Dynamic programming and Monte Carlo

Herke van Hoof

Announcements

Tutorial groups slightly rearranged, group A is now online

- If this concerns you, you should have gotten email, but let me know if something went wrong

Exam situation slightly improved

- Presentations for DL4NLP moved

Plan for today

Last lecture, we talked about the exploration/exploitation tradeoffs in bandits and MDPs as model for decision making

Today, we'll define **optimal behaviour**, and see how we can find optimal policies for **known MDPs**

Markov decision processes

Last lecture, we introduced MDPs, formally consisting of:

- A finite set of states
- A finite set of actions for each state (often the same in all states)
- A dynamics function

$$p\left(s', r|s, a\right) \doteq \Pr\left\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\right\}$$

Sometimes written as: transition function $p\left(s'|s, a\right)$

reward function $p(r|s, a, s')$

- A discount factor $\gamma \in [0,1)$

Trajectories

The experience of the agent can be summarised as a trajectory, or sequence of states, actions, and rewards

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Some environment have natural "terminal states" (game won or lost, exit of maze found)

Then, it is natural to split the trajectory in episodes

$$S_0^{(1)}, A_0^{(1)}, R_1^{(1)}, S_1^{(1)}, A_1^{(1)}, \ldots, R_{T^{(1)}}^{(1)}, S_{T^{(1)}}^{(1)}$$
$$S_0^{(2)}, A_0^{(2)}, R_1^{(2)}, S_1^{(2)}, A_1^{(2)}, \ldots, R_{T^{(2)}}^{(2)}, S_{T^{(2)}}^{(2)}$$

Return and horizons

Last lecture, we have introduced the notion of **return**
(only for episodic tasks)

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

As well as **discounted return**

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

(continuing or episodic tasks)

Unified notation

Different notation for episodic and continuing tasks is a bit awkward.

We will always use the discounted return, but allow $\gamma=1$ if every episode terminates

Return and horizons

Note that subsequent returns are related

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

Return and horizons

Note that subsequent returns are related

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \\
G_{t+1} &= R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+3} + \cdots \\
G_t &= R_{t+1} + \gamma \left(R_{t+2} + \gamma^2 R_{t+3} + \cdots \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Policies and value functions

What we really care about, is learning how to take good actions

The way actions are selected is called the policy $\pi(a|s)$

For every state, the policy specifies a probability distribution over actions

Of course, there are many possible policies. Which is best?

Policies and value functions

A policy should get high expected (discounted) returns

The state-value function expresses how good π is from *s*:

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t | S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right]$$

So, if we start from *s*, we might prefer π over π' if

$$v_\pi(s) > v_{\pi'}(s)$$

We'll see later how *v* can be computed or learned!

Policies and value functions

Of course, for a fixed policy the value function also measures how good it is for the agent to be in a **state**

Similarly, we might wonder how good it is to be in a (state,action) pair. We'll express this with a state-action value function

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right]$$

Similar to the bandit case, we could use q to select actions
e.g.: greedy policy: $a^* = \max_a q_\pi(s,a)$

Policies and value functions

Like the return, value functions are related

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$
$$= \quad \square$$
$$= \quad \square$$
$$= \quad \square$$

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$
$$= \quad \square$$
$$= \quad \square$$

Similar identities hold for q

Policies and value functions

goal: the relation between value func of current states and value func of future state

Like the return, value functions are related

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + G_{t+1} | S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1}] + \mathbb{E}_{a\sim\pi, s'} \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \\
&= \mathbb{E}_\pi[R_{t+1}] + \mathbb{E}_{a\sim\pi, s'} v_\pi(s')
\end{aligned}
$$

conditional expected return

expected return for the future next time step

s' is a random variable. because we dont know whhich states we end up in

action that we will randomly draw from the policy

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\
&= \mathbb{E}_{a\sim\pi}\left[\mathbb{E}_\pi[G_t | S_t = s, A_t = a]\right] \\
&= \mathbb{E}_{a\sim\pi} q_\pi(s, a)
\end{aligned}
$$

relation between value func and the state-value func

relation between current q func and the state-action pair in the next time step

relation between q and v

Similar identities hold for q

Policies and value functions

Finally we can define optimal policies

A policy is optimal if

$$v_\pi(s) \geq v_{\pi'}(s) \quad \forall s, \pi'$$

Multiple policies can be optimal, share optimal value functions:

$$v_*(s) \doteq \max_\pi v_\pi(s)$$

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a)$$

We'll see how to learn these, too...

Policies and value functions

Optimal policies are again related:

$$q_*(s,a) = \qquad v_*(S_{t+1})$$

$$v_*(s) = \quad q_*(s,a)$$

$$= \qquad\qquad v_*(S_{t+1})$$

$$q_*(s,a) = \qquad\qquad q_*(S_{t+1}, a')$$

Policies and value functions

relation between q* and v*
optimal q func at a certain state-action pair = immediate reward + optimal future reward.
at current state s, we take action a. from here onwards, we will take all optimal policy

Optimal policies are again related:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*\left(S_{t+1}\right) | S_t = s, A_t = a\right]$$

relation between v* and q*. choose the action which gives me the max reward

$$v_*(s) = \max_a q_*(s,a)$$

plug in q func. then we get:
recursive relation between v*(s) and v*(S_t+1)

$$= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*\left(S_{t+1}\right) | S_t = s, A_t = a\right]$$

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*\left(S_{t+1}, a'\right) | S_t = s, A_t = a\right]$$

plug in v*(S_t+1), then we get relation between q*(s,a) and q*(next state, action at next state)

Last two equations called *Bellman optimality equations*, and show how value of subsequent states / (s,a) pairs is related

How to find optimal policy & value fcs?

Thanks to Jan Peters

How to find optimal policy & value fcs?

You have won an award in Madrid!

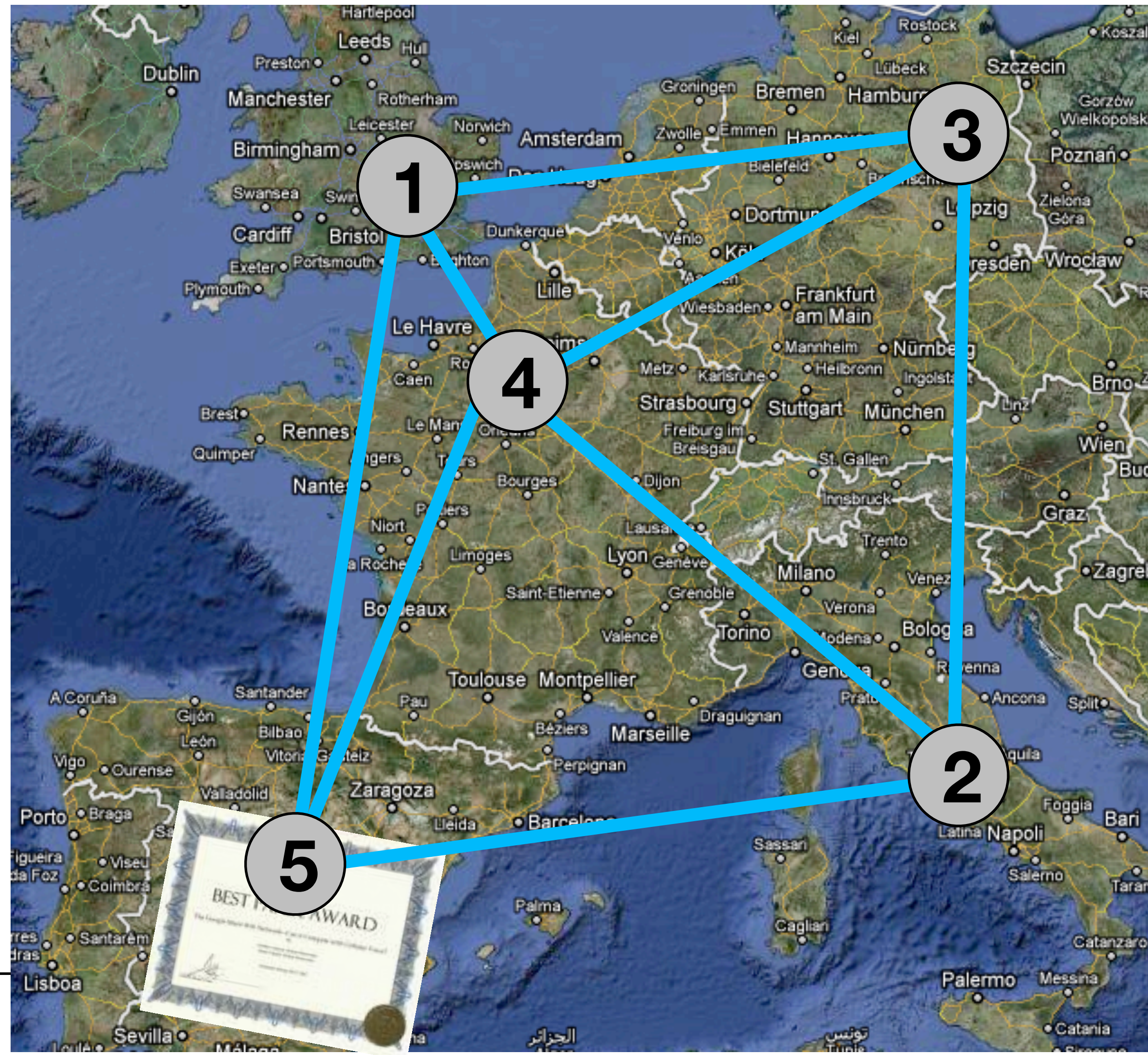
What is the Optimal Policy to Collect it?

