Solving the reinforcement learning problem with dynamic programming

Reinforcement Learning: An Introduction
by Richard S. Sutton and Andrew G. Barto

chapter:4

Presented by Vibhor kumar

CONTENTS

Dynamic programming

Refer to the certain algorithms designed to compute optimal policies for a perfect model of the environment

eg. To match two sequences:

GAATTCAGTTA

GGATCGA

The key idea is to organise and structure the search for good policies

```
G _ A A T T C A G T T A
|       |   | |   |       |
G G _ A _ T C _ G _ _ A
```

DP from reinforcement learning point of view

For a finite MDP

The target is to find optimal value function V*

$$V^*(s) = \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s,\ a_t = a\}$$

$$= \max_a \Sigma_{s'} \mathcal{P}^a_{ss'} [R^a_{ss'} + \gamma V^*(s')]$$

reward *state* *action*

Bellman optimality equation

value function

$Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$

$E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$

Policy Evaluation

We know that state value function for arbitrary policy π

$$V^{\pi}(s) = E_{\pi}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \gamma^3 r_{t+2} \mid s_t = s\}$$

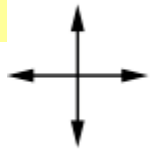
$$= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s+1) \mid s_t = s\} = \boldsymbol{\Sigma_a \pi(s, a) \Sigma P^a_{ss'}[R^a_{ss'} + V^{\pi}(s')]}$$

Each successive approximation is obtained by using the Bellman equation as the update rule.

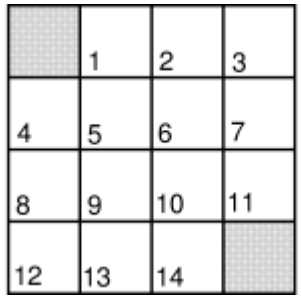
Each iteration backs up the value of every state once to produce the new approximate value function V_{k+1}.

Either state or state-action pair can be backed up.

Policy evaluation : example



actions



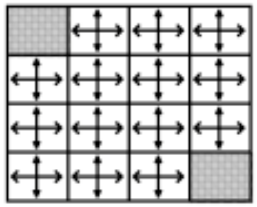
$r = -1$
on all transitions

V_k for the Random Policy

Greedy Policy w.r.t. V_k

$k = 0$

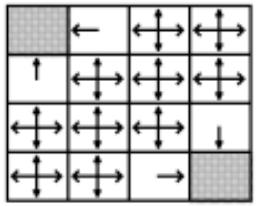
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

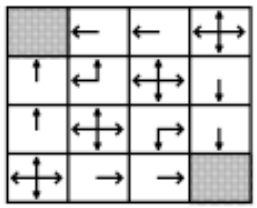
$k = 1$

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



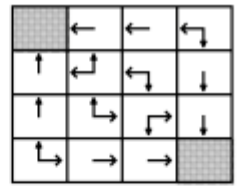
$k = 2$

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



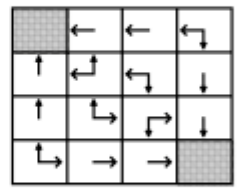
$k = 3$

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



$k = 10$

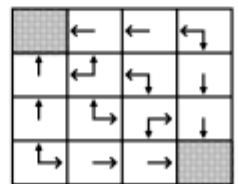
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



optimal policy

$k = \infty$

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



Policy Improvement

Policy π' is considered better than π if

$$Q^{\pi}(s,\pi'(s)) \geq V^{\pi}(s) \quad \text{or} \quad V^{\pi'}(s) \geq V^{\pi}(s)$$

Strict inequality at any state

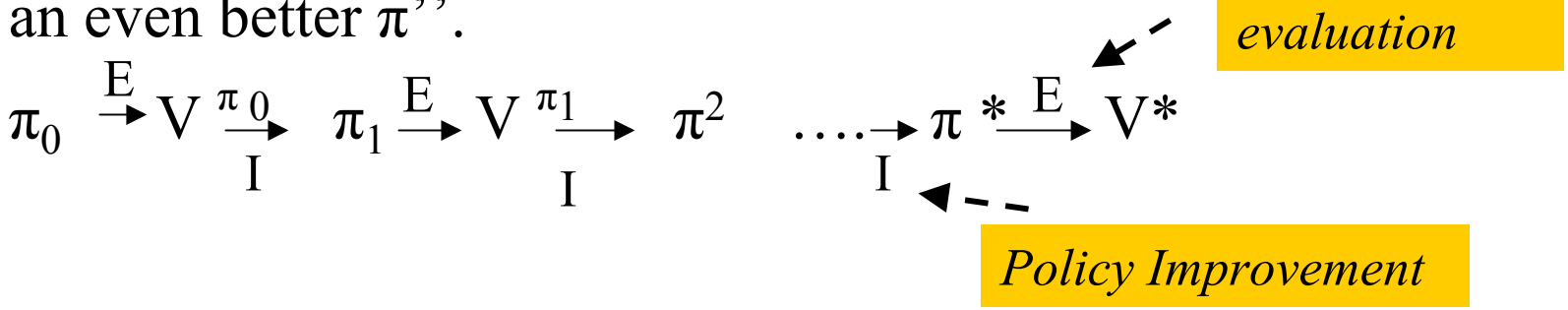
Strict inequality at least one state

It asserts that the evaluation of change in policy to a particular action should not be done only for one state but also for all states taking all actions in to consideration.

