or should we introduce a stopping condition
  - e.g. ϵ-convergence of value function
- Or simply stop after k iterations of iterative policy evaluation?
- For example, in the small gridworld $k = 3$ was sufficient to achieve optimal policy
- Why not update policy every iteration? i.e. stop after $k = 1$
  - This is equivalent to *value iteration* (next section)

Generalised Policy Iteration





Policy evaluation Estimate v^π

Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm

Deterministic Value Iteration

- If we know the solution to subproblems $v_*(s')$
- Then it is easy to construct the solution to $v_*(s)$

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + v_*(s')$$

- The idea of value iteration is to apply these updates iteratively
- e.g. Starting at the goal (horizon) and working backwards

Example: Shortest Path

Problem

g			

V_1

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_2

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_3

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_4

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_7

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

Value Iteration in MDPs

- MDPs don't usually have a finite horizon
- They are typically loopy
- So there is no "end" to work backwards from
- However, we can still propagate information backwards
- Using Bellman optimality equation to *backup* $V(s)$ from $V(s')$
- Each subproblem is "easier" due to discount factor γ
- Iterate until convergence

Optimality in MDPs

An optimal policy π^* must provide both

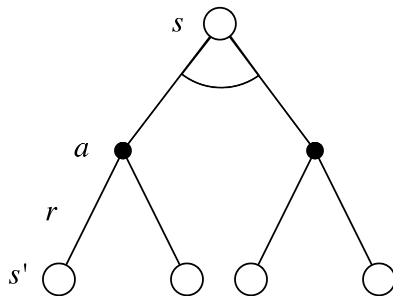
- An optimal first action a^* from any state s,
- Followed by an optimal policy from successor state s'

$$v^*(s) = \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v^*(s')$$

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $V_1 \rightarrow V_2 \rightarrow ... \rightarrow v^*$
- Using synchronous backups
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $V_{k+1}(s)$ from $V_k(s')$
- Convergence to v^* will be proven later
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

Value Iteration (2)



$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

Synchronous Dynamic Programming Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s,a)$ or $q_*(s,a)$
- Complexity $O(m^2n^2)$ per iteration

Asynchronous Dynamic Programming

- DP methods described so far used *synchronous* backups
- i.e. all states are backed up in parallel
- *Asynchronous DP* backs up states individually, in any order
- For each selected state, apply the appropriate backup
- Can significantly reduce computation
- Guaranteed to converge if all states continue to be selected

Asynchronous Dynamic Programming

Three simple ideas for asynchronous dynamic programming:

- *In-place* dynamic programming
- *Prioritised sweeping*
- *Real-time* dynamic programming

In-Place Dynamic Programming

- Synchronous value iteration stores two copies of value function

  for all s in $\mathcal{S}$

$$V_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_{old}(s') \right)$$

$$V_{old} \leftarrow V_{new}$$

- In-place value iteration only stores one copy of value function

  for all s in $\mathcal{S}$

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V(s') \right)$$

Prioritised Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.

$$\left| \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right) - V_k(s) \right|$$

- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue

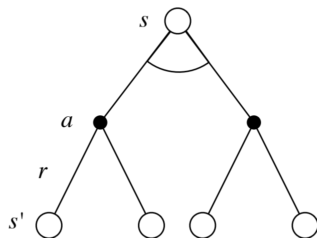
Real-Time Dynamic Programming

- Idea: only states that are relevant to agent
- Use agent's experience to guide the selection of states
- After each time-step S_t, A_t, R_{t+1}
- Backup the state S_t

$$V(S_t) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}^a_{S_t} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{S_t s'} V(s') \right)$$

Full-Width Backups

- DP uses *full-width* backups
- For each backup (sync or async)
  - Every successor state and action is considered
  - Using knowledge of the MDP transitions and reward function
- DP is effective for medium-sized problems (millions of states)
- For large problems DP suffers Bellman's *curse of dimensionality*
  - Number of states $n = |\mathcal{S}|$ grows exponentially with number of state variables
- Even one backup can be too expensive