Thanks to Jan Peters

How to find optimal policy & value fcs?

You have won an award in Madrid!

What is the Optimal Policy to Collect it?

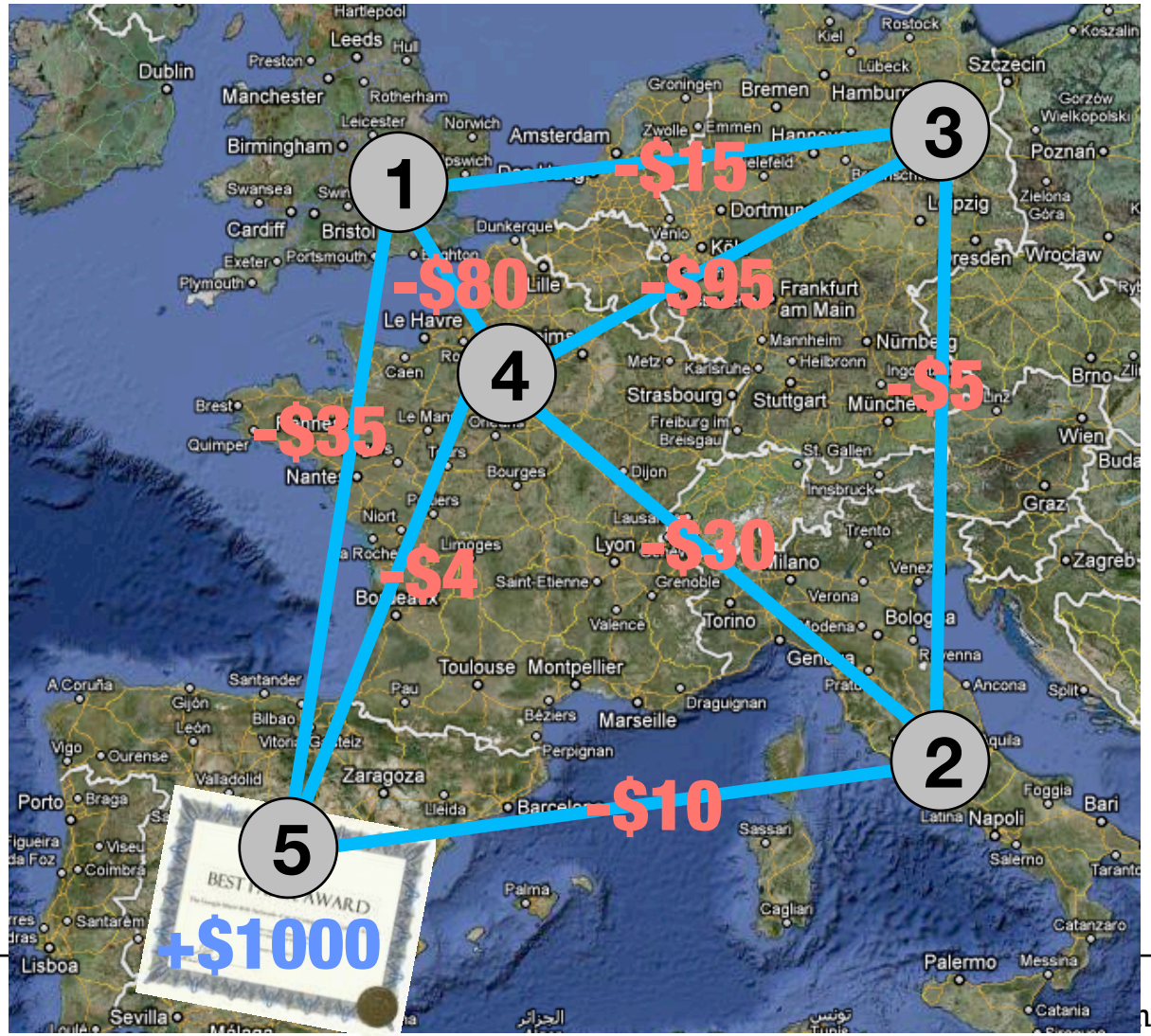


Thanks to Jan Peters

How to find optimal policy & value fcs?

You have won an award in Madrid!

What is the Optimal Policy to Collect it?



Thanks to Jan Peters

Dynamic programming

Main idea from dynamic programming:

- Break a problem down into subproblem
- Optimal solutions of subproblems used in solution of larger problem

Bellman's principle of optimality:

"An optimal sequence of controls in a multistage optimization problem has the property that whatever the initial stage, state and controls are, the remaining controls must constitute an optimal sequence of decisions for the remaining problem with stage and state resulting from previous controls considered as initial conditions"

Richard Bellman, Dynamic Programming, 1957

Thanks to Jan Peters

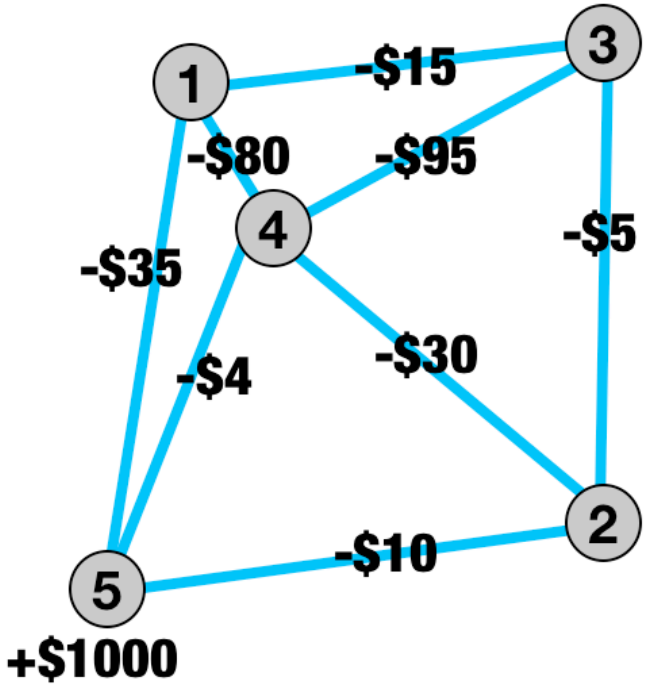
Basic idea

"whatever the initial stage [...], remaining controls must constitute an optimal sequence of decisions for the remaining problem with stage and state resulting from previous controls considered as initial conditions"

Bellman's quote suggests trying to first find the last controls, and working backwards. A solution for the last 1 step will be part of any solution for more than one step. Then solve for 2 steps, etc.