Policy improvement relies on choosing greedy policy over other policies.

Policy Iteration

Once a policy,π , has been improved using V^{π} to yield a better policy, π' , we can then compute $V^{\pi'}$ and improve it again to yield an even better π''.



Jack's Problem

Maximum 20 cars

gets10$
for a car

2$

Maximum 5
cars in night

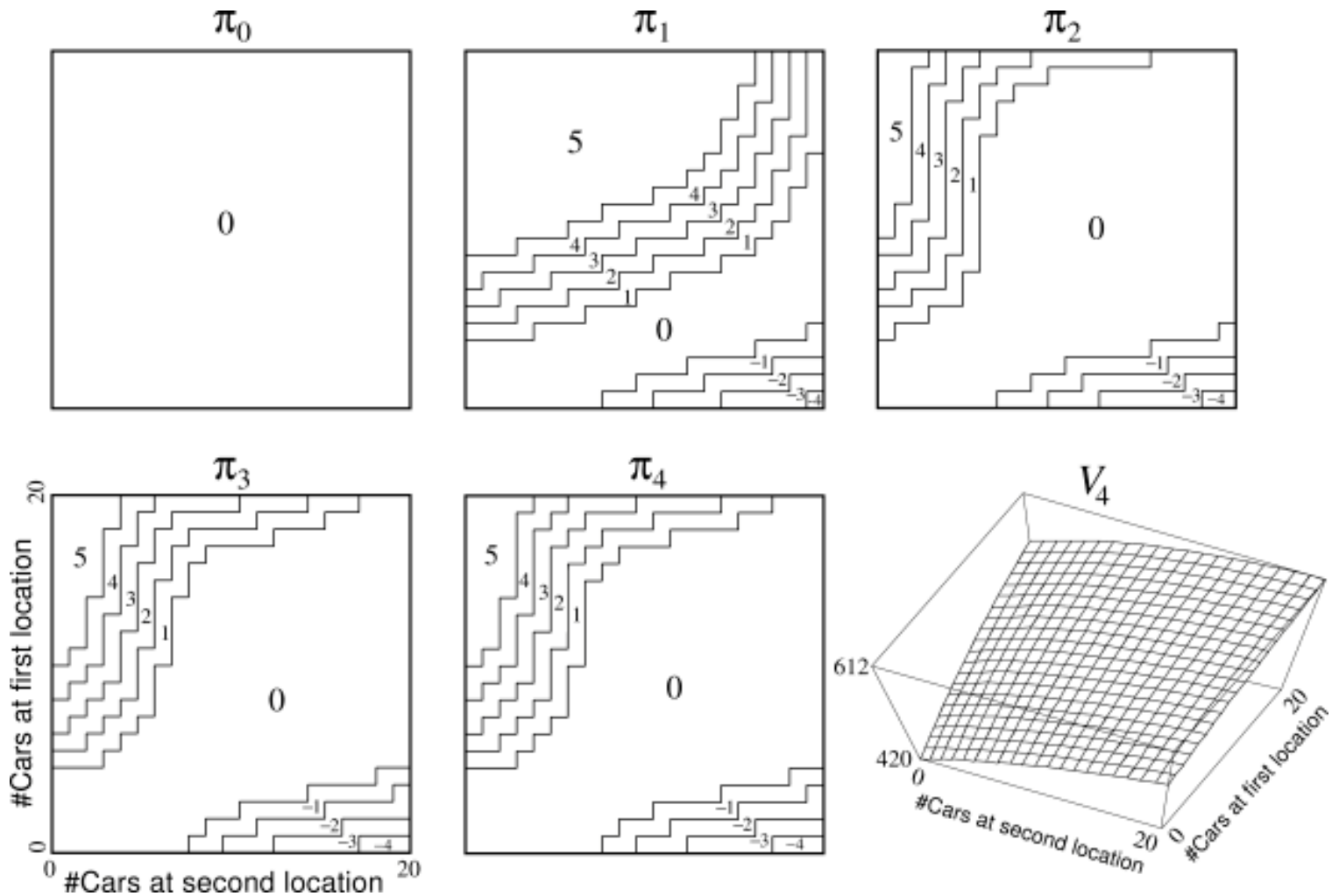
Maximum 20 cars

gets10$
for a car

Customer return
after 1 day

Poisson
random
number

Policy iteration for Jack's problem



Value Iteration

The policy evaluation step in policy iteration can be truncated to avoid iterative multiple sweeps through the whole state set.

