Reinforcement Learning

Examples and Design

Andrea Bonarini

Artificial Intelligence and Robotics Lab
Department of Electronics and Information
Politecnico di Milano

E-mail: bonarini@elet.polimi.it
URL:http://www.dei.polimi.it/people/bonarini

Learning from interaction

(Sutton, Barto, 1996)

Reinforcement learning is learning from interaction with an environment to achieve a goal, despite the **uncertainty** about the environment.

Agent’s **actions affect the environment**, and so its options and opportunities, including the possibility to learn further.

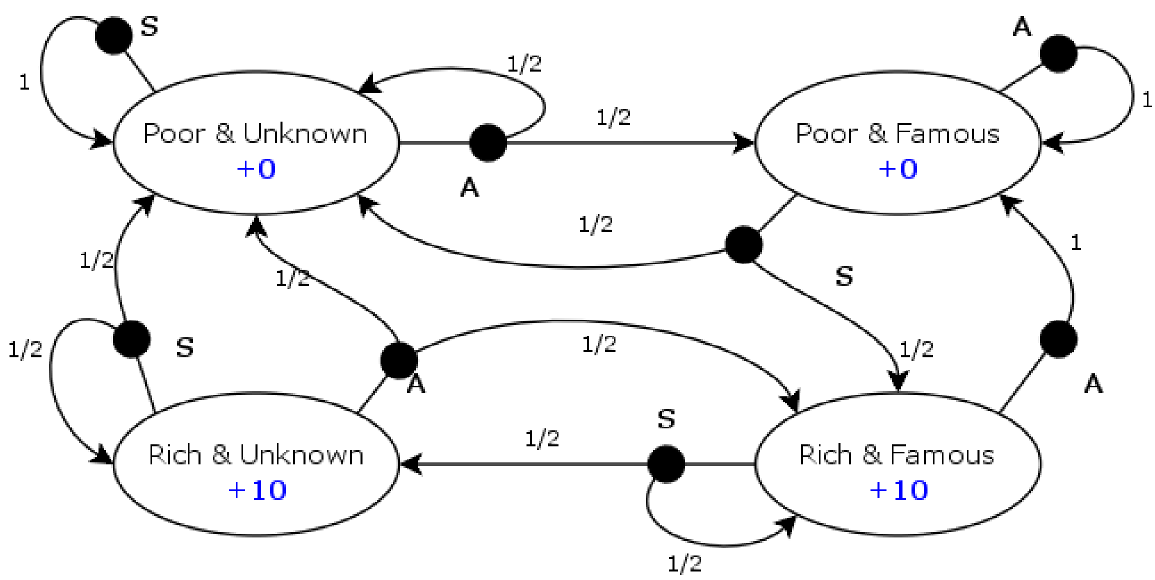
A correct choice requires taking into account **indirect**, **delayed consequences** of actions, which often cannot be predicted appropriately, due to uncertainty and poor information about the environment.

The agent knows when it has reached its own **goals.**

The agent can use its **experience** to improve.

Value Iteration: an example (I)

Let’s apply Value Iteration to the simple example here below…



… You have just opened a start-up company and in any state you have to decide whether to save the money you have (S) or to spend it in advertising (A).

Which is the optimal policy, given that $\gamma=0.9$*?*

Dynamic Programming: Value Iteration

1. *Initialization*

   $\pi \leftarrow$ *an arbitrary policy*

   $V \leftarrow$ *an arbitrary function* : $\mathbf{S} \rightarrow \Re$ *(e.g., V(s)=0 for all s)*

   $\theta \leftarrow$ *a small positive number*

2. *State evaluation*

   Repeat

   $\Delta \leftarrow 0$

   For each $s \in \mathbf{S}$:

