

Dynamic Programming

Mattia Pellegrino, Ph.D Fellow
mattia.pellegrino@unipr.it



Summary

- Policy Evaluation and Iteration
- Value Iteration
- Dynamic programming

DYNAMIC PROGRAMMING

- A process that aim to optimize a program (i.e. policy) exploiting the sequential or the temporal component of the problem
 - A method for solving complex problems
 - Breaking them into subproblems
 - Subproblems are usually easier to solve
 - Once solved the solutions is just a combination of the subproblems' solutions

DYNAMIC PROGRAMMING

- Dynamic Programming is a very general solution method.
- Problems must have some properties:
 - Optimal structure
 - Optimality principle
 - Optimal solution can be decomposed into subproblems
 - Overlapping subproblems
 - Subproblems recur many times
 - Solutions can be kept and reused
 - Markov decision processes satisfy both properties
 - Bellman equation gives recursive decomposition
 - Value function can be stored and reused

DYNAMIC PROGRAMMING

- We assume the full knowledge of the MDP
- It is used for planning in an MPD

- For prediction

$input = MDP \langle S, A, P, R, \gamma \rangle \text{ and } \pi$

$input = MRP \langle S, P^\pi, R^\pi, \gamma \rangle$

$output = v_\pi$

- For control

$input = MDP \langle S, A, P, R, \gamma \rangle$

$output = v_* \text{ and } \pi_*$

DYNAMIC PROGRAMMING

- It is used to solve many other problems:
 - Scheduling
 - String algorithms
 - Graph algorithms
 - Graphical Models
 - Bioinformatics

ITERATIVE POLICY EVALUATION

- **Problems:**

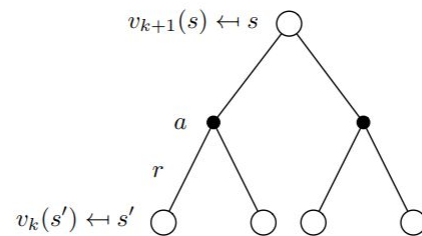
Evaluate a policy π

- **Solution:**

Iterative application of Bellman expectation backup

- 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

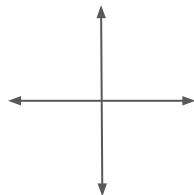


$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$
$$\mathbf{v}^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}^k$$