The Bellman optimality equation is used as update rule

The maximum is taken over all the actions not over all the states

Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.

Asynchronous Dynamic Programming

Asynchronous DP algorithms are in-place iterative DP algorithms

not organized in terms of systematic sweeps of the state set.

back up the values of states in any order

values of some states may be backed up several times

make it easier to intermix computation with real-time interaction

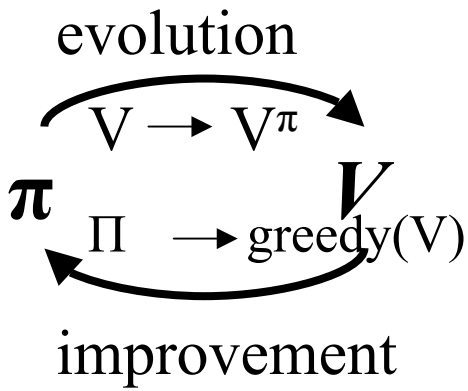
experience can be used to determine the states to which the DP algorithm applies its backups

Generalized Policy Iteration

Policy iteration consists of two simultaneous

interacting processes, one making the value function consistent with the current policy (policy evaluation)

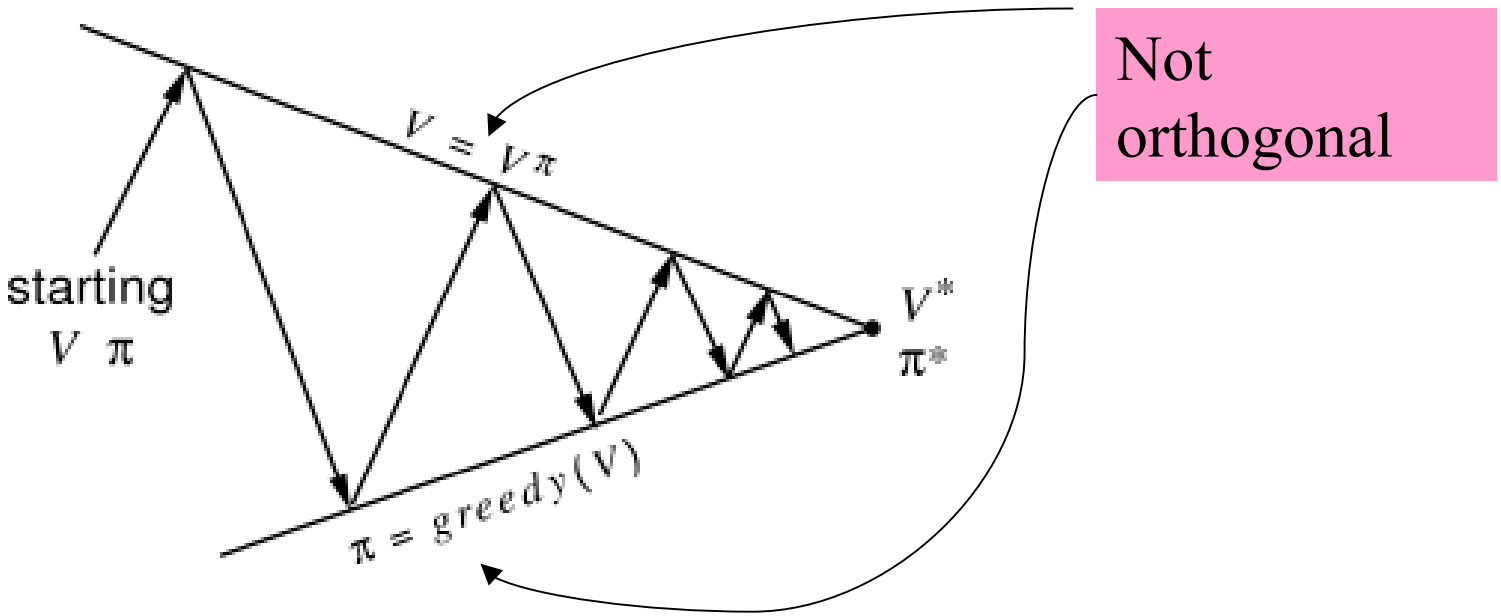
and the other making the policy greedy with respect to the current value function (policy improvement).



Generalized Policy Iteration

value function stabilizes only when it is consistent with the current policy

the policy stabilizes only when it is greedy with respect to the current value function



CONCLUSION

For ***n*** states and ***m*** actions DP method takes number of computational operations less than polynomial function of ***n*** and ***m*** out of original m^n deterministic policies.

DP is limited by *curse of dimensionality* : the number of states often grows exponentially with the number of state variables

Asynchronous DP methods are often preferred for large problems

Iterative policy evaluation : a classical approximation algorithm for solving a system of linear equation.its version

It's *in-place* version is called Gauss-Seidel-Style.