Thanks to Jan Peters

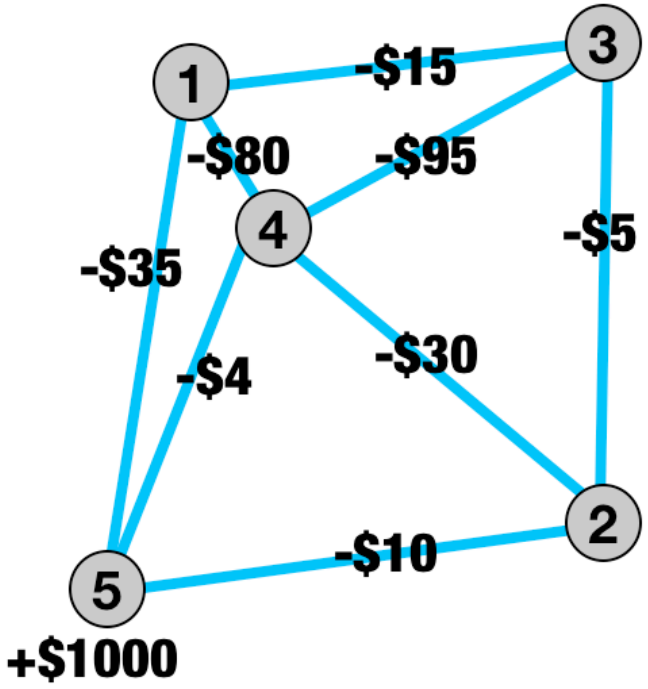
Let's try



	T-4	T-3	T-2	T-1	T
1					
2					
3					
4					
5					

Thanks to Jan Peters

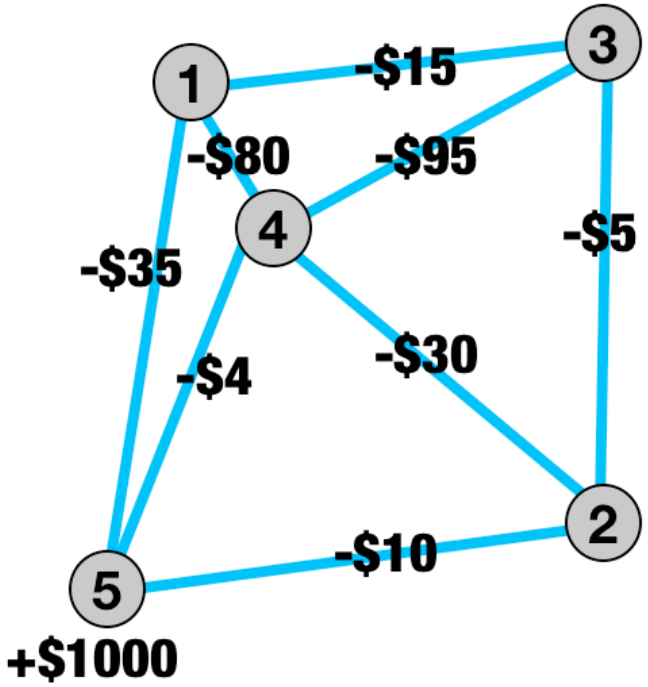
Let's try



	T-4	T-3	T-2	T-1	T	
					0	1
					0	2
					0	3
					0	4
					1000	5

Thanks to Jan Peters

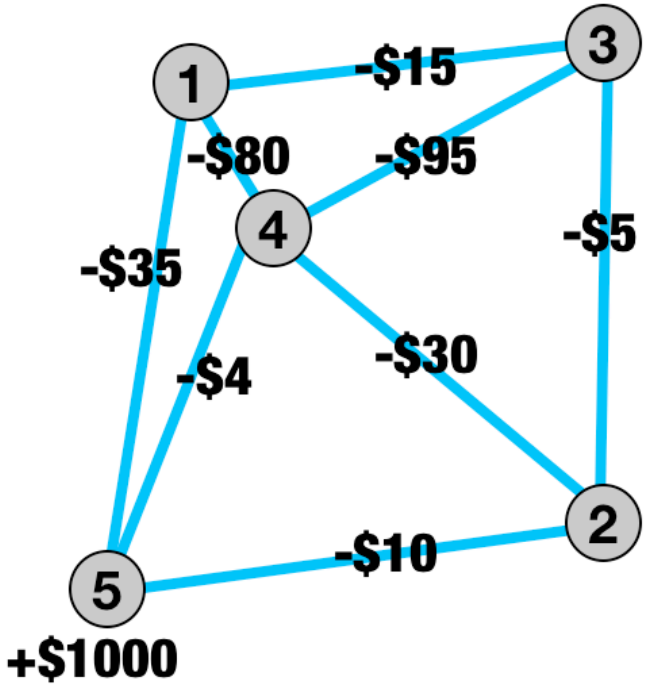
Let's try



	T-4	T-3	T-2	T-1	T	
				965	0	1
					0	2
					0	3
					0	4
					1000	5

Thanks to Jan Peters

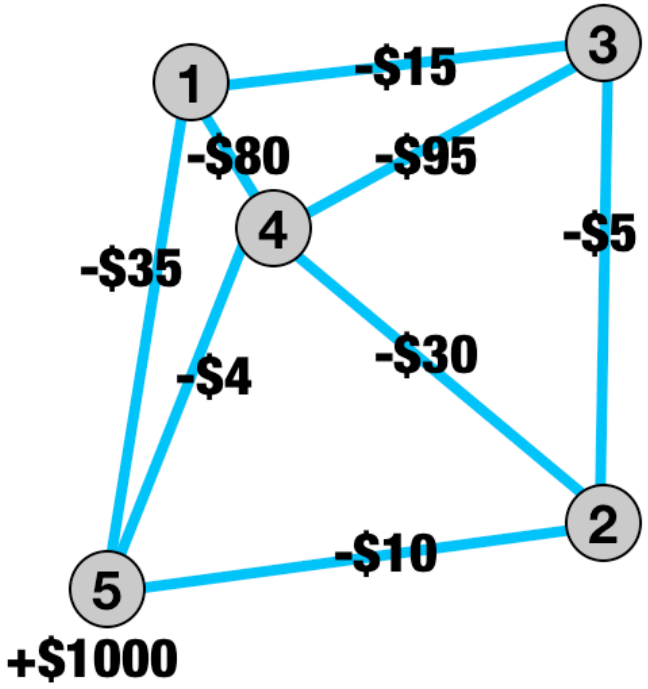
Let's try



	T-4	T-3	T-2	T-1	T	
				965	0	1
				990	0	2
					0	3
					0	4
					1000	5

Thanks to Jan Peters

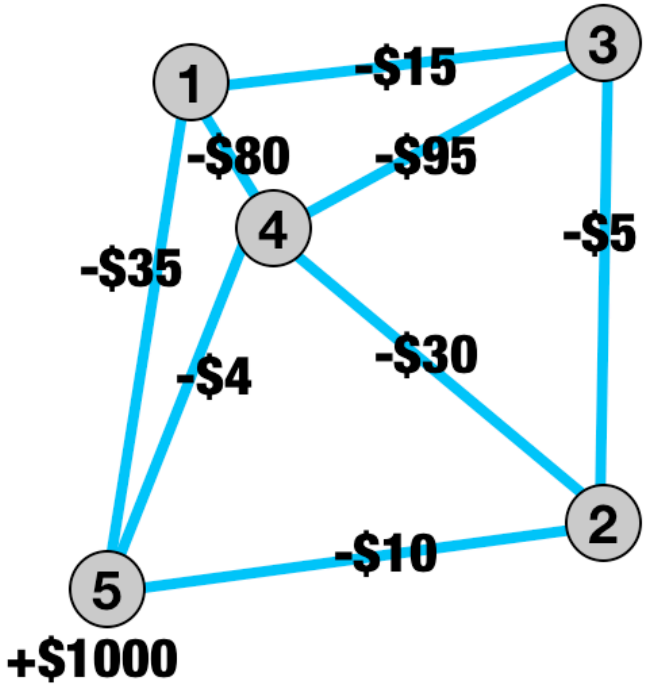
Let's try



	T-4	T-3	T-2	T-1	T	
				965	0	1
				990	0	2
				0	0	3
					0	4
					1000	5

Thanks to Jan Peters

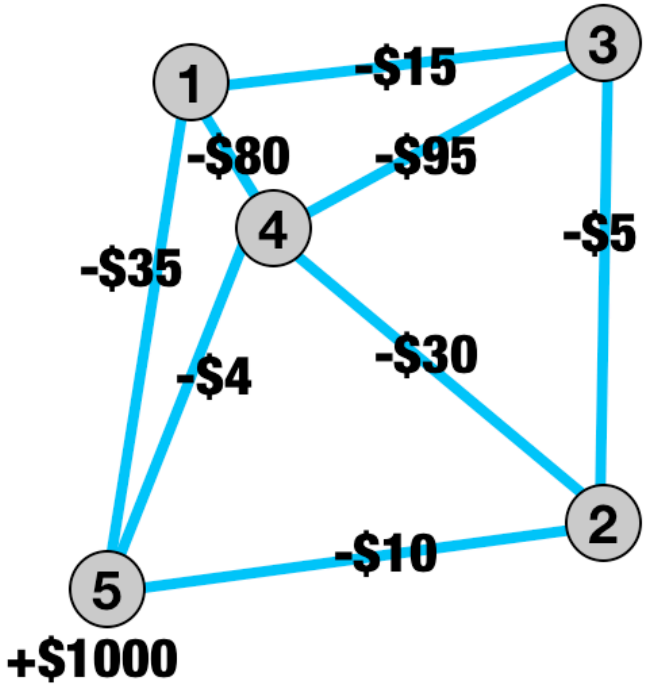
Let's try



	T-4	T-3	T-2	T-1	T	
				965	0	1
				990	0	2
				0	0	3
				996	0	4
				1000	1000	5

Thanks to Jan Peters

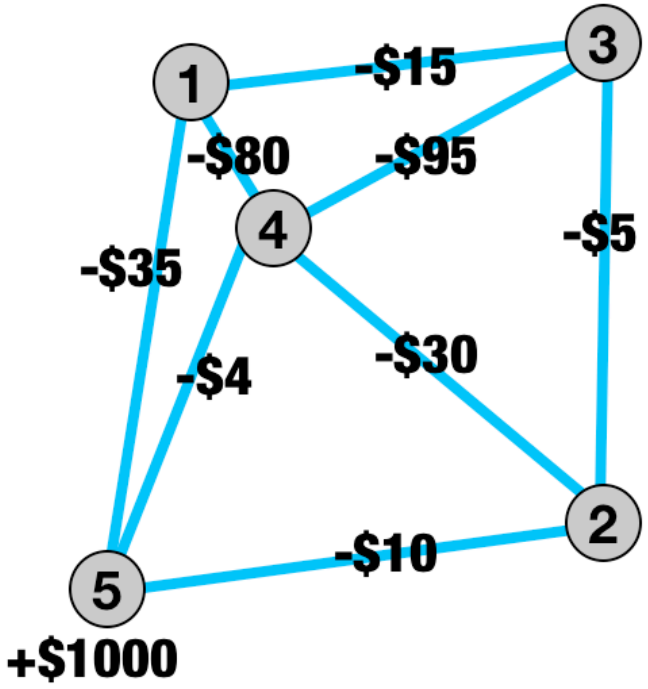
Let's try



	T-4	T-3	T-2	T-1	T	
			965	965	0	1
				990	0	2
				0	0	3
				996	0	4
				1000	1000	5

Thanks to Jan Peters

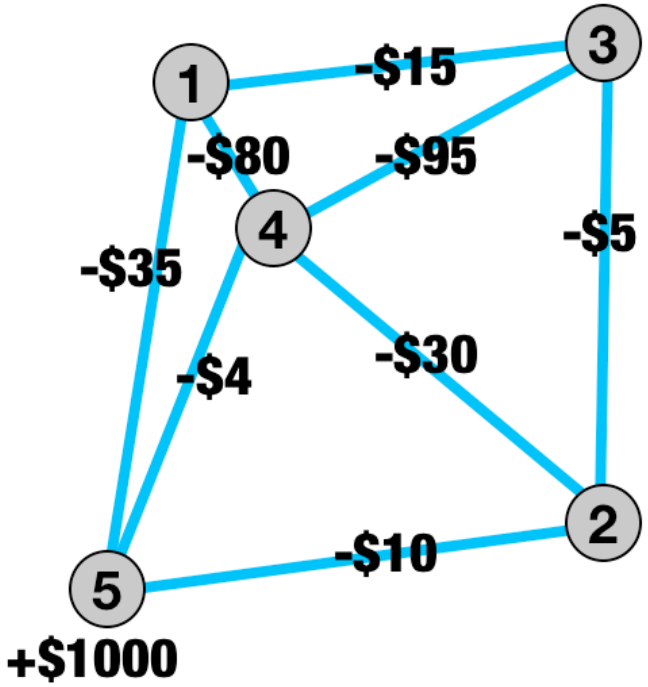
Let's try



	T-4	T-3	T-2	T-1	T	
			965	965	0	1
			990	990	0	2
				0	0	3
				996	0	4
				1000	1000	5

Thanks to Jan Peters

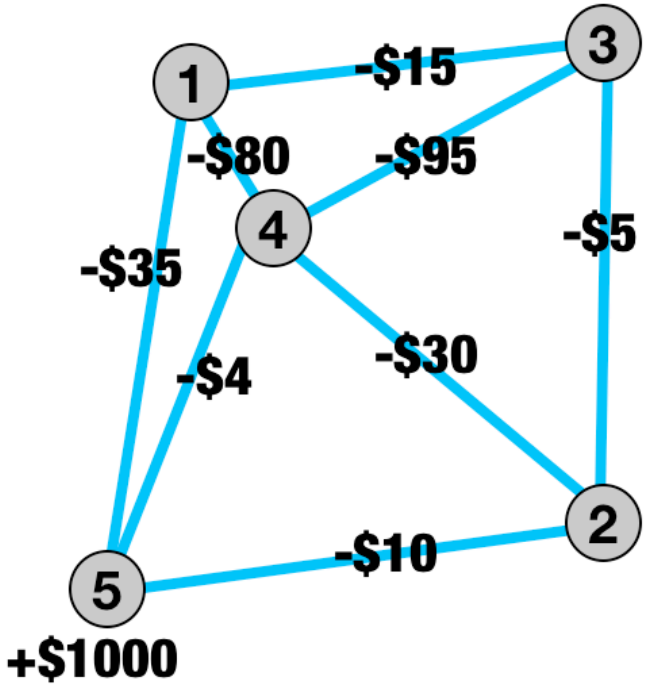