RANDOM POLICY IN A SMALL WORLD

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

Actions



Properties

$r = -1$ on every action

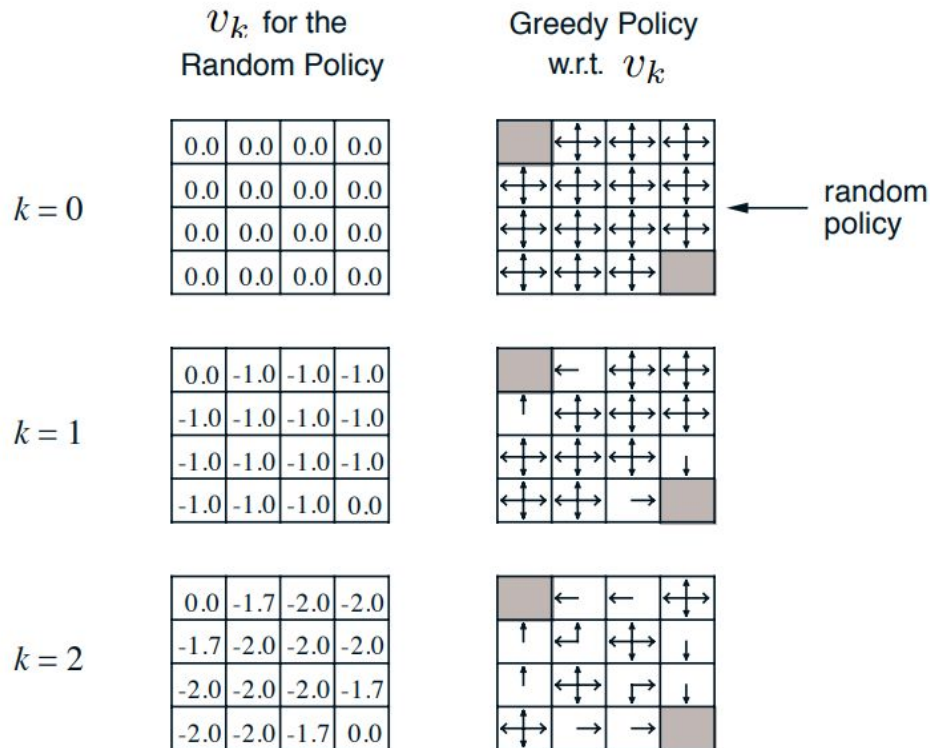
$\gamma = 1$ undiscounted

$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$

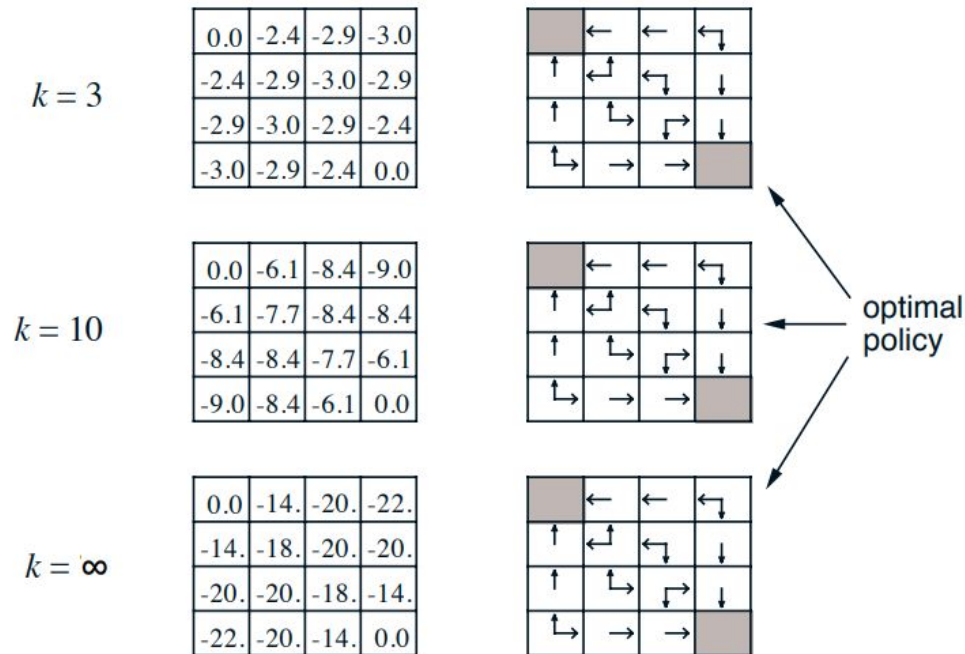
Terminal states

Agent follows an uniform random policy

RANDOM POLICY IN A SMALL WORLD



RANDOM POLICY IN A SMALL WORLD



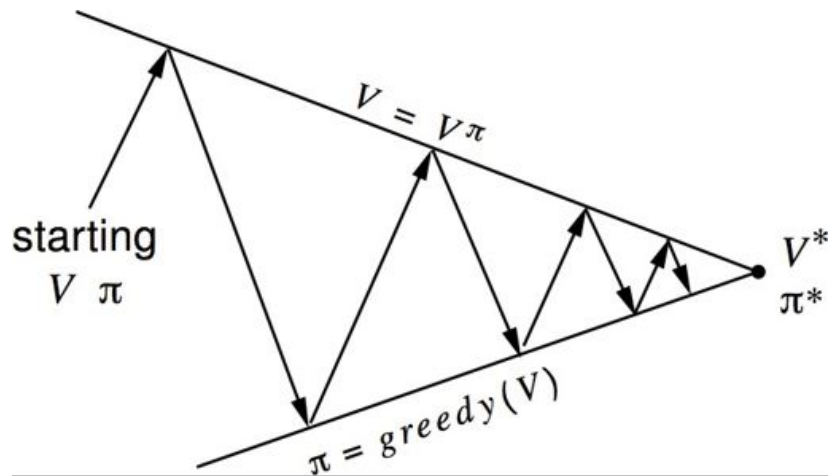
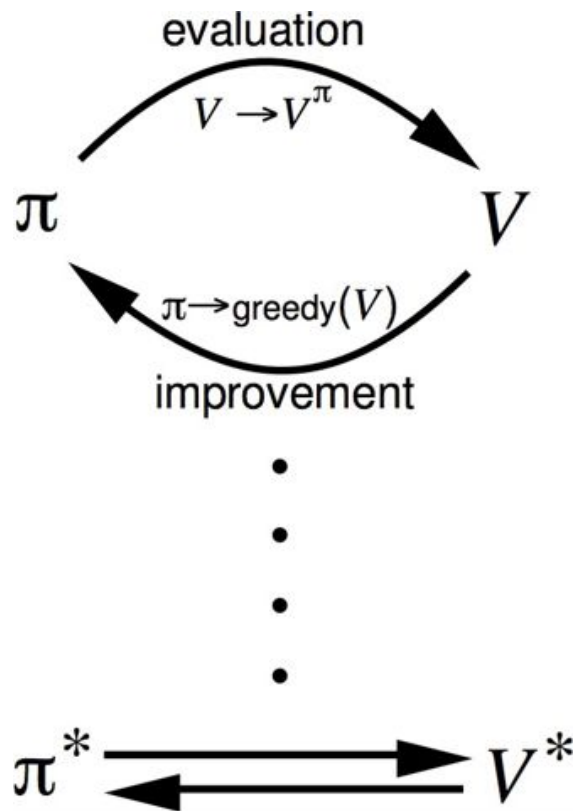
POLICY IMPROVEMENT

- Give a policy π
 - We want to evaluate: $v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} | S_t = s]$
 - We want to improve by acting greedily

$$\pi' = greedy(v_\pi)$$

- In a small gridworld improved policy was optimal, $\pi^1 = \pi^*$
