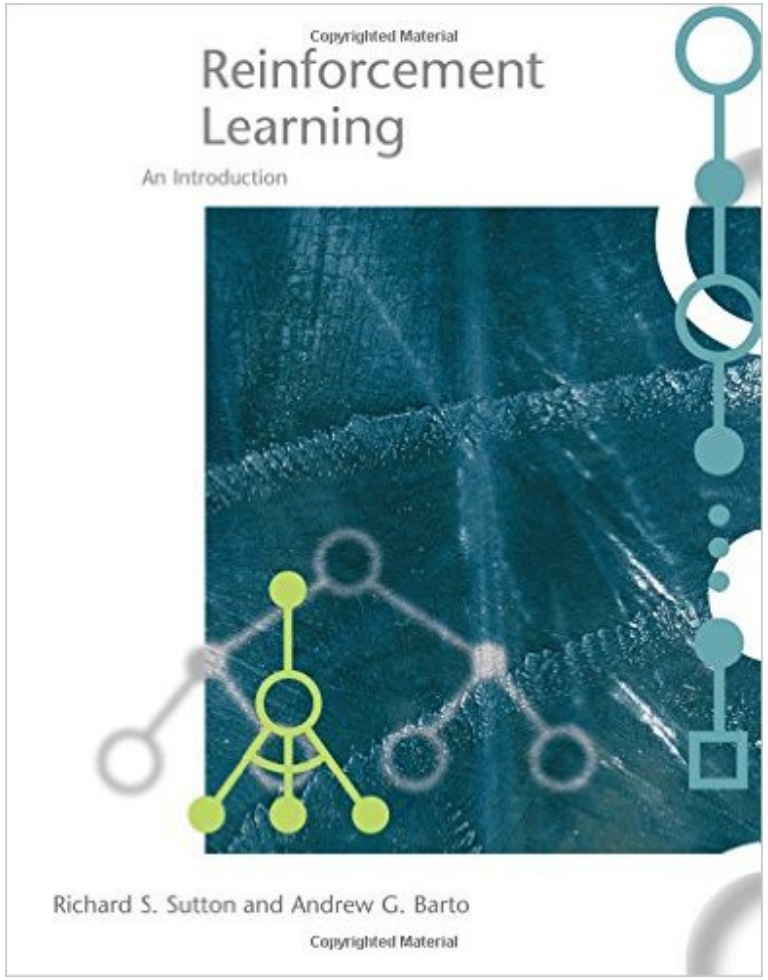
Reinforcement Learning

Tutorial 1: Dynamic Programming and Monte Carlo methods

Gabriela Tavares
Computational and Cognitive Neuroscience Summer School
NYU Shanghai
July 14, 2017

Reinforcement Learning tutorial outline

- Day 1: Dynamic Programming and Monte Carlo
- Day 2: Temporal Difference Learning
- Day 3: Survey of advanced topics, mini projects

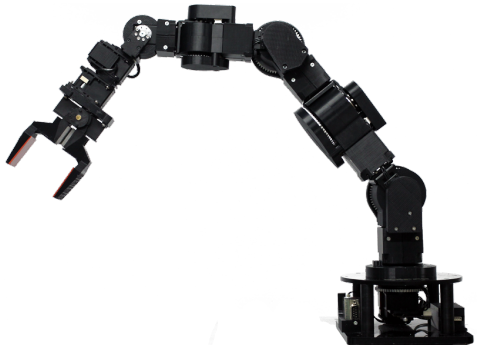


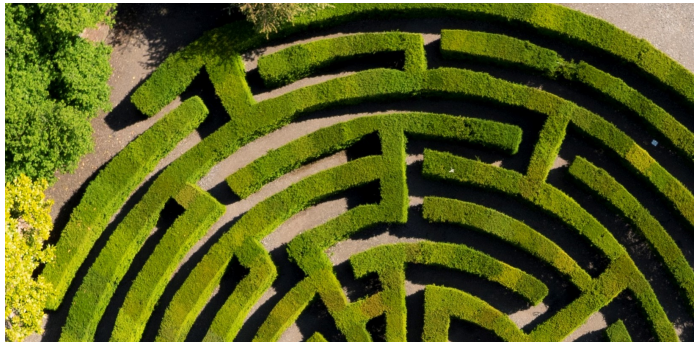
Rich Sutton

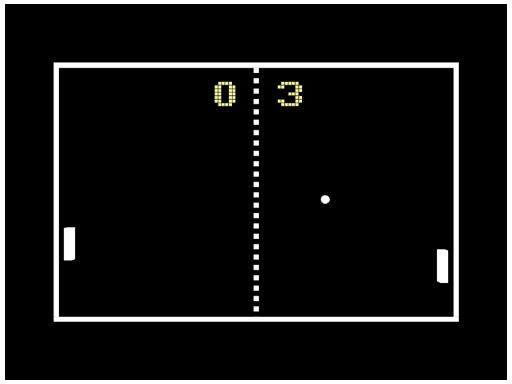
Andy Barto

Uses of Reinforcement Learning

Reinforcement Learning solves problems of control: choosing the right/best action at the right time.