   $v \leftarrow V(s)$

   $V(s) \leftarrow \max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$

   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   until $\Delta < \theta$

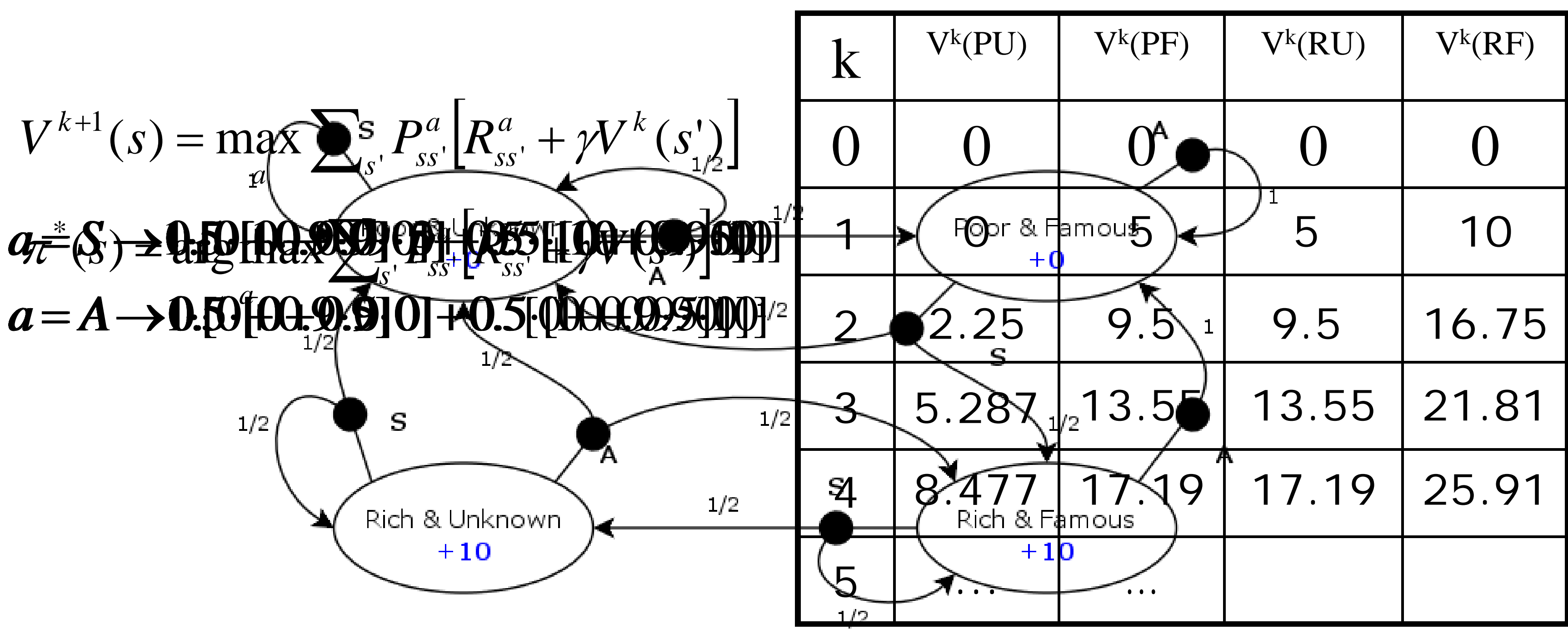
3. *Output*

   Output a deterministic policy such that $\pi(s) \leftarrow \arg\max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$

Value Iteration: an example (II)

Let's apply the Value Iteration Algorithm to our MDP

$$V^{k+1}(s) = \max_a \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^k(s')\right]$$



k	$V^k(PU)$	$V^k(PF)$	$V^k(RU)$	$V^k(RF)$
0	0	0	0	0
1	0	5	5	10
2	2.25	9.5	9.5	16.75
3	5.287	13.55	13.55	21.81
4	8.477	17.19	17.19	25.91
5	...	...		

$$\pi^*(PU) = A, \quad \pi^*(PF) = S$$

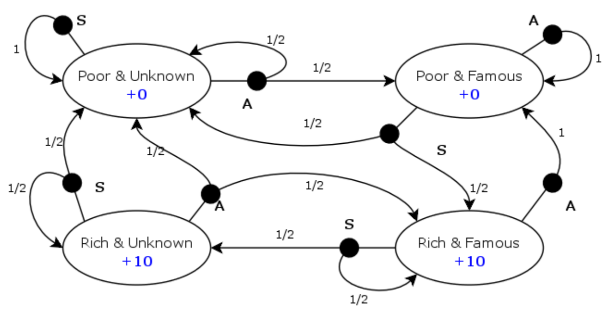
$$\pi^*(RU) = A, \quad \pi^*(RF) = S$$

Value Iteration: an example (III)

Summary of the results:

$$\pi^*(PU) = A, \quad \pi^*(PF) = S$$
$$\pi^*(RU) = A, \quad \pi^*(RF) = S$$

k	$V^k(PU)$	$V^k(PF)$	$V^k(RU)$	$V^k(RF)$
0	0	0	0	0
1	0	5	5	10
2	2.25	9.5	9.5	16.75
3	5.287	13.55	13.55	21.81
4	8.477	17.19	17.19	25.91
5	...	...		



