Carnegie Mellon
School of Computer Science

Markov Decision Processes (2)

Lecture 4, CMU 10-403

Katerina Fragkiadaki

Used Materials

- **Disclaimer**: Some material and slides for this lecture were borrowed from Rich Sutton's class and David Silver's class on Reinforcement Learning.

Contraction Mapping Theorem

An operator F on a normed vector space $\mathcal{X}$ is a γ**-contraction**, for $0 < \gamma < 1$, provided for all $x, y \in \mathcal{X}$

$$||T(x) - T(y)|| \leq \gamma ||x - y||$$

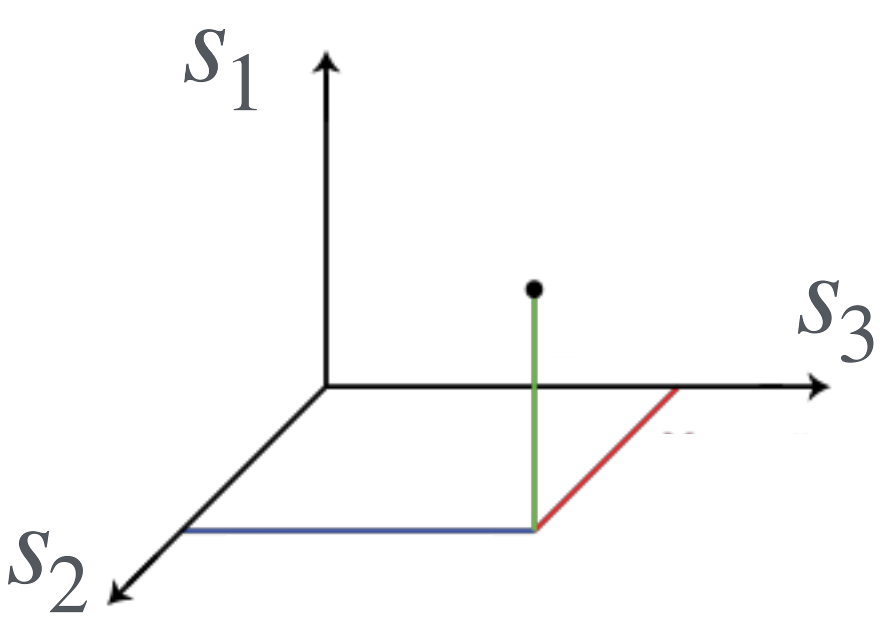
Theorem (Contraction mapping)
For a γ-contraction F in a complete normed vector space $\mathcal{X}$

- F converges to a unique fixed point in $\mathcal{X}$
- at a linear convergence rate γ

Remark. In general we only need metric (vs normed) space

Value Function Sapce

- Consider the vector space V over value functions
- There are $|\mathcal{S}|$ dimensions
- Each point in this space fully specifies a value function $\mathrm{v}(s)$
- Bellman backup brings value functions closer in this space
- And therefore the backup must converge to 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,

$$||\mathrm{u} - \mathrm{v}||_\infty = \max_{s \in \mathcal{S}} |\mathrm{u}(s) - \mathrm{v}(s)|$$

$$||u||_\infty = \max_{s \in \mathcal{S}} |u(s)|$$

Bellman Expectation Backup is a Contraction

- Define the Bellman expectation backup operator

$$F^{\pi}(\mathrm{v}) = r^{\pi} + \gamma T^{\pi}\mathrm{v}$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ,

$$\begin{aligned}
\|F^{\pi}(u) - F^{\pi}(v)\|_{\infty} &= \|(r^{\pi} + \gamma T^{\pi}u) - (r^{\pi} + \gamma T^{\pi}v)\|_{\infty} \\
&= \|\gamma T^{\pi}(u - v)\|_{\infty} \\
&\leq \|\gamma T^{\pi}(\mathbf{1}\|(u - v)\|_{\infty})\|_{\infty} \\
&= \|\gamma(T^{\pi}\mathbf{1})\|u - v\|_{\infty}\|_{\infty} \\
&= \|\gamma\mathbf{1}\|u - v\|_{\infty}\|_{\infty} \\
&= \gamma\|u - v\|_{\infty}
\end{aligned}$$

Convergence of Iter. Policy Evaluation and Policy Iteration

- The Bellman expectation operator F^{π} has a unique fixed point
- v_{π} is a fixed point of F^{π} (by Bellman expectation equation)
- By contraction mapping theorem
- Iterative policy evaluation converges to v_{π}

Policy Improvement

- Suppose we have computed v_π for a deterministic policy π
- For a given state s, would it be better to do an action $a \neq \pi(s)$?
- It is better to switch to action a for state s if and only if

$$q_\pi(s, a) > \mathrm{v}_\pi(s)$$

- And we can compute $q_\pi(s, a)$ from v_π by:

$$q_\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) v_\pi(s')$$

Policy Improvement Cont.