- We need more iterations of improvement, in this way we can converge to π^*

POLICY ITERATION



Policy evaluation

→ Estimate v_π

Policy improvement

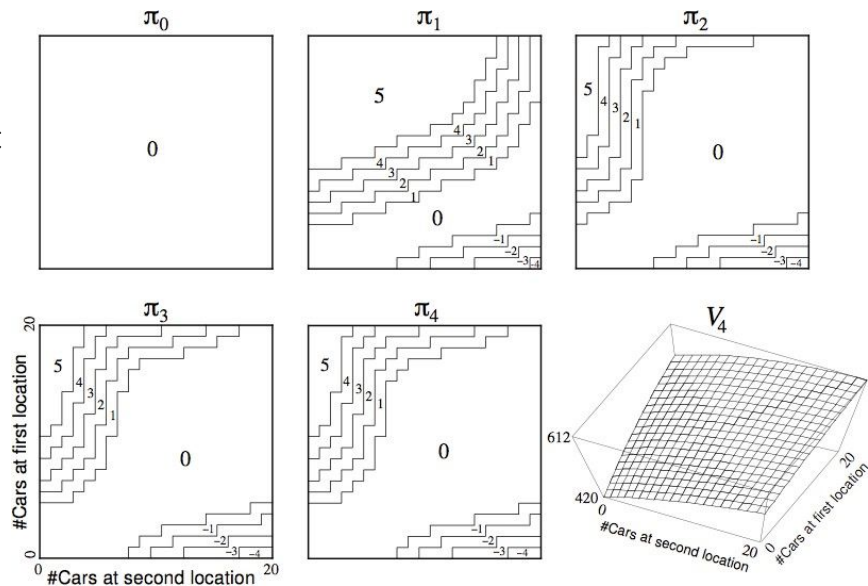
→ Generate $\pi' \geq \pi$

EXAMPLE - JACK'S CAR RENTAL (from sutton)

- The Jack's Car Rental problem is a classic reinforcement learning problem described by Richard Sutton in his book, "Reinforcement Learning: An Introduction."
- The problem involves managing two car rental locations to maximize profit.
- Each location has a limited number of cars that can be rented out or returned each day.
- The number of cars requested and returned at each location is a random variable.
- The goal is to find the optimal policy for transferring cars between the two locations to maximize profit.