Policy Iteration

The policy Iteration algorithm alternates the evaluation of the state and the policy improvement

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

Dynamic Programming: Policy Iteration I

1. *Initialization*

   $\pi \leftarrow$ *an arbitrary policy*

   $V \leftarrow$ *an arbitrary function* : $\mathbf{S} \rightarrow \Re$

   $\theta \leftarrow$ *a small positive number*

2. *Policy evaluation*

   Repeat

   $\Delta \leftarrow 0$

   For each $s \in \mathbf{S}$:

   $v \leftarrow V(s)$

   $V(s) \leftarrow \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$

   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   until $\Delta < \theta$

3. *Policy improvement (cont.)*

Probability to get in state s’ from state s, by taking action a

Dynamic Programming: Policy Iteration II

3. Policy improvement (cont'd)

policy-stable $\leftarrow$ *true*

For each $s \in \mathbf{S}$:

b$\leftarrow \pi(s)$

$\pi(s) \leftarrow argmax_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$

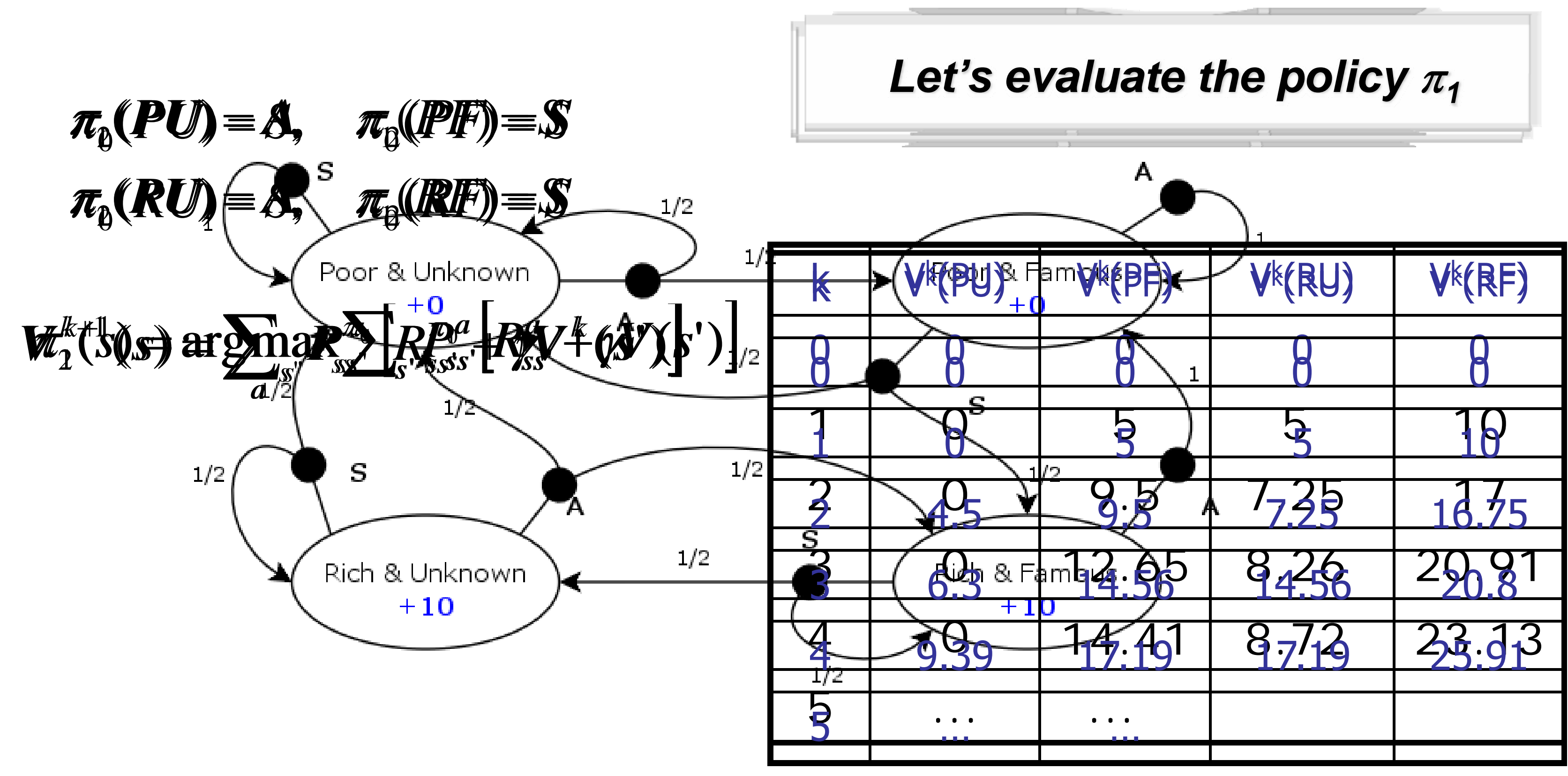
If $b \neq \pi(s)$ *then policy-stable* $\leftarrow$ *false*