- Do this for all states to get a new policy $\pi' \geq \pi$ that is greedy with respect to v_π:

$$
\begin{aligned}
\pi'(s) &= \arg\max_a q_\pi(s,a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma \mathrm{v}_\pi(s') | S_t = s, A_t = a] \\
&= \arg\max r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a)\, \mathrm{v}_\pi(s')
\end{aligned}
$$

Policy Improvement Cont.

- Do this for all states to get a new policy $\pi' \geq \pi$ that is greedy with respect to v_π:

$$\begin{aligned}\pi'(s) &= \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma \mathrm{v}_\pi(s') | S_t = s, A_t = a] \\ &= \arg\max r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a)\, \mathrm{v}_\pi(s')\end{aligned}$$

- After policy update it holds that:

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s,a)) \\ &= \max_a q_{\pi_k}(s,a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s). \end{aligned}$$

$$\mathrm{v}_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

Policy Improvement Cont.

- After policy update it holds that:

$$v_{\pi_k}(s) \leq q_{\pi_k}(s, \pi_{k+1}(s))$$

- We have indeed improved the policy (or ended up on an equally good policy)

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \bigm| S_t = s\big] \\
&\leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \bigm| S_t = s\big] \\
&\;\;\vdots \\
&\leq \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \bigm| S_t = s\big] \\
&= v_{\pi'}(s).
\end{aligned}$$

Policy Improvement Cont.

- If policy is unchanged after the greedification step, this means that:

$$\mathrm{v}_\pi(s) = \max_{a \in \mathscr{A}} \left(r(s,a) + \gamma \sum_{s'} \mathrm{p}(s' \,|\, s,a) v_\pi(s') \right)$$

$$\mathrm{v}_\pi(s) = \max_{a \in \mathscr{A}} q_\pi(s,a)$$

- But this is the Bellman optimality Equation. So v_pi=v* and \pi is optimal

Policy Iteration

$$\pi_0 \xrightarrow{\text{E}} \mathrm{v}_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} \mathrm{v}_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} \mathrm{v}_*$$

policy evaluation

policy improvement
"greedification"

Policy Iteration

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation *(Till convergence)*
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $\text{v} \leftarrow V(s)$
   $V(s) \leftarrow \Sigma_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \Sigma_{s' \in \mathcal{S}} p(s'|s,a) V(s')\right)$
   $\Delta \leftarrow \max(\Delta, |\text{v} - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

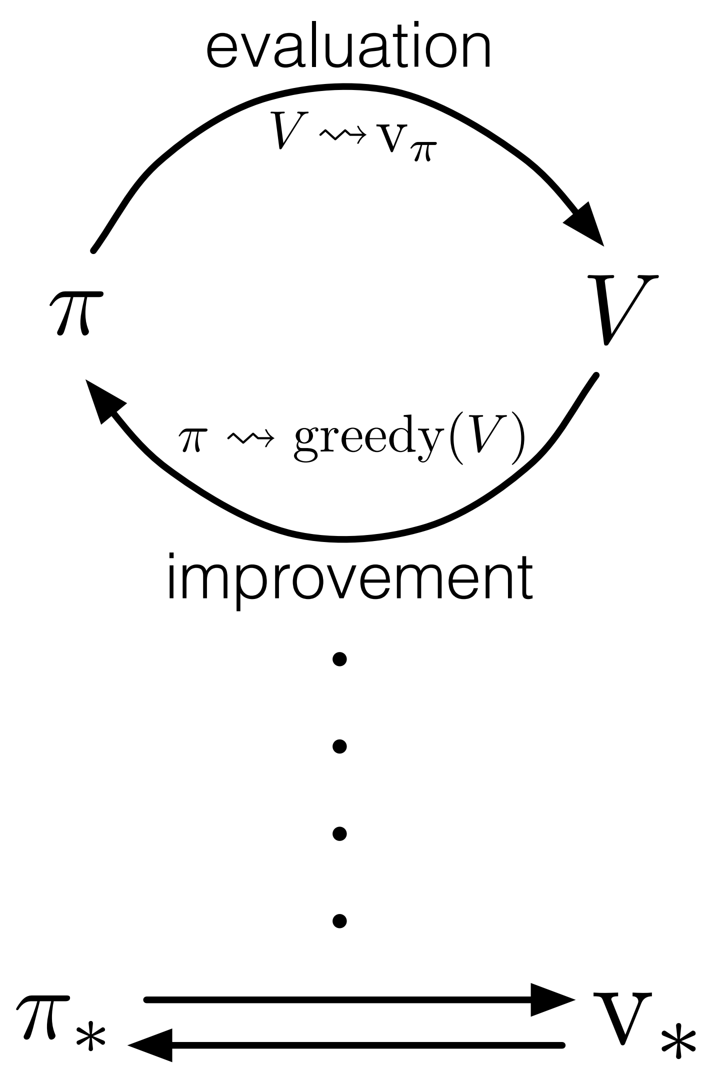
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $a \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max r(s,a) + \gamma \Sigma_{s' \in \mathcal{S}} p(s'|s,a) \text{v}_\pi(s')$
   If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