Let's try



	T-4	T-3	T-2	T-1	T	
			965	965	0	1
			990	990	0	2
			985	0	0	3
				996	0	4
				1000	1000	5

Thanks to Jan Peters

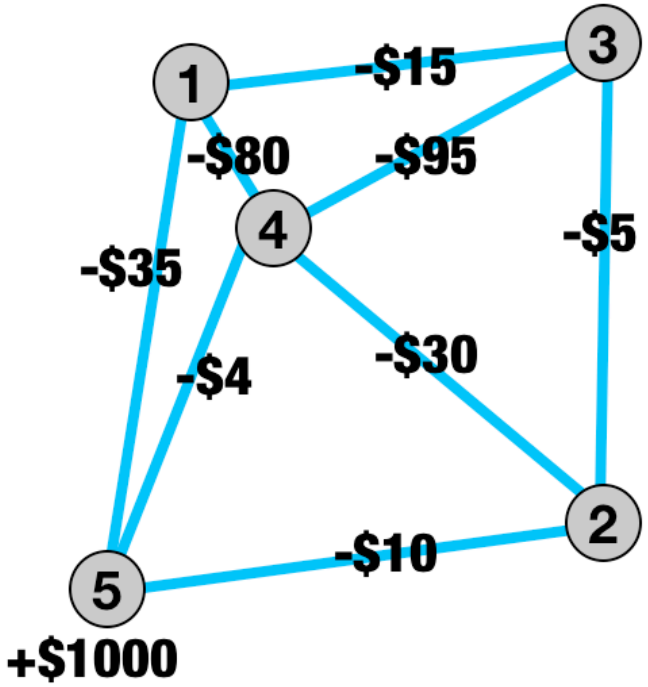
Let's try



	T-4	T-3	T-2	T-1	T	
			965	965	0	1
			990	990	0	2
			985	0	0	3
			996	996	0	4
			1000	1000	1000	5

Thanks to Jan Peters

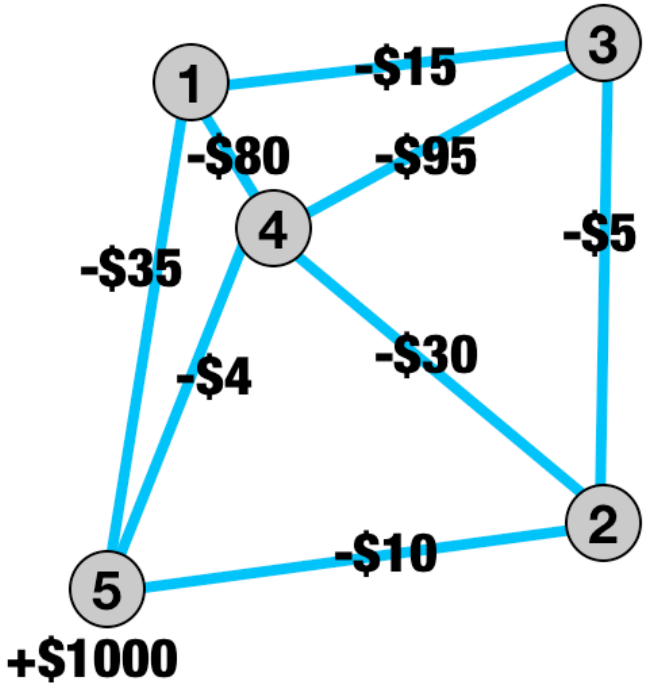
Let's try



	T-4	T-3	T-2	T-1	T	
		970	965	965	0	1
			990	990	0	2
			985	0	0	3
			996	996	0	4
			1000	1000	1000	5

Thanks to Jan Peters

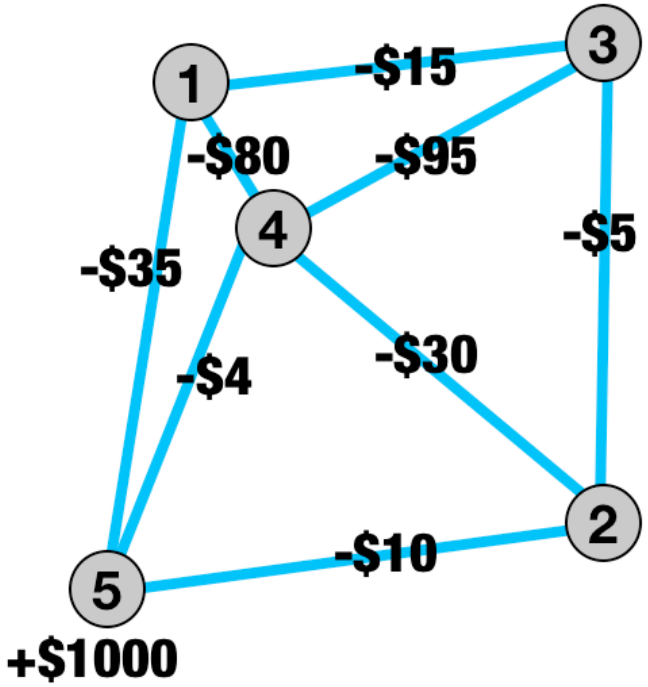
Let's try



	T-4	T-3	T-2	T-1	T	
		970	965	965	0	1
		990	990	990	0	2
		985	985	0	0	3
		996	996	996	0	4
		1000	1000	1000	1000	5

Thanks to Jan Peters

Let's try



	T-4	T-3	T-2	T-1	T	
	970	970	965	965	0	1
	990	990	990	990	0	2
	985	985	985	0	0	3
	996	996	996	996	0	4
	1000	1000	1000	1000	1000	5

Thanks to Jan Peters

Formalization

1.At the last step, we have the value function

$$V_T^*(\mathbf{x}) = 0$$

2.We compute the optimal policy such that

$$\pi_t^*(\mathbf{a} \mid \mathbf{s}) = \text{argmax}_\pi \quad \left[r(\mathbf{s}, \mathbf{a}) + V_{t+1}^*(\mathbf{s}')\right]$$

3.Obtain next value function

$$V_t^*(\mathbf{s}) = \max_\pi \quad \left[r(\mathbf{s}, \mathbf{a}) + V_{t+1}^*(\mathbf{s}')\right]$$

4.If not converged, go back to Step 2.