If policy-stable then stop else go to 2

Notice that, whenever a policy is changed, all the policy evaluation activity (step 2) has to be performed again.

Policy Iteration: an example

Let's apply the Policy Iteration Algorithm to our MDP



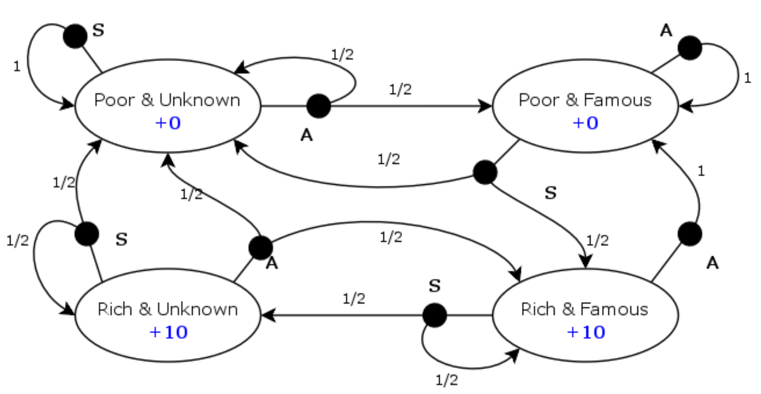
k	$V^k(PU)$	$V^k(PF)$	$V^k(RU)$	$V^k(RF)$
0	0	0	0	0
1	0	5	5	10
2	4.5	9.5	7.25	16.75
3	6.3	14.56	14.56	20.8
4	9.39	17.19	17.19	25.91
5	...	...		

k	$V^k(PU)$	$V^k(PF)$	$V^k(RU)$	$V^k(RF)$
0	0	0	0	0
1	0	5	5	10
2	0	9.5	7.25	17
3	0	12.65	8.26	20.91
4	0	14.41	8.72	23.13
5	...	...		

Policy Iteration: summary of results

$$\pi^*(PU) = A, \quad \pi^*(PF) = S$$
$$\pi^*(RU) = A, \quad \pi^*(RF) = S$$

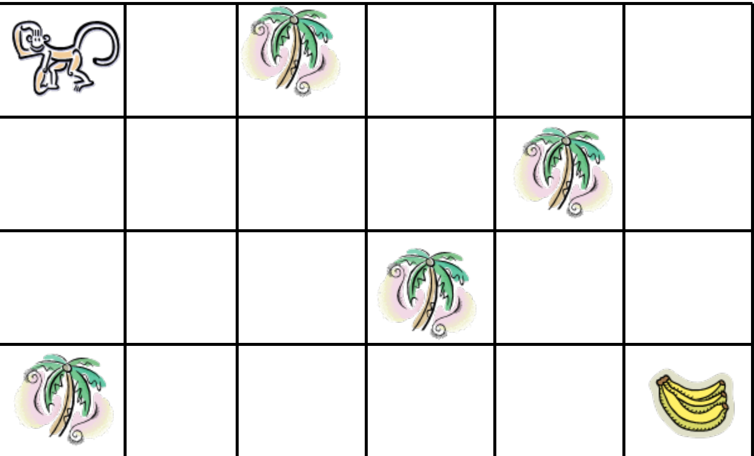
k	$V^k(PU)$	$V^k(PF)$	$V^k(RU)$	$V^k(RF)$
0	0	0	0	0
1	0	5	5	10
2	4.5	9.5	7.25	16.75
3	6.3	14.56	14.56	20.8
4	9.39	17.19	17.19	25.91
5	…	…		



A (possibly starving) MDP…

Let’s take that the agent A is a monkey which should learn how to reach bananas

- The states are the cells of the grid reported in the drawing
- The possible actions are movements of one cell in the four cardinal directions, but on the borders and on the obstacles this does not have any effect
- The monkey receives a reinforcement of 1000 when reaching bananas
- The discount factor is $\gamma=0.9$



Q-Learning: an example (Trial 0)