EXAMPLE - JACK'S CAR RENTAL (from Sutton)

- States: Two locations, maximum of 20 cars at each
- Actions: Move up to 5 cars between locations overnight
- Reward: \$10 for each car rented (must be available)
- Transitions: Cars returned and requested randomly



POLICY IMPROVEMENT

- Consider a deterministic policy: $a = \pi(s)$
- We can improve this policy with a greedily method:

$$\pi'(s) = \operatorname{argmax}_{a \in A} q_{\pi}(s, a)$$

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

$$q_{\pi}(s, \pi'(s)) = \max_{a \in A} q_{\pi}(s, a) \geq q_{\pi}(s, \pi(s)) = v_{\pi}(s)$$

- This will improve the value function: $v_{\pi'}(s) \geq v_{\pi}(s)$

$$\begin{aligned} v_{\pi}(s) &\leq q_{\pi}(s, \pi'(s)) = \mathbb{E}_{\pi'} [R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma q_{\pi}(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \gamma^2 q_{\pi}(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s] = v_{\pi'}(s) \end{aligned}$$

POLICY IMPROVEMENT

- If the improvement stops,

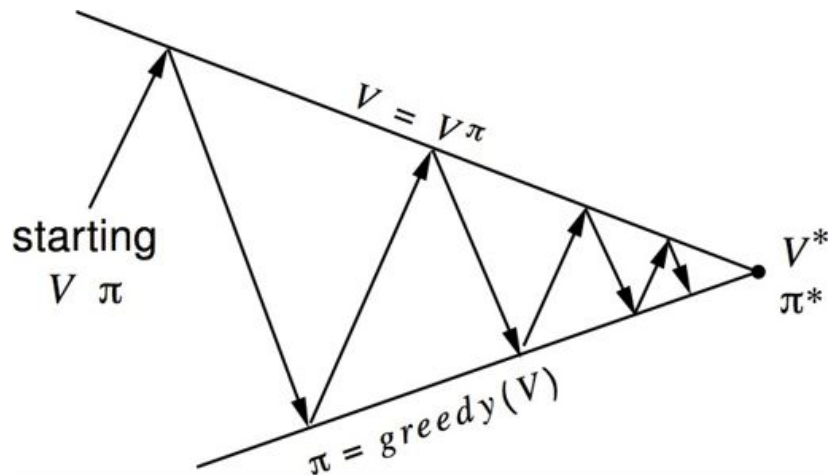
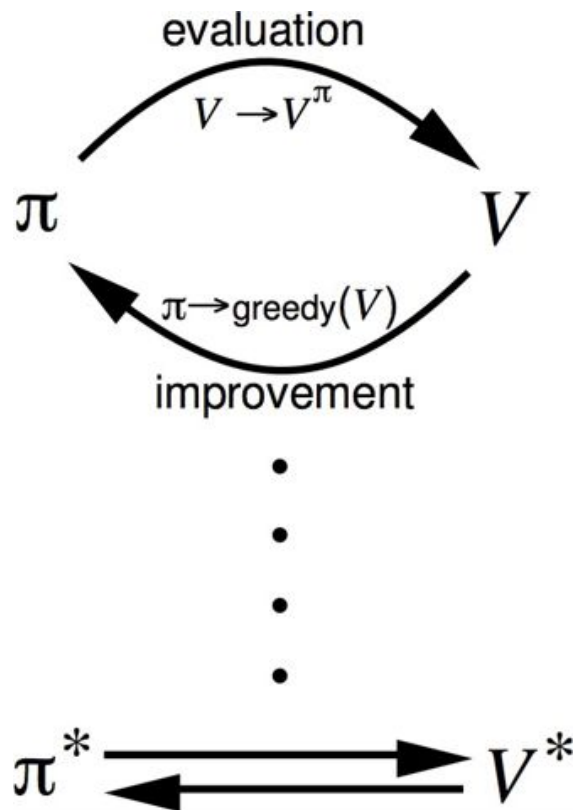
$$q_{\pi}(s, \pi'(s)) = \max_{a \in A} q_{\pi}(s, a) = q_{\pi}(s, \pi(s)) = v_{\pi}(s)$$