Thanks to Jan Peters

T-2	T-1	T	
$V_{T-2}(1)$	$V_{T-1}(1)$	$V_T(1)$	1
$V_{T-2}(2)$	$V_{T-1}(2)$	$V_T(2)$	2
$V_{T-2}(3)$	$V_{T-1}(3)$	$V_T(3)$	3
$V_{T-2}(4)$	$V_{T-1}(4)$	$V_T(4)$	4
$V_{T-2}(5)$	$V_{T-1}(5)$	$V_T(5)$	5

Formalization

1.At the last step, we have the value function

$$V_T^*(\mathbf{x}) = 0$$

2.We compute the optimal policy such that

$$\pi_t^*(\mathbf{a} \mid \mathbf{s}) = \operatorname{argmax}_\pi \mathbb{E}\left[r(\mathbf{s}, \mathbf{a}) + V_{t+1}^*(\mathbf{s}')\right]$$

3.Obtain next value function

$$V_t^*(\mathbf{s}) = \max_\pi \mathbb{E}\left[r(\mathbf{s}, \mathbf{a}) + V_{t+1}^*(\mathbf{s}')\right]$$

4.If not converged, go back to Step 2.

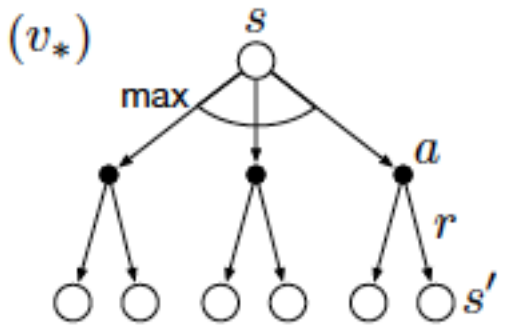
Thanks to Jan Peters

T-2	T-1	T	
$V_{T-2}(1)$	$V_{T-1}(1)$	$V_T(1)$	1
$V_{T-2}(2)$	$V_{T-1}(2)$	$V_T(2)$	2
$V_{T-2}(3)$	$V_{T-1}(3)$	$V_T(3)$	3
$V_{T-2}(4)$	$V_{T-1}(4)$	$V_T(4)$	4
$V_{T-2}(5)$	$V_{T-1}(5)$	$V_T(5)$	5

Value iteration

This is the value iteration algorithm

At convergence, we have found v* (as v no longer changes until “infinity”, this is solution to infinite-horizon problem)



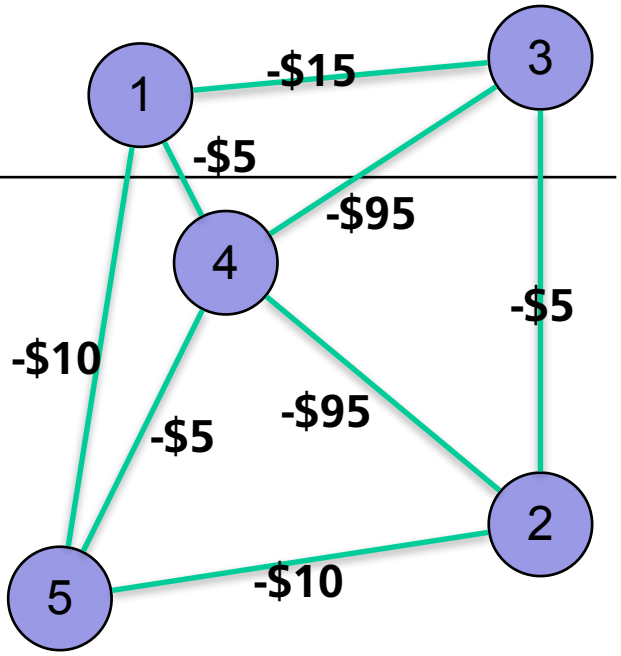
backup diagram

In the tutorial sessions, we’ll go through key steps in the convergence proof

How about finding v_π? (*Evaluation* of policy π)

Figure: Sutton&Barto; RL:AI

Policy evaluation



Let's try the policy that goes to a **random** other state

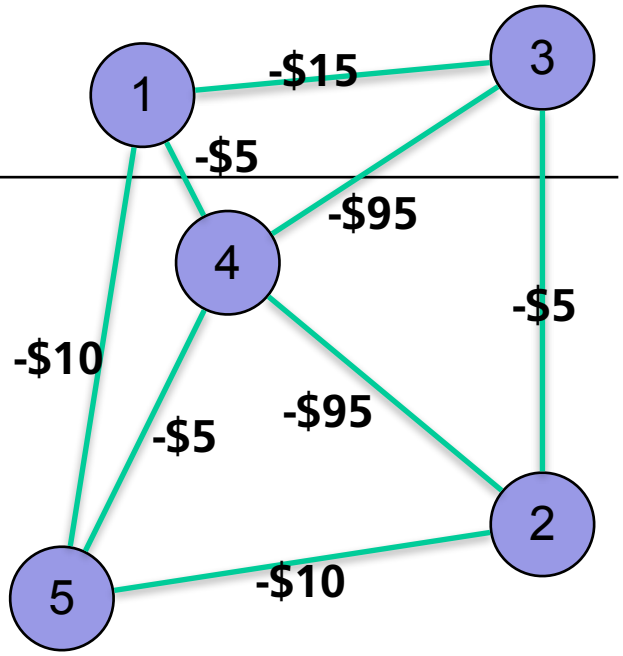
Start considering fixed-length episodes

Again: work back from last step

Bellman equation: $v_\pi(s) = \mathbb{E}_{a\sim\pi}\mathbb{E}_{s',r}[r + \gamma v_\pi(s')|s, a]$

$v_{\pi,T-1}(4) =$

Policy evaluation



Let's try the policy that goes to a **random** other state

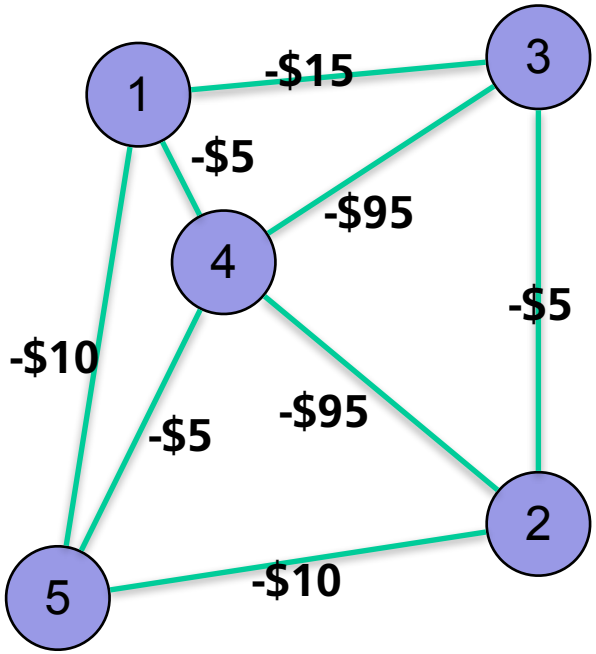
Start considering fixed-length episodes

Again: work back from last step

Bellman equation: $v_\pi(s) = \mathbb{E}_{a\sim\pi}\mathbb{E}_{s',r}[r + \gamma v_\pi(s')|s, a]$

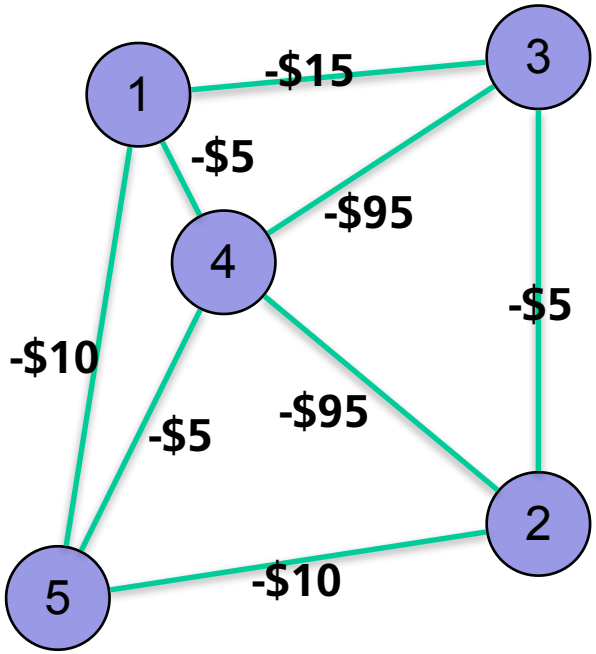
$$v_{\pi,T-1}(4) = \frac{-5+0}{4} + \frac{-95+0}{4} + \frac{-95+0}{4} + \frac{-5+1000}{4} = 200$$

Policy evaluation



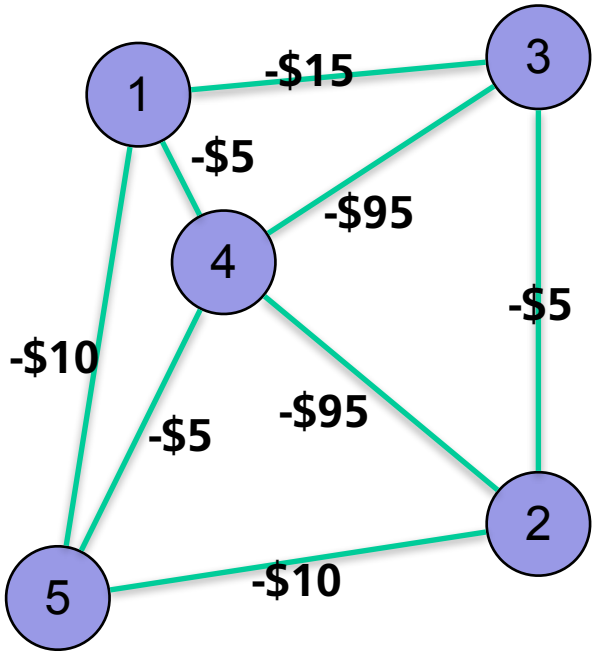
$V_{\pi,T-2}$	$V_{\pi,T-1}$	$V_{\pi,T}$	
		0	1
		0	2
		0	3
	200	0	4
		1000	5