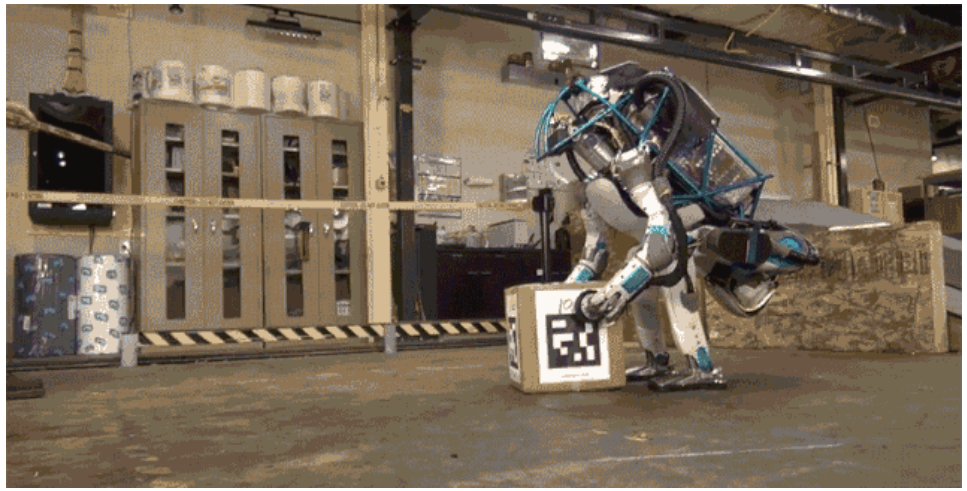




Applications of Reinforcement Learning

Some successful applications include:

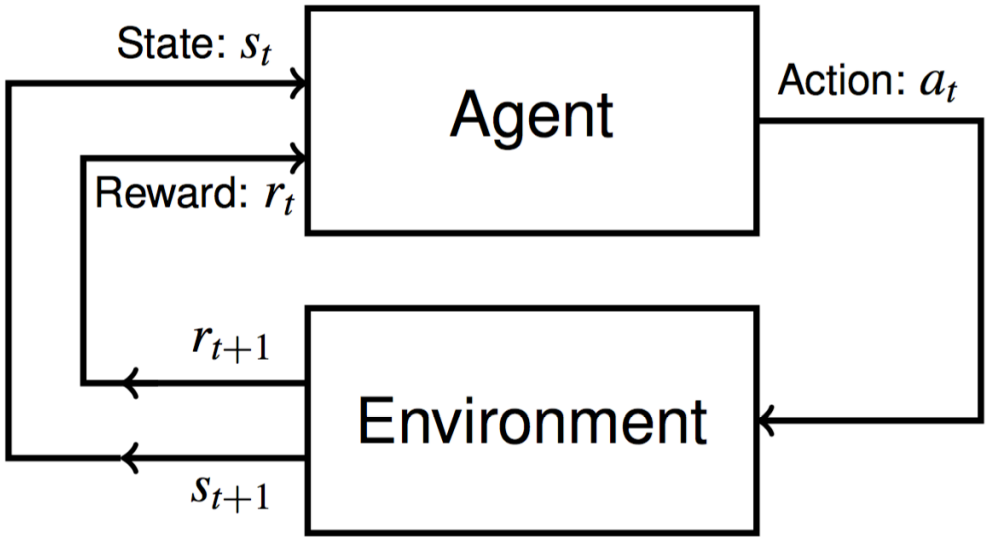
- Backgammon (Tesauro, 1994)
- Inventory management (Van Roy et al., 1996)
- Dynamic channel allocation (Singh and Bertsekas, 1997)
- Elevator scheduling (Crites and Barto, 1998)
- Robocop soccer (Stone and Velloso, 1999)
- Helicopter control (Ng, 2003; Abbeel and Ng, 2006)
- HIV treatment strategies (Ernst et al., 2006)
- Playing Pong (Mnih et al., 2015) and Go (Silver et al., 2016)
- Many robots (navigation, bipedal walking, grasping, switching between skills, …)



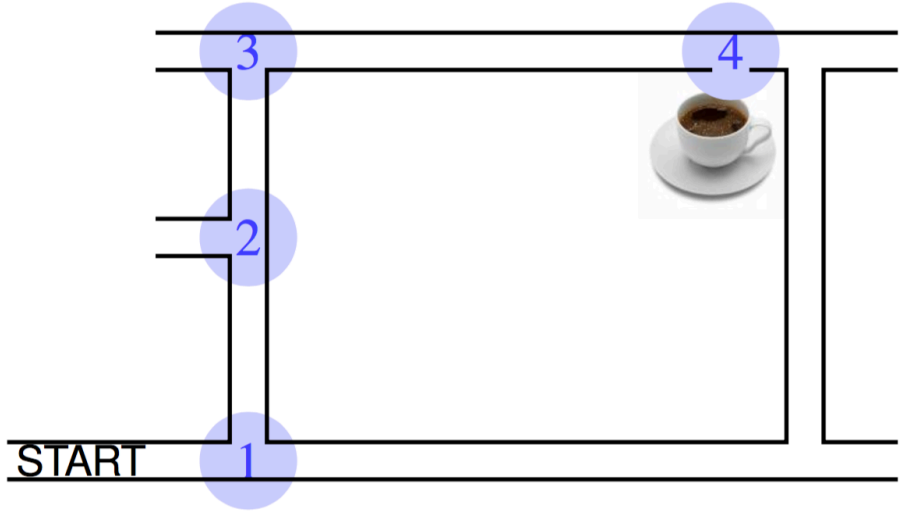
Agent-environment interface

We want to model how agents interact with their environment.

- At each time step t, the environment is in some state (configuration) s_t
- At time t, the agent perceives state s_t and chooses action a_t
- The environment changes to state s_{t+1}
- At time $t+1$, the agent perceives state s_{t+1} and receives reward r_{t+1}



An example: coffee policy



States are 1, 2, 3 and 4.