Approximate Dynamic Programming

- Approximate the value function
- Using a *function approximator* $V^{\theta}(s) = v(s; \theta)$, with a parameter vector $\theta \in \mathbb{R}^n$.
- The estimated value function at iteration k is $V_k = V^{\theta_k}$
- Use dynamic programming to compute $V^{\theta_{k+1}}$ from V^{θ_k}.
- e.g. Fitted Value Iteration repeats at each iteration k,
  - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
  - For each sample state $s \in \tilde{\mathcal{S}}$, compute target value using Bellman optimality equation,

$$\tilde{V}_k(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V^{\theta_k}(s') \right)$$

  - Train next value function $V^{\theta_{k+1}}$ using targets $\left\{ \langle s, \tilde{V}_k(s) \rangle \right\}$

Some Technical Questions

- How do we know that value iteration converges to v^*?
- Or that iterative policy evaluation converges to v^π?
- And therefore that policy iteration converges to v^*?
- Is the solution unique?
- How fast do these algorithms converge?
- These questions are resolved by *contraction mapping theorem*

Value Function Space

- *Consider the vector space $\mathcal{V}$ over value functions*
- There are $|\mathcal{S}|$ dimensions
- Each point in this space fully specifies a function $V(s)$
- What does a Bellman backup do to points in this space?
- We will show that it brings value functions *closer*
- And therefore the backups must converge on a unique solution

Value Function ∞-Norm

- We will measure distance between state-value functions U and V by the ∞-norm
- i.e. the largest difference between state values,

$$||U - V||_\infty = \max_{s \in \mathcal{S}} |U(s) - V(s)|$$

- Define the *Bellman expectation backup operator* T^π,

$$T^\pi(V) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi V$$

where $\mathcal{R}^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}_s^a$
and $(\mathcal{P}^\pi V)(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V(s')$.

Bellman Expectation Backup is a Contraction

- The Bellman expectation backup operator is a γ-contraction, i.e. it makes value functions closer by at least γ,

$$\begin{aligned}
||T^\pi U - T^\pi V||_\infty &= \max_s |(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi U)(s) - (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi V)(s))| \\
&= \max_s |\gamma \textstyle\sum_a \pi(a|s) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} (U(s') - V(s'))| \\
&\leq \max_s \gamma \textstyle\sum_a \pi(a|s) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} |U(s') - V(s')| \\
&\leq \max_s \gamma \textstyle\sum_a \pi(a|s) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} ||U - V||_\infty \\
&\leq \gamma ||U - V||_\infty (\max_s \textstyle\sum_a \pi(a|s) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}) \\
&\leq \gamma ||U - V||_\infty
\end{aligned}$$

Contraction Mapping Theorem

Theorem (Contraction Mapping Theorem)

For any metric space $\mathcal{V}$ that is complete (i.e. contains its limit points) under an operator $T(V)$, where T is a γ-contraction,

- *T converges to a unique fixed point*
- *At a linear convergence rate of γ*

Convergence of Iter. Policy Evaluation and Policy Iteration

- The Bellman expectation operator T^π has a unique fixed point
- v^π is a fixed point of T^π (by Bellman expectation equation)
- By contraction mapping theorem
- Iterative policy evaluation converges on v^π
- Policy iteration converges on v^*

Bellman Optimality Backup is a Contraction

- Define the *Bellman optimality backup operator* T^*,

$$T^*(V) = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a V$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ (similar to previous proof)

$$||T^*(U) - T^*(V)||_\infty \leq \gamma ||U - V||_\infty$$

Convergence of Value Iteration

- The Bellman optimality operator T^* has a unique fixed point
- v^* is a fixed point of T^* (by Bellman optimality equation)
- By contraction mapping theorem
- Value iteration converges on v^*