Policy evaluation



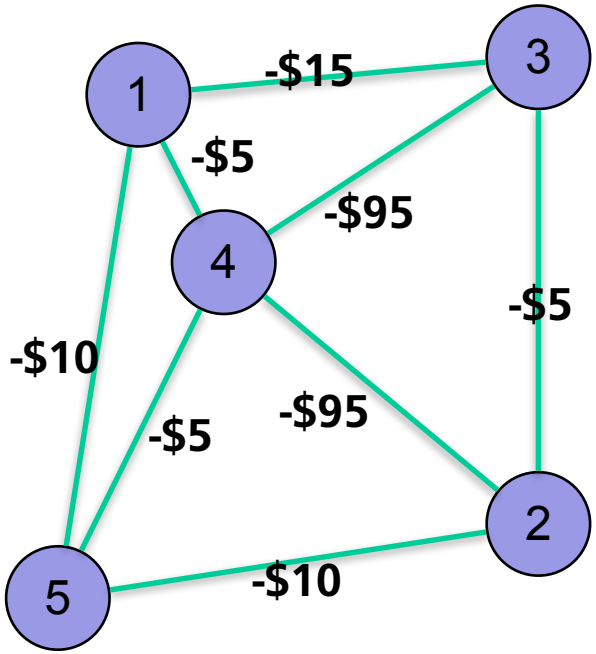
$V_{\pi,T-2}$	$V_{\pi,T-1}$	$V_{\pi,T}$	
	323,33	0	1
	296,67	0	2
	-38.33	0	3
	200	0	4
	1000	1000	5

Policy evaluation



$V_{\pi,T-2}$	$V_{\pi,T-1}$	$V_{\pi,T}$	
	323,33	0	1
	296,67	0	2
	-38.33	0	3
345,41	200	0	4
	1000	1000	5

Policy evaluation



$V_{\pi,1}$	$V_{\pi,2}$	.	$V_{\pi,T-1}$	$V_{\pi,T}$
870,33	870,33		323,33	0
843,67	843,67		296,67	0
810	810		-38.33	0
831	831		200	0
1000	1000		1000	1000

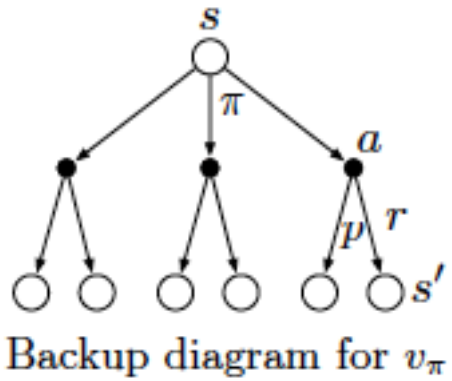
Policy evaluation

Policy evaluation converges to v_π

Again, at convergence equal to solution of infinite-horizon problem

Can we also use v_π to select better actions?

That is, can we define a **policy improvement** step**?**



Backup diagram for v_π

Figure: Sutton&Barto; RL:AI

Policy improvement

Consider π′ equal to π *except* at s_t

Fact:

$$\mathbb{E}_{a_t \sim \pi'(s_t)} q_\pi(s_t, a_t) > v_\pi(s_t) \quad \Rightarrow \quad v_{\pi'}(s) \geq v_\pi(s) \forall s$$

How can we use this to suggest an improved policy?

Policy improvement

Consider π′ equal to π *except* at s_t

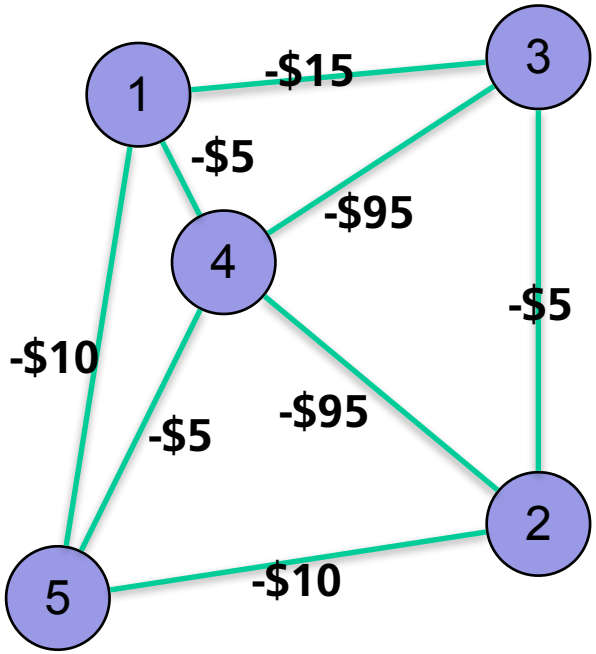
Fact:

$$\mathbb{E}_{a_t \sim \pi'(s_t)} q_\pi(s_t, a_t) > v_\pi(s_t) \quad \Rightarrow \quad v_{\pi'}(s) \geq v_\pi(s) \forall s$$

How can we use this to suggest an improved policy?

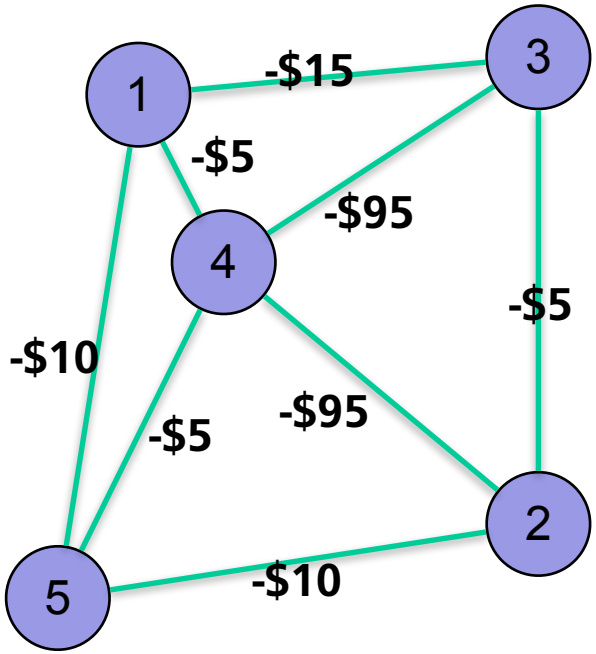
- We can act greedily at s_t (select action maximising $q_\pi(s_t, \cdot)$)
- We can repeat this argument for each possible state

Policy improvement



$V_{\pi,1}$	$V_{\pi,2}$	.	$V_{\pi,T-1}$	$V_{\pi,T}$
870,33	870,33		290	0
843,67	843,67		318.33	0
810	810		-38.33	0
831	831		197.75	0
1000	1000		1000	1000

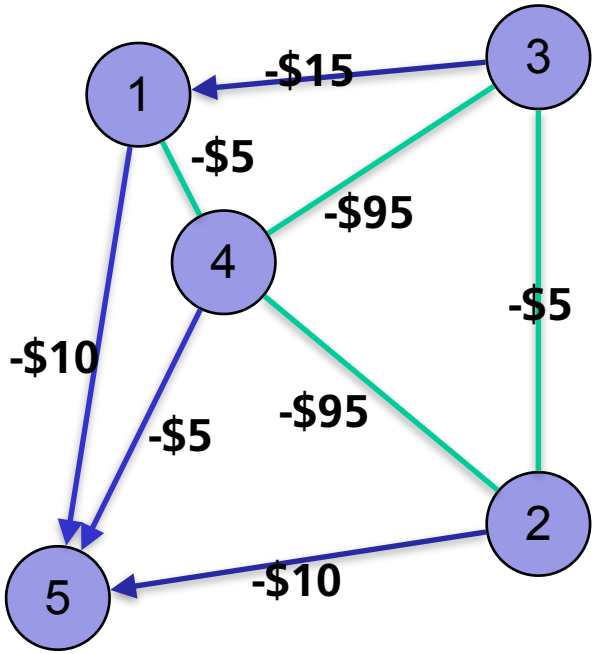
Policy improvement



$V_{\pi,1}$	$V_{\pi,2}$	.	$V_{\pi,T-1}$	$V_{\pi,T}$
870,33	870,33		290	0
843,67	843,67		318.33	0
810	810		-38.33	0
831	831		197.75	0
1000	1000		1000	1000

Policy improvement

The new policy is better than the old one
Can the policy be improved further? Why or why not?