A policy for this task is:

$\pi(1)$ = turn left

$\pi(2)$ = straight on

$\pi(3)$ = turn right

$\pi(4)$ = go through door

More generally we use a probabilistic policy:

$$\pi_t(s, a) = p(a_t = a | s_t = s)$$

Markov Decision Processes

Markov Decision Processes (MDPs) describe a class of control problems and consist of:

- A set of states, S.
- A set of actions, A.
- A transition function, t, where $t(s, a, s') = p(s'|s, a)$ for $s, s' \in S$ and $a \in A$.
- A reward function, r, where $r(s, a, s')$ is the reward for moving from s to s' under a.
- A policy π takes the current state and gives an action in response.
  - This can be deterministic, i.e., $\pi(s) = a$
  - Or probabilistic, i.e., $\pi(s, a) = p(a|s, \pi)$

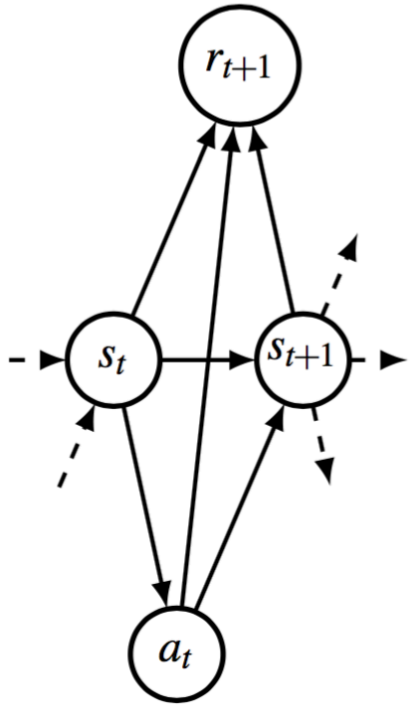
Modeling Markov Decision Processes

The Markov property/assumption:

$$p(s_{t+1}, r_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0) = p(s_{t+1}, r_{t+1} | s_t, a_t)$$

States: s_t, s_{t+1} Actions: a_t Rewards: r_{t+1}

- Variable s_t is the state of the world at time t
- Agent takes action a_t and the world state changes to s_{t+1}
- This transition gives a reward r_{t+1}
- We can also think in terms of costs (i.e. negative rewards)



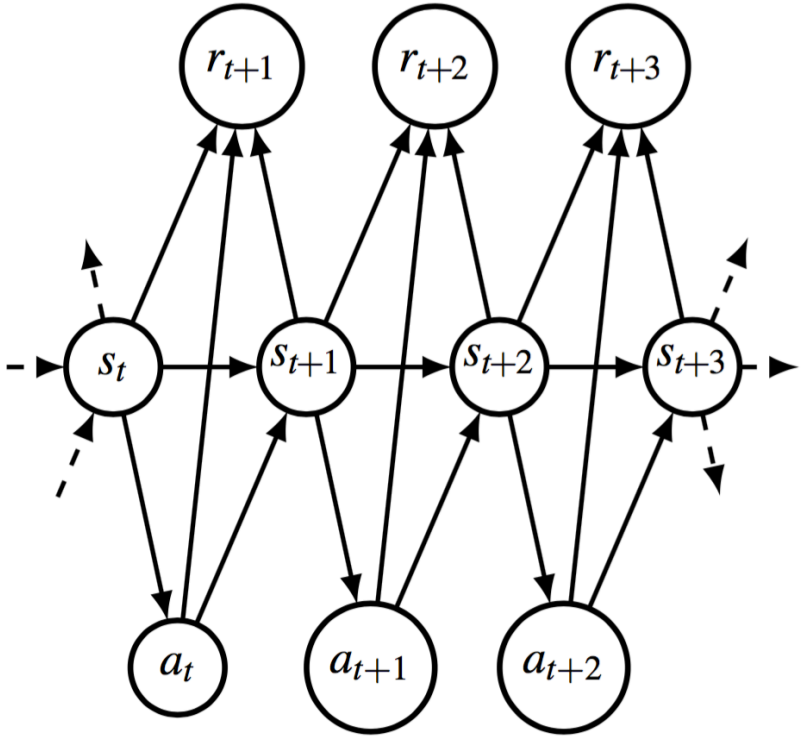
Modeling Markov Decision Processes

Recall that:

$\pi_t(s, a) = p(a_t = a | s_t = s)$

$t(s, a, s') = p(s_{t+1} = s' | s_t = s, a_t = a)$

$r(s, a, s')$ is a reward for going from s to s' under a.



What do we want to achieve?