- The Bellman optimality equation is then satisfied

$$v_{\pi} = \max_{a \in A} q_{\pi}(s, a)$$

- Hence $v_{\pi}(s) = v_{*}(s) \forall s \in S \leftarrow \pi$ optimal policy

POLICY ITERATION



Any Policy evaluation

→ Estimate v_π

Any Policy improvement

→ Generate $\pi' \geq \pi$

VALUE ITERATION – PRINCIPLE OF OPTIMALITY

- Any optimal policy can be divided into two parts:
 - An optimal first action A_*
 - An optimal policy from successor state S'
- **A policy $\pi(a|s)$ achieves optimal value from state s , $v_\pi = v_*(s)$ if and only if**
 - For any state s' reachable from s , π achieve the optimal value from state s' , $v_\pi(s') = v_*(s')$

DETERMINISTIC VALUE ITERATION

- If we know the solution of a generic subproblem $v_*(s')$
- Then the solutions $v_*(s)$ can be found using the formula:

$$v_* \leftarrow \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

- The idea is to apply these updates iteratively, start with final rewards and work backwards

EXAMPLE - VALUE ITERATION

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

VALUE ITERATION

- **Problems:**

Find optimal policy π

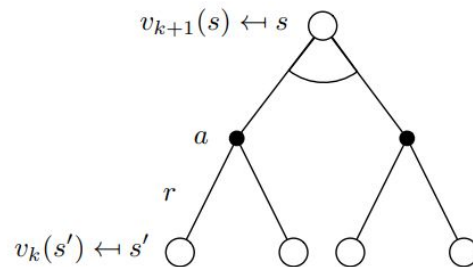
- **Solution:**

Iterative application of Bellman optimality backup

- Synchronous backups:

- At each iteration $k + 1$
- For all states $s \in \mathcal{S}$
- Update $v_{k+1}(s)$ from $v_k(s')$

- There is no policy here



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

SYNCHRONOUS DYNAMIC PROGRAMMING

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

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
- 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 S

$$v_{new}(s) \leftarrow \max_{a \in A} (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{old}(s'))$$

- In-place value iteration only stores one copy of value for all s in S

$$v(s) \leftarrow \max_{a \in A} (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v(s'))$$

PRIORITISED SWEEPING

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

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

- 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

PRIORITISED SWEEPING

- 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

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

EXERCISE: JACK'S CAR RENTAL

Jack's Car Rental Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited \$10 by the national company. If he is out of cars at that location, then the business is lost. Cars become available for renting the day after they are returned. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of \$2 per car moved. We assume that the number of cars requested and returned at each location are Poisson random variables, meaning that the probability that the number is n is $\lambda^n * e^{(-\lambda)} / n!$, where λ is the expected number. Suppose λ is 3 and 4 for rental requests at the first and second locations and 3 and 2 for returns. To simplify the problem slightly, we assume that there can be no more than 20 cars at each location (any additional cars are returned to the nationwide company, and thus disappear from the problem) and a maximum of five cars can be moved from one location to the other in one night. We take the discount rate to be $\gamma = 0.9$ and formulate this as a continuing finite MDP, where the time steps are days, the state is the number of cars at each location at the end of the day, and the actions are the net numbers of cars moved between the two locations overnight.

What is the optimal transfer policy of cars between branches?

EXERCISE: JACK'S CAR RENTAL

- Modify the source code available at the following address:
https://github.com/sowide/reinforcement_learning_course/16-05_exercise
- Deliver the modified source code: <https://shorturl.at/elBIM> [name_surname.zip]