$V_{\pi,1}$	$V_{\pi,2}$	.	$V_{\pi,T-1}$	$V_{\pi,T}$
870,33	870,33		290	0
843,67	843,67		318.33	0
810	810		-38.33	0
831	831		197.75	0
1000	1000		1000	1000

Policy iteration

Iterating the two steps is called policy iteration:

- Policy evaluation
- Policy improvement

Again, this will converge to the optimal policy & value function

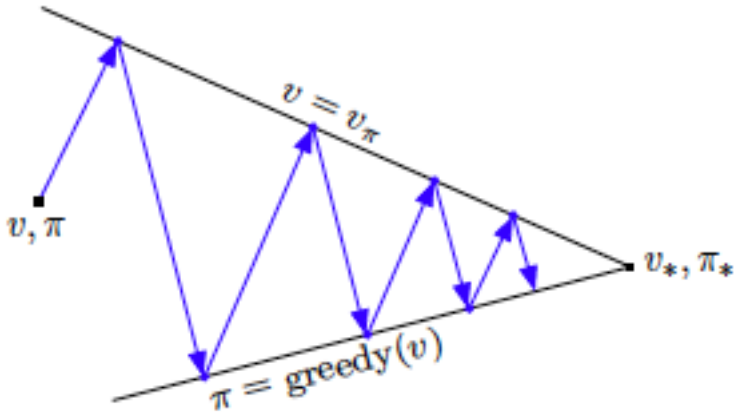
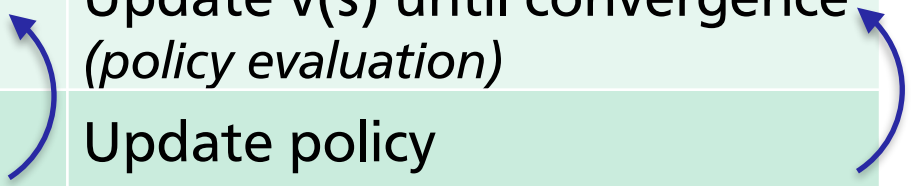


Figure: Sutton&Barto; RL:AI

Policy iteration and value iteration

Value iteration*	Policy iteration
Update v(s) once for each s	Update v(s) until convergence *(policy evaluation)*
Update policy	Update policy *(policy improvement)*



Both examples of generalised policy iteration. Many others:

- Asynchronous updates (update single states)
- Do several value updates (but not until convergence)
- Start from any value function (e.g. value function from last iteration)
- Very similar procedures to compute state-action value function (q)

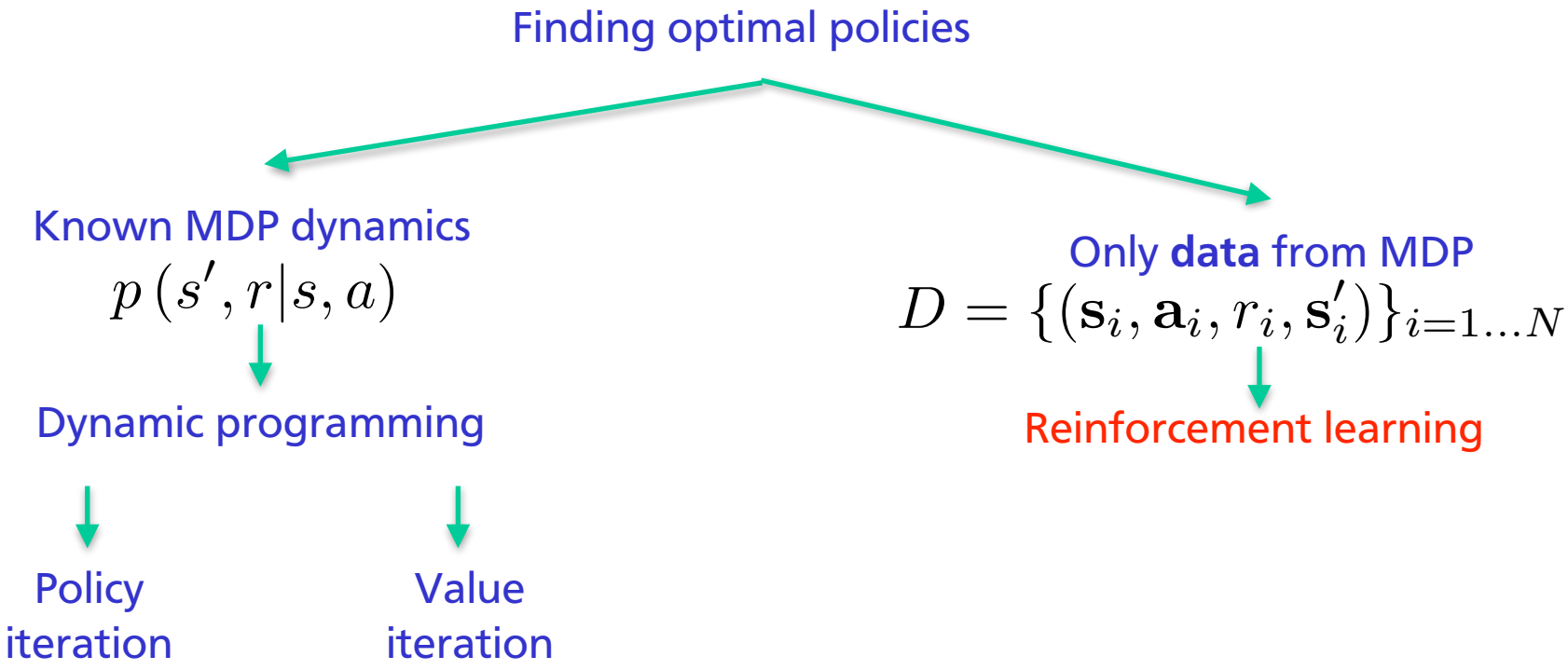
* In algorithms the policy update often happens implicitly, but it can be written out with a separate value- and policy update step as e.g. in this previous slide

Policy iteration and value iteration

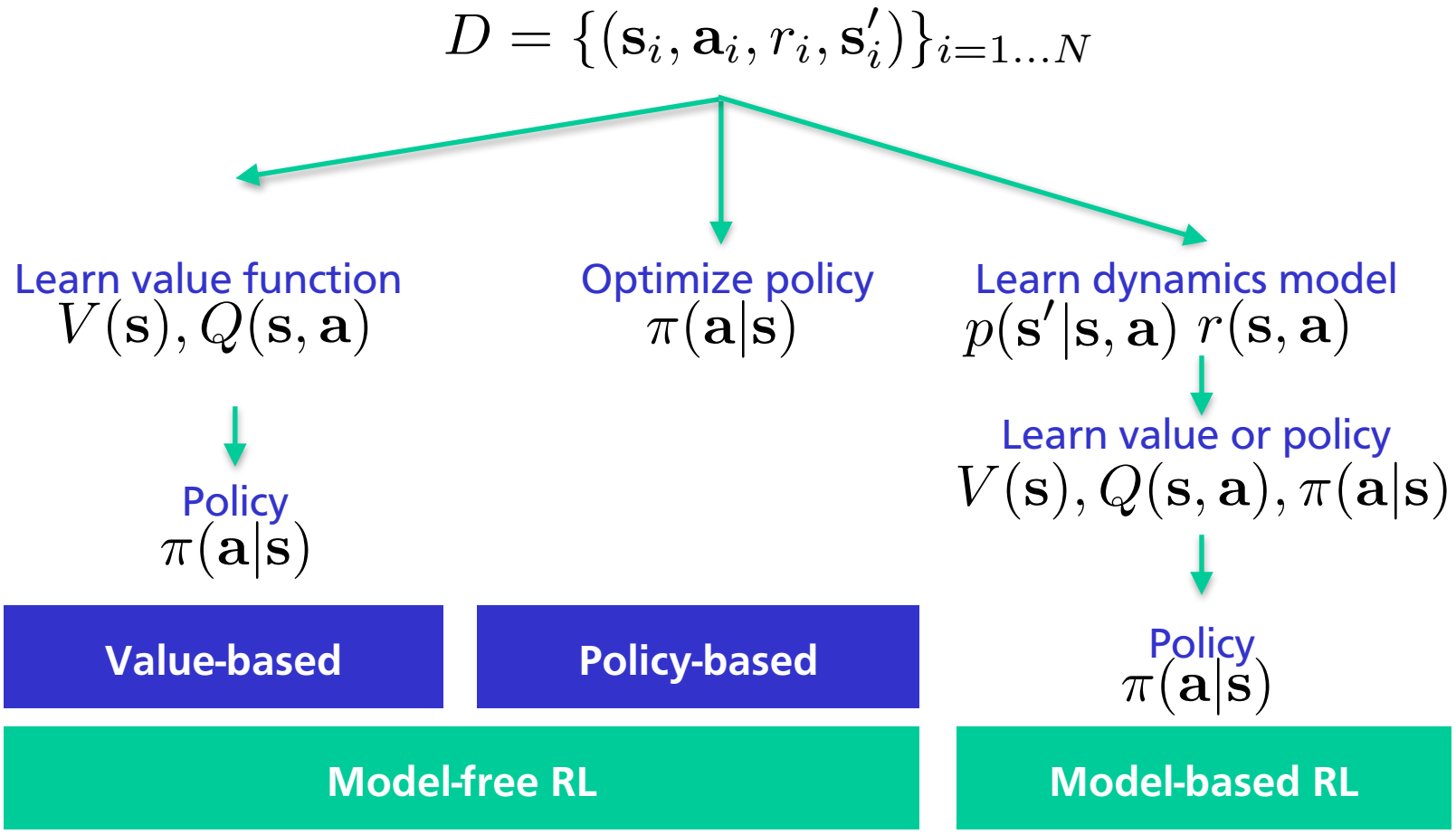
Dynamic programming requires knowing the transition probabilities!