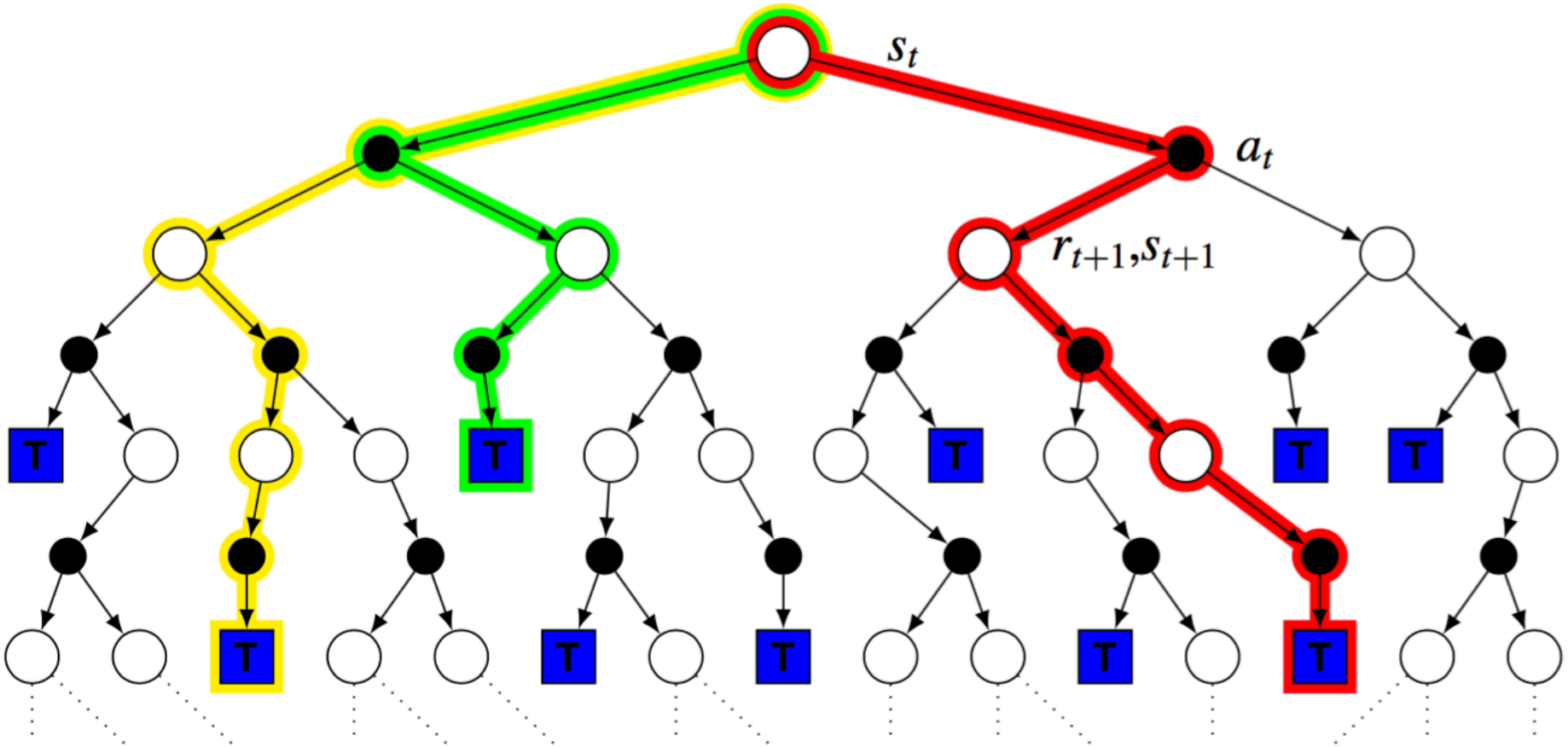
The question we want to answer is: how do we find the best policy for controlling the MDP?

- For this we need to know how to rank or score policies.
- An obvious choice is to obtain the expected performance of a policy. So we must be able to evaluate performance.
- We often need more details than this: we may want to score states and actions as well as policies.

Traces

How do we evaluate policy performance?

We begin by considering traces.



Aggregating rewards

- A **trace** is a state-action-reward sequence, $\tau = s_0, a_0, r_1, s_1, a_1, r_2, \ldots, r_N, s_N = \tau_{[0,N]}$
- A **return** measures the value of τ,

$$R(\tau) = \sum_{k=0}^{N} \gamma_k r_{k+1}$$

- γ_k is the discount factor at time k. Three common choices are:
  - A simple sum, $\gamma_k = 1$.
  - Average reward, $\gamma_k = \frac{1}{N}$.
  - Geometric discount, $\gamma_k = \gamma^k$, with $0 < \gamma < 1$.
- We will use geometric discount:

$$R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$$

Values

- The **expected return** predicts average trace return for policy π:

$$J(\pi) = \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k r_{k+1} \,|\pi\right)$$

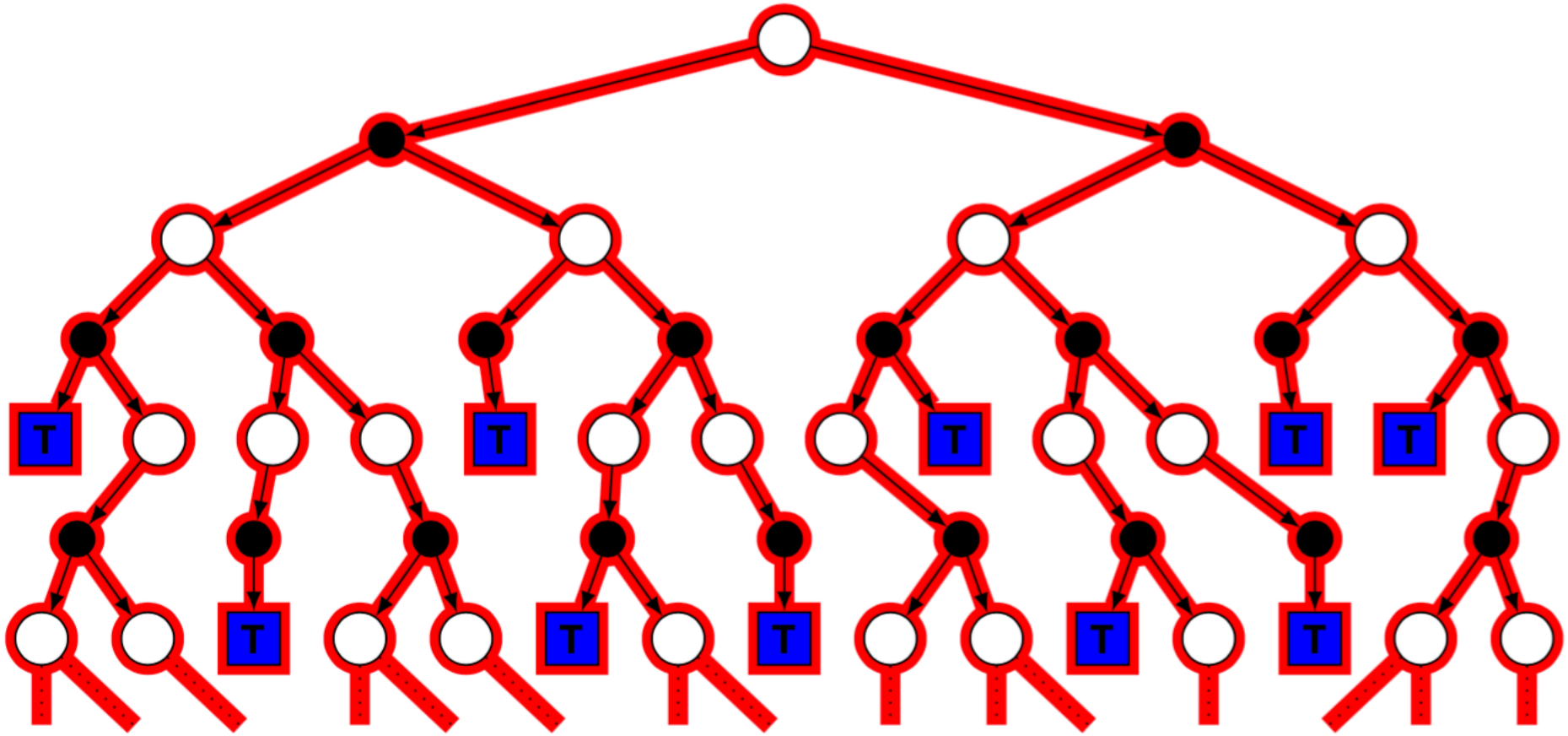
- The **value function** is the expected return given a state:

$$V^{\pi}(s) = \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\pi, s_t = s\right)$$

- The **Q function** (state-action value function) is the expected return given a state and action:

$$Q^{\pi}(s,a) = \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\pi, s_t = s, a_t = a\right)$$

Brute force estimates



Brute force estimates

Naively, we might want to estimate these values by performing a weighted sum over all possible traces.

- Expected return:

$$J(\pi) = \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k r_{k+1} \,|\pi\right) = \sum_{\tau} p(\tau|\pi) R(\tau)$$

- State-value function:

$$V^{\pi}(s) = \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\pi, s_t = s\right) = \sum_{\tau} p(\tau|\pi, s_0 = s) R(\tau)$$

- State-action value function:

$$Q^{\pi}(s,a) = \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\pi, s_t = s, a_t = a\right) = \sum_{\tau} p(\tau|\pi, s_0 = s, a_0 = a) R(\tau)$$

More on traces

Consider the trace $\tau = s_0, a_0, r_1, s_1, a_1, r_2, \ldots, r_N, s_N$. We can relate certain properties of traces using t and π:

$$p_\tau(s_t = s, a_t = a|\pi) = \pi(s, a)\, p(s_t = s|\pi)$$

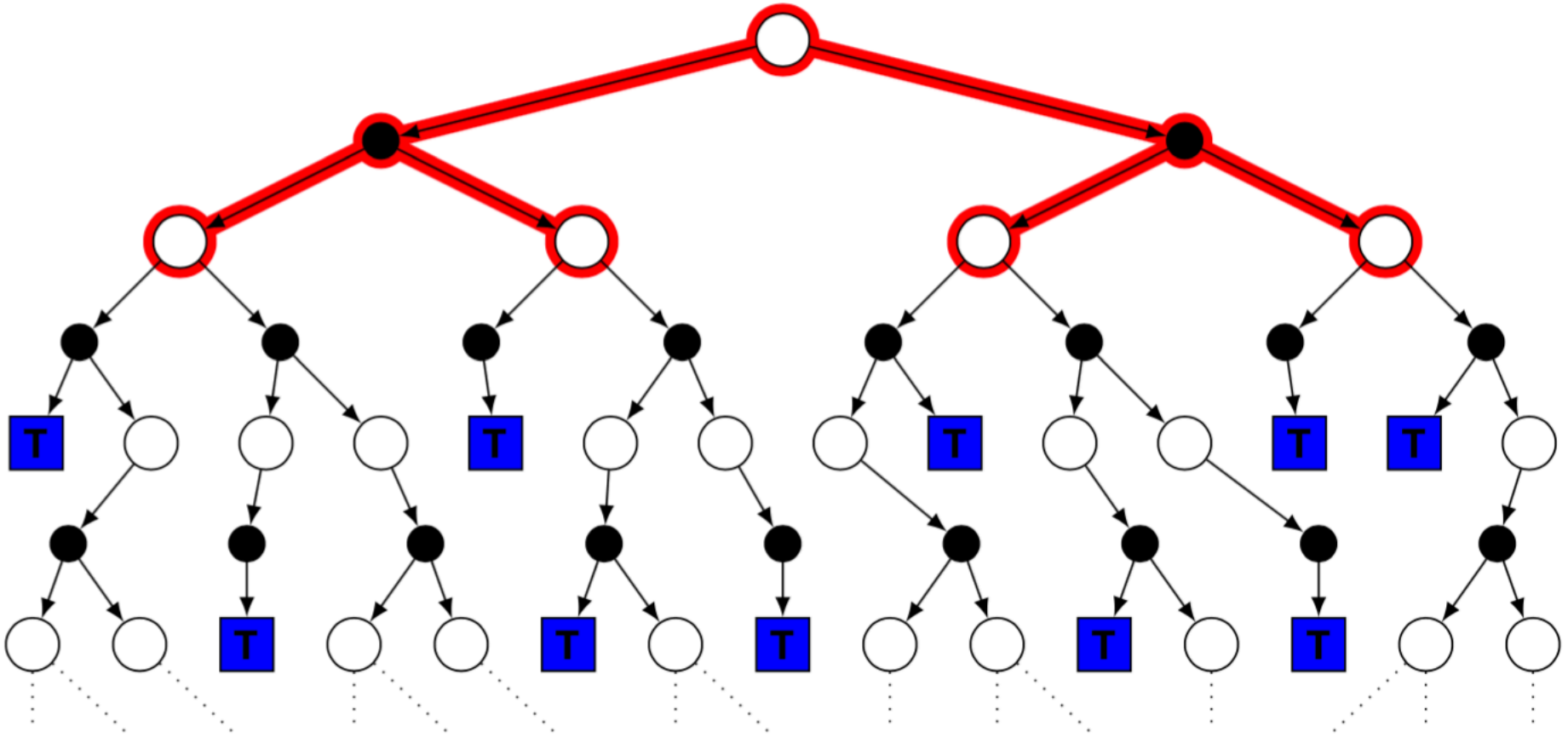
and

$$p_\tau(s_t = s, a_t = a, s_{t+1} = s'|\pi) = t(s, a, s')p(s_t = s, a_t = a|\pi) = t(s, a, s')\pi(s, a)p(s_t = s|\pi)$$

And the probability of a trace:

$$p(\tau|\pi) = p(s_0)\prod_{t=0}^{N-1} p(a_t|s_t)p(s_{t+1}|s_t, a_t) = p(s_0)\prod_{t=0}^{N-1} \pi(s_t, a_t)t(s_t, a_t, s_{t+1})$$

A smarter way



A smarter way

Using the Markov property we can rewrite V^{π}:

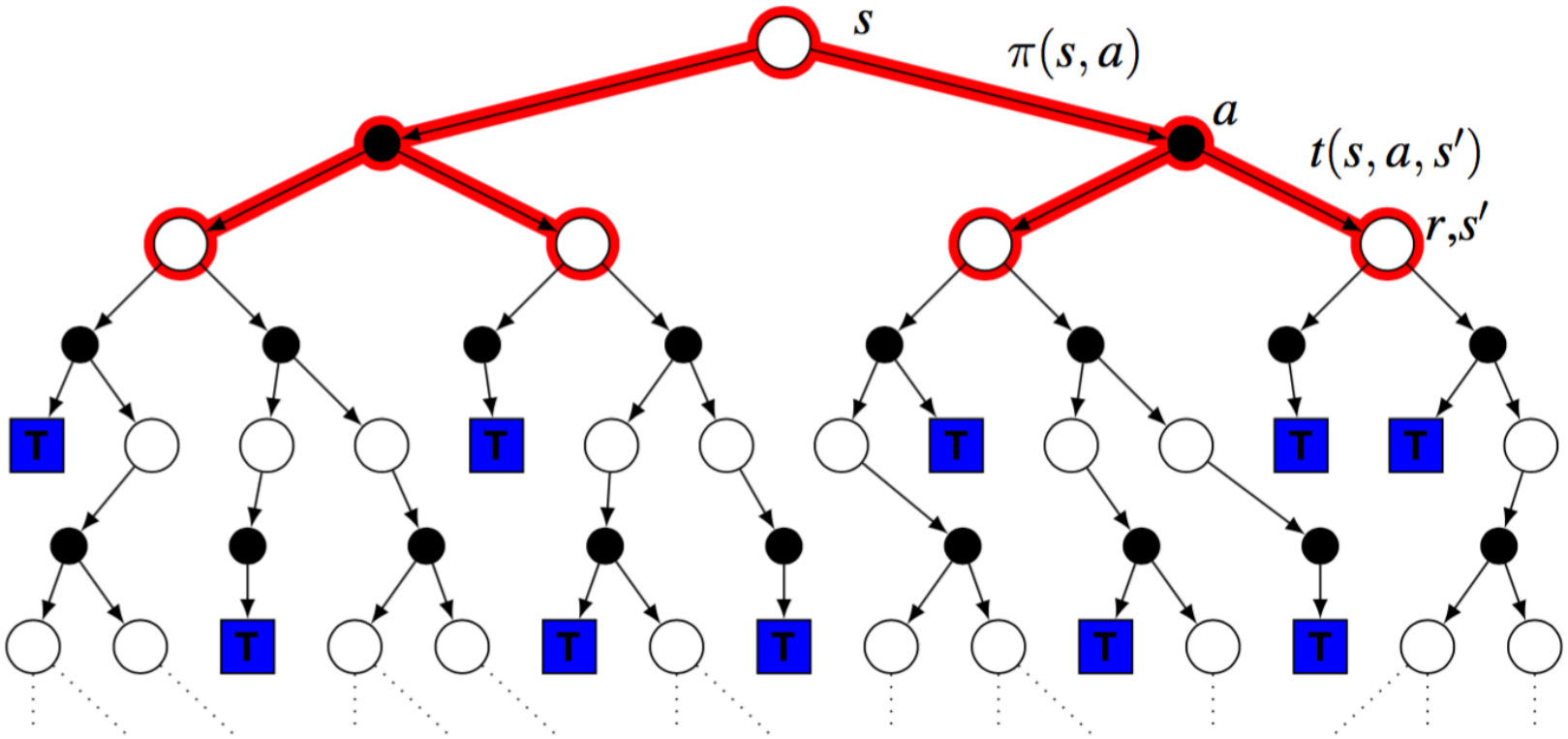
$$V^{\pi}(s) = \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|\pi, s_t = s\right) = \sum_a \pi\,(s,a) \sum_{s'} t(s,a,s')(r(s,a,s') + \gamma V^{\pi}(s'))$$

This is called the **Bellman equation** for fixed π.

We can prove this using the Markov property and our definitions of $V^{\pi}(s)$, $p(\tau|s_0 = s, \pi)$ and $R(\tau)$.