Initialize all the Q values to 0

The agent moves in the environment updating the Q values (let's take $\alpha=0.5$)

$$\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \alpha\left(R(s_t,a_t) + \gamma \max_{a'} Q(s_{t+1},a') - \hat{Q}(s_t,a_t)\right)$$

Reward = 1000

$$\hat{Q}(s_t,a_t) \leftarrow 0 + 0.5(0+0-0)$$

$$\hat{Q}(s_t,a_t) \leftarrow 0 + 0.5(0+0-0)$$

$$\hat{Q}(s_t,a_t) \leftarrow 0 + 0.5(1000+0-0)$$

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 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	0 0 0 0	0 0 0 0	0 0 500 0	

Q-Learning: an example (Trial 1)

The agent moves in the environment updating the Q values
(let's take $\alpha=0.5$)

$$\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \alpha\left(R(s_t,a_t) + \gamma \max_{a'} Q(s_{t+1},a') - \hat{Q}(s_t,a_t)\right)$$

Reward = 1000

$$\hat{Q}(s_t,a_t) \leftarrow 0 + 0.5(0 + 0.9 \cdot 500 - 0)$$

$$\hat{Q}(s_t,a_t) \leftarrow 500 + 0.5(1000 + 0 - 500)$$

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 0 0 0		0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0		0 0 0 225	0 0 0 0
	0 0 0 0	0 0 0 0	0 0 0 0	0 0 750 0	

Q-Learning: an example (Trial 2)

The agent moves in the environment updating the Q values
(let's take $\alpha=0.5$)

$$\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \alpha\left(R(s_t,a_t) + \gamma \max_{a'} Q(s_{t+1},a') - \hat{Q}(s_t,a_t) \right)$$

Reward = 1000

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 0 0 0		0 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0		0 0 0 225	0 101 0 0
	0 0 0 0	0 0 0 0	0 0 0 0	0 0 856 0	

Q-Learning: an example (Trial 10)

The agent moves in the environment updating the Q values
(let's take $\alpha=0.5$)

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha\left(R(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') - \hat{Q}(s_t, a_t)\right)$$

Reward = 1000

0 0 0 0	0 0 0 0		0 0 91 0	0 0 222 0	0 0 0 408
0 0 0 0	0 0 0 0	0 0 4 0	26 0 0 0		0 0 0 603
0 0 0 0	0 0 0 0	0.37 0 0 0		0 0 0 890	0 765 0 0
	0 0 0 0	0 0 0 0	0 0 449 0	0 0 999 0	

Q-Learning: an example (Trial 11)

By leaving the old path for the new,
you know what you leave
but not what you can find!

You can learn only by
making errors!

Exploitation/Exploitation

Would the monkey like to leave sure food to explore a new way which might be a possible shortcut??

A summary of the design strategy

1. Identify the **state** in terms of variables and their possible values: the state is described by the set of state variables
2. Identify the **actions** that can be taken: each action should bring the agent from one state to another
3. Identify eventual **constraints** that make actions not executable in certain states, or states that cannot be reached
4. Identify the **reinforcement** function, which should reinforce for the results obtained (not for the action taken) either at each step or only when a state is reached: if we know which action is good and which is bad we do not need RL
5. Identify the proper **reinforcement distribution algorithm**

Let’s try with a simple example…

A design example

Let's try to learn the best actions for a robot that has to collect two different types of trash (say cans and paper) and bring them to the respective recycle bins. A possible design might be:

1. **State definition:** TrashInHand ∈{Can, Paper}, TrashInFront∈{True, False}, CloserTrashDirection∈{N, NE, E, SE, S, SW, W, NW}, TrashBinInFront∈{Can, Paper}, PaperBinDirection ∈{N, NE, E, SE, S, SW, W, NW}, CanBinDirection ∈{N, NE, E, SE, S, SW, W, NW}
2. **Action definition:** MoveDirection ∈{N, NE, E, SE, S, SW, W, NW, Stop}, HandAction ∈{NOP, Take, Release}
3. **Constraints definition:** the robot cannot go over the bins
4. **Reinforcement:** 100 when the trash is put on the correct bin, -100 if in the wrong bin, 50 when (any) trash is taken in hand
5. **Reinforcement distribution:** Q(λ), since in a given state it is important to select the correct action (so better Q than TD), with eligibility trace, since reinforcement is delayed. DP would be incorrect since reinforcement is missing in many state transitions. Montecarlo would require too many (for a real robot) experiments to converge.