Generalized Policy Iteration

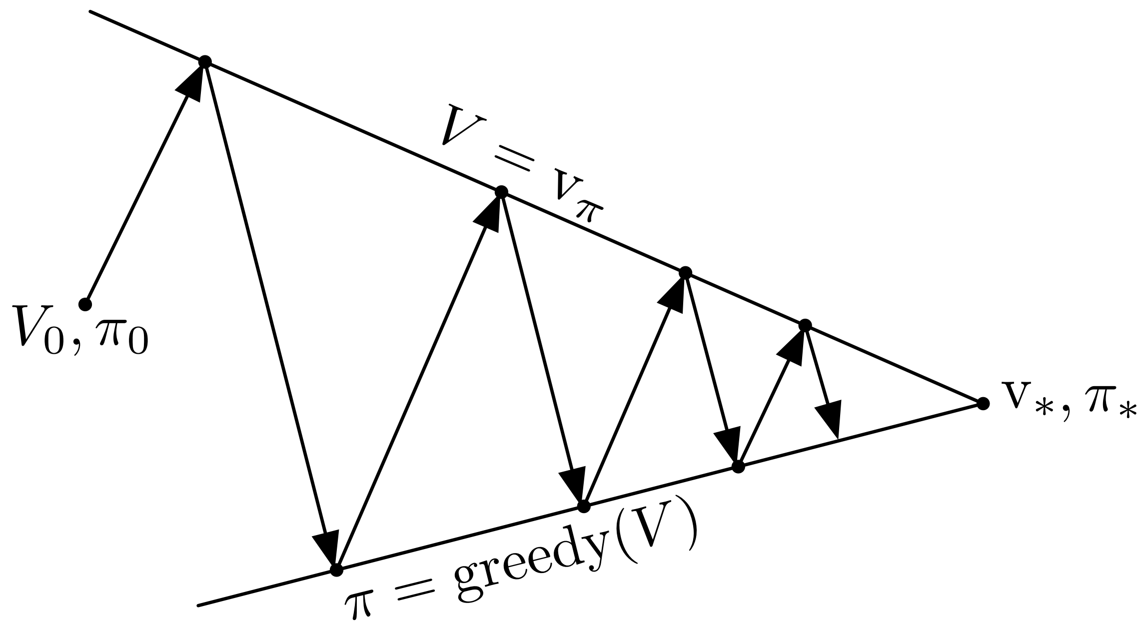
- Does policy evaluation need to converge to v_π ?
- Or should we introduce a stopping condition
  - e.g. ϵ-convergence of value function
- Or simply stop after ***k*** iterations of iterative policy evaluation?
- For example, in the small grid world ***k*** = 3 was sufficient to achieve optimal policy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

Generalized Policy Iteration

Generalized Policy Iteration (GPI): any interleaving of policy evaluation and policy improvement, independent of their granularity.



A geometric metaphor for convergence of GPI:



Principle of Optimality

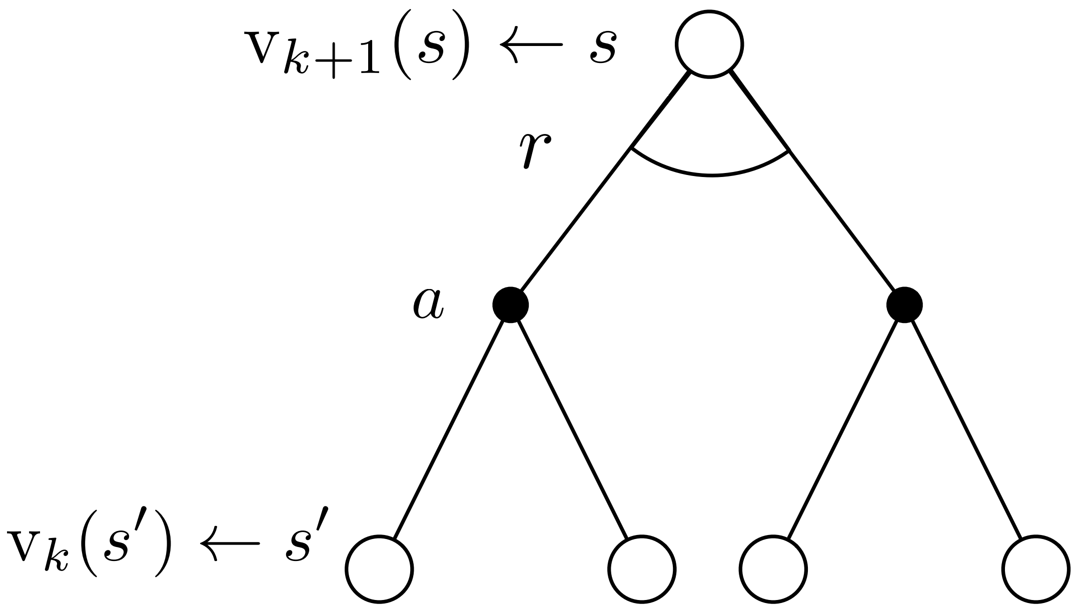
- Any optimal policy can be subdivided into two components:
  - An optimal first action $\mathcal{A}_*$
  - Followed by an optimal policy from successor state $\mathcal{S}'$