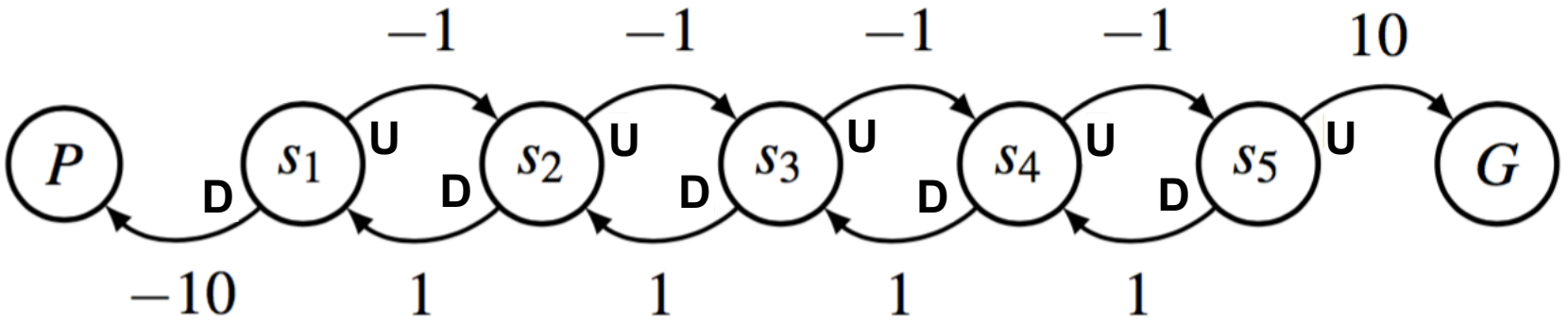
(…but we won't)

Dynamic Programming estimates



The Bellman equation allows us to look just one step ahead in the tree and use the estimates located there.

Example: a very simple MDP (stair climbing)



The start state is s_3. States P and G are absorbing. $\gamma = 0.9$.

- Begin with an unbiased random policy: for all states s, $\pi(s, D) = \pi(s, U) = \frac{1}{2}$.
- We want to obtain the value $V^{\pi}(s)$ at each state s.
- What is the best policy, π^*? How will $V^{\pi}(s)$ help us find π^*?

Estimating state values

How do we estimate V^{π} for some MDP?

- We begin with estimates $\hat{V}(s) = 0$ for all states s.
- We can get improved estimates using updates based on the Bellman equation:

  For all s,

$$\hat{V}'(s) \leftarrow \sum_{a} \pi(s,a) \sum_{s'} t(s,a,s') \left(r(s,a,s') + \gamma \hat{V}(s') \right)$$

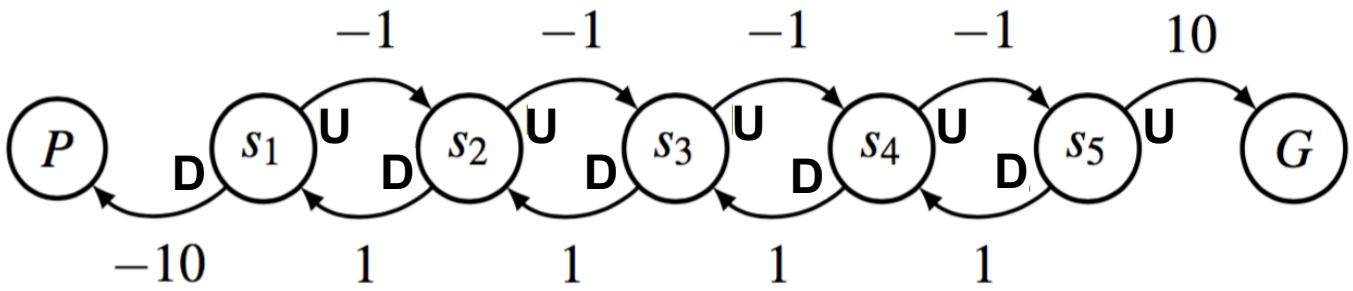
- We set each $\hat{V}(s) \leftarrow \hat{V}'(s)$, and repeat the above update until the desired accuracy.

Policy Evaluation algorithm

1: **procedure** POLICYEVALUATION(π,θ)
2: Initialise $\hat{V}(s) = 0$ for all $s \in S$
3: **repeat**
4: **for all** $s \in S$ **do**
5: $\hat{V}'(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} t(s,a,s') \left(r(s,a,s') + \gamma \hat{V}(s')\right)$
6: $\Delta \leftarrow \max_{s \in S} \mid \hat{V}'(s) - \hat{V}(s) \mid$
7: **for all** $s \in S$ **do**
8: $\hat{V}(s) \leftarrow \hat{V}'(s)$
9: **until** $\Delta < \theta$

Policy Evaluation on stair climbing MDP

Policy evaluation for $\pi(s, D) = \pi(s, U) = \frac{1}{2}$, with $\gamma = 0.9$:



$\hat{V}_0$:	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$\hat{V}_1$:	0.0	−5.5	0.0	0.0	0.0	5.5	0.0
$\hat{V}_2$:	0.0	−5.5	−2.48	0.0	2.48	5.5	0.0
$\hat{V}_3$:	0.0	−6.61	−2.48	0.0	2.48	6.61	0.0
$\hat{V}_4$:	0.0	−6.61	−2.98	0.0	2.98	6.61	0.0
$\hat{V}_\infty$:	0.0	−6.90	−3.10	0.0	3.10	6.90	0.0

More on Policy Evaluation

Policy Evaluation:

- Estimates the value function, i.e. $\forall s$, $\hat{V}(s) \approx V^{\pi}(s)$.
- Requires direct access to transition probabilities and reward function (i.e., the model of the MDP).
- Requires the Markov property.
- Can use any initial values for $\hat{V}_0$.
- Can be performed in-place, i.e., using only one copy of the estimates $\hat{V}$ (this gives a small performance gain).

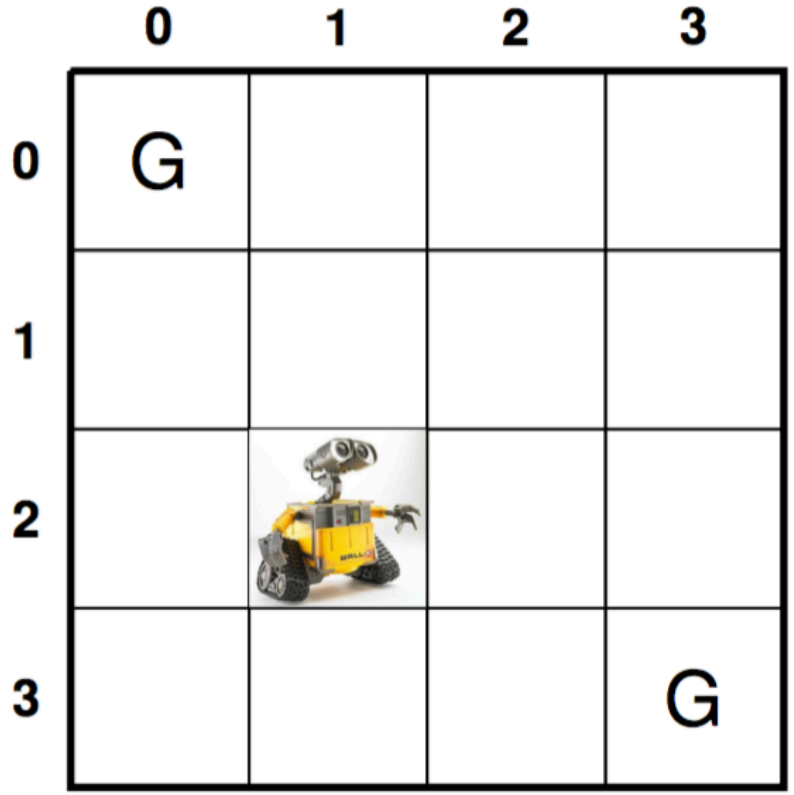
Using $\hat{V}$

A grid world example:

- Actions R, U, L and D
- Action effects are deterministic
- (0,0) and (3,3) are absorbing (goal) states
- $r = -1$ on all transitions
- $\pi(s, a) = \frac{1}{4}$ for all s and a

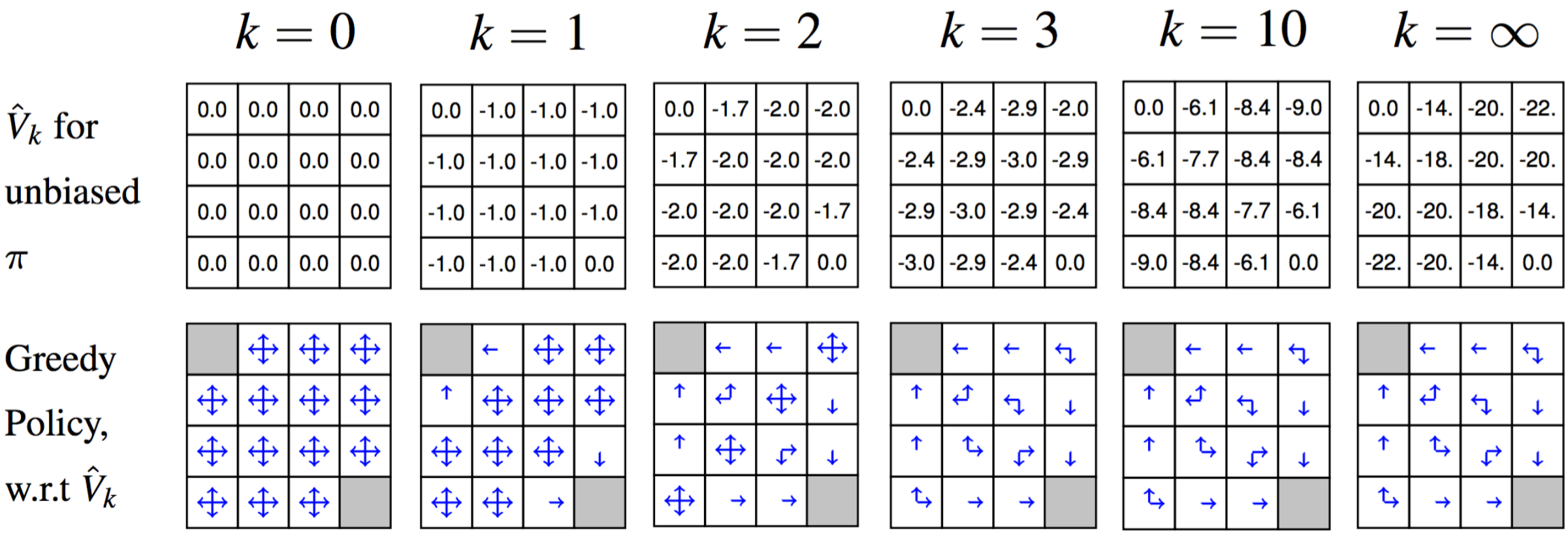
What is the best policy?

Can we use estimates $\hat{V}(s)$ to find it?



(Sutton and Barto)

Policy Evaluation on grid world MDP



(Sutton and Barto)

Policy Iteration algorithm

Using the estimates $\hat{V}$, we can **greedily** choose a better