- In RL, we typically assume we do not have them…
- We thus need to **learn** something about the environment
- We could learn the transition probabilities
- Can be more effective to learn the value function directly
- And we can even learn a policy directly, without value function

Big picture

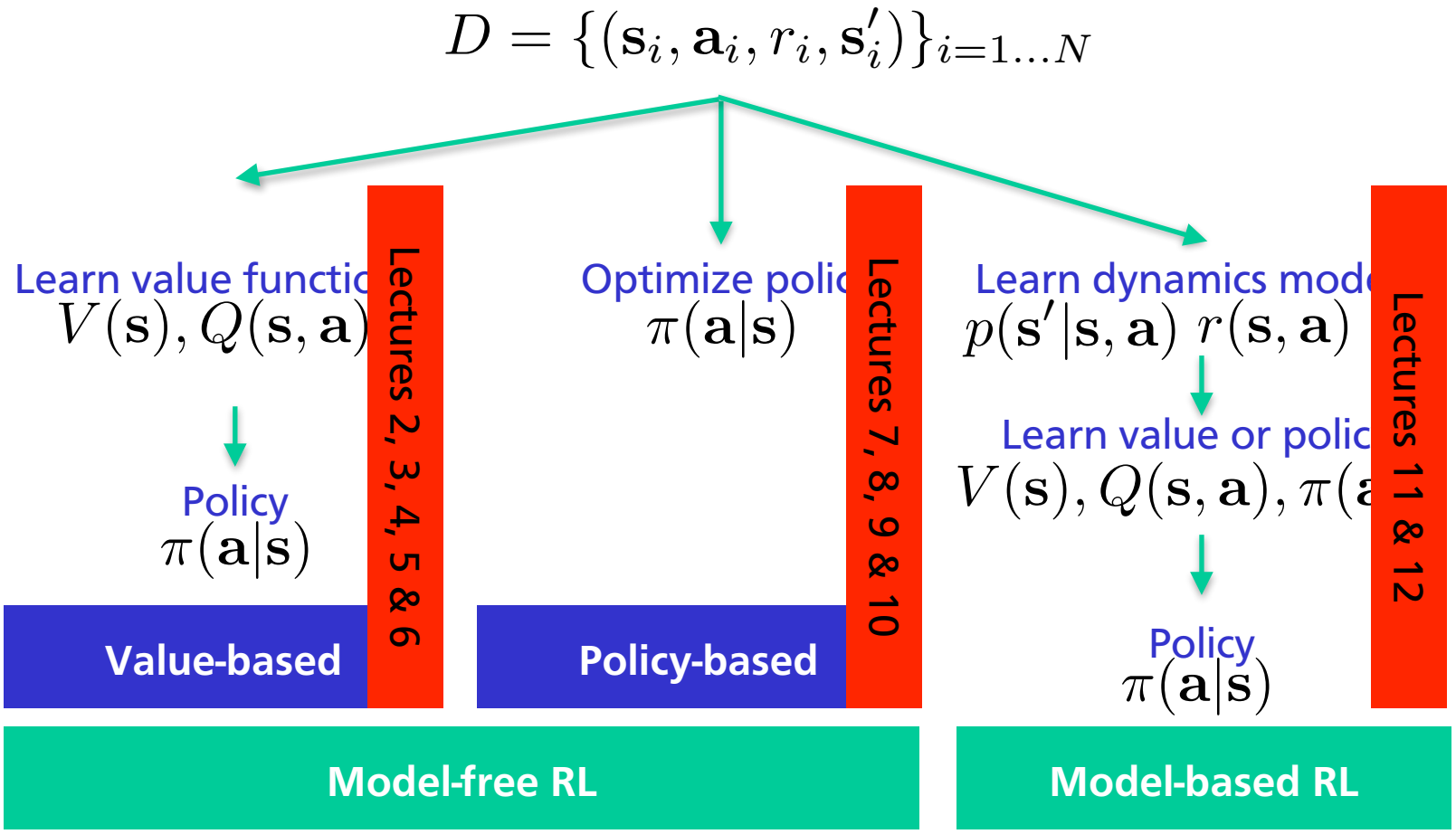

Finding optimal policies
Known MDP dynamics
$p(s', r|s, a)$
Dynamic programming
Policy iteration
Value iteration
Only **data** from MDP
$D = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1...N}$
Reinforcement learning

Big picture: How to learn policies



Thanks to Jan Peters

Big picture: How to learn policies



Thanks to Jan Peters

Monte Carlo prediction

We now want to *learn* value function

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t | S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]$$

The expected (average) return under the policy can be approximated by simply trying the policy multiple times!

(Only if task is episodic)

First-visit Monte-Carlo relies on this insight by estimating the value as the average of return from the first visits (in each episode) to this state

MC prediction

Generate whole trajectory before any update

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

Figure: Sutton&Barto; RL:AI

MC prediction

Generate whole trajectory before any update

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$: ← Loop backwards in time

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

Figure: Sutton&Barto; RL:AI

MC prediction

Generate whole trajectory before any update

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$: ← Loop backwards in time

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

Here: only use the first visit
(Can also do every-visit MC)

Figure: Sutton&Barto; RL:AI

MC prediction

Generate whole trajectory before any update

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$: ← Loop backwards in time

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

Here: only use the first visit
(Can also do every-visit MC)
Need to store all returns?

Figure: Sutton&Barto; RL:AI

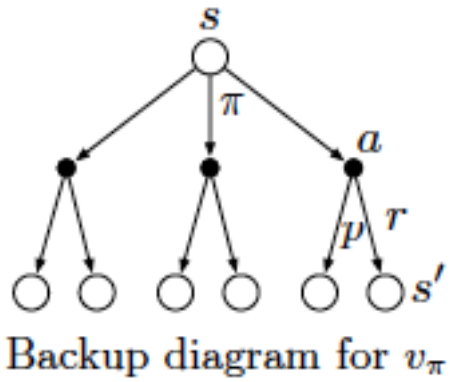
Monte Carlo and DP

Dynamic programming

Width: all next states

Depth: Looks one step ahead

Only possible if transitions known exactly



Backup diagram for v_π

Monte-Carlo

Width: a single possible future

Depth: until the end of episode

Only possible if task is episodic!

Figures: Sutton&Barto; RL:AI

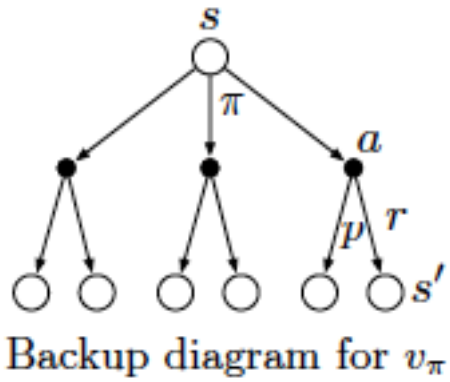
Monte Carlo and DP

Dynamic programming

Width: all next states

Depth: Looks one step ahead

Only possible if transitions known exactly



Backup diagram for v_π

Monte-Carlo

Width: a single possible future

Depth: until the end of episode

Only possible if task is episodic!

In what situations do we not know transition distribution, but are able to obtain experience trajectories?

Figures: Sutton&Barto; RL:AI

Monte Carlo for control

Can we now select actions with the learned value function v?

Monte Carlo for control

Can we now select actions with the learned value function v?

To do that, we again need to know the *dynamics function:*
Which actions lead to states with high v?

Instead, learn a state-action function q.
Very similar to learning v: average return from (s,a) pair visits

From some state s, for which actions will we learn a meaningful q function?

Monte Carlo for control

Guarantee every state-action pair continues to be visited:

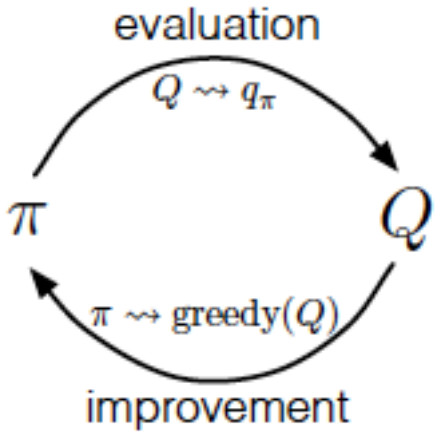
- 'Exploring starts': Start from random (s,a) pair.

Generalized policy iteration suggests:

- If PI step moves Q closer to q_π
- And PU step moves π closer to greedy(Q)
- And all (s,a) pairs continue to be visited

We expect to eventually find optimal policy

Learned Q doesn't equal q_π with finite experience, but: move towards q_π



$$\pi(s) \doteq \arg\max_a q(s,a)$$

Monte Carlo for control

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0

Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \text{argmax}_a \, Q(S_t, a)$

Figure: Sutton&Barto; RL:AI

Monte Carlo for control

‘Exploring starts’ requires starting from random (s,a) pair.

- Not always possible!

Other way to ensure we’ll visit all state-action pairs?

Monte Carlo for control

'Exploring starts' requires starting from random (s,a) pair.

- Not always possible!

Other way to ensure we'll visit all state-action pairs?

- Like in the bandit case, use 'soft' policy that takes each action with non-zero probability (discussed next week!)

Thanks for your attention!

Feedback?

h.c.vanhoof@uva.nl