- Theorem (Principle of Optimality)
  - A policy $\pi(a|s)$ achieves the optimal value from state s, $\mathrm{v}_\pi(s) = \mathrm{v}_*(s)$, if and only if
  - For any state s' reachable from s, π achieves the optimal value from state s', $\mathrm{v}_\pi(s') = \mathrm{v}_*(s')$

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $\mathrm{v}_1 \rightarrow \mathrm{v}_2 \rightarrow \ldots \rightarrow \mathrm{v}_*$
- Using synchronous backups
  - At each iteration ***k*** + 1
  - For all states $s \in \mathcal{S}$
  - Update $\mathrm{v}_{k+1}(s)$ from $\mathrm{v}_k(s')$

Value Iteration (2)



$$v_{[k+1]}(s) = \max_{a \in \mathscr{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathscr{S}} p(s' \mid s,a) v_{[k]}(s') \right), \forall s$$

$$\mathrm{v}_{k+1} = \max_{a \in \mathcal{A}} r(a) + \gamma p(a) v_k$$

Bellman Optimality Backup is a Contraction

- Define the Bellman optimality backup operator F^*,

$$F^*(\mathrm{v}) = \max_{a \in \mathcal{A}} r(a) + \gamma p(a) v$$

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

$$||F^*(\mathrm{u}) - F^*(\mathrm{v})||_\infty \leq \gamma ||\mathrm{u} - \mathrm{v}||_\infty$$

Convergence of Value Iteration

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

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

Summary so far

- We are investigating finite MDPs: finite sets of actions and states
- We explained why value functions are important
- We discussed two ways to compute optimal policies: policy iteration and value iteration
- We saw that value iteration and policy evaluation converge to v* and v_pi and that policy iteration converges to the optimal policy and optimal value function (\pi*,v*)
- We have understood that exhaustive state sweeps (synchronous dynamic programming) are hopeless…

  Can we change that?

Efficiency of DP

- To find an optimal policy is polynomial in the number of states…
- BUT, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called “the curse of dimensionality”).
- In practice, classical DP can be applied to problems with a few millions of states.

Asynchronous DP

- All the DP methods described so far require exhaustive sweeps of the entire state set.
- Asynchronous DP does not use sweeps. Instead it works like this:
  - Repeat until convergence criterion is met:
    - Sample a state at random and apply the appropriate backup
- Still need lots of computation, but does not get locked into hopelessly long sweeps
- Guaranteed to converge if *all* states continue to be selected

Asynchronous Dynamic Programming

- Three simple ideas for asynchronous dynamic programming:
  - In-place dynamic programming
  - Prioritized sweeping
  - Real-time dynamic programming

In-Place Dynamic Programming

- Synchronous value iteration stores two copies of value function
  - for all s in $\mathcal{S}$

$$\mathrm{v}_{new}(s) \leftarrow \max_{a\in\mathcal{A}} \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p\left(s'|s,a\right)\mathrm{v}_{old}(s') \right)$$

$$\mathrm{v}_{old} \leftarrow \mathrm{v}_{new}$$

- In-place value iteration only stores one copy of value function
  - for all s in $\mathcal{S}$

$$\mathrm{v}(s) \leftarrow \max_{a\in\mathcal{A}} \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p\left(s'|s,a\right)\mathrm{v}(s') \right)$$

Prioritized Sweeping

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

$$\left| \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \mathrm{v}(s') \right) - \mathrm{v}(s) \right|$$

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

Example: Shortest Path

g			

Problem

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

V_1

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

V_2

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

V_3

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

V_4

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

V_5

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

V_6

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

V_7

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 $\mathcal{S}_t, \mathcal{A}_t, r_{t+1}$
- Backup the state $\mathcal{S}_t$

$$\mathrm{v}(\mathcal{S}_t) \leftarrow \max_{a \in \mathcal{A}} \left(r(\mathcal{S}_t, a) + \gamma \sum_{s' \in \mathcal{S}} p\,(s' | \mathcal{S}_t, a) \mathrm{v}(s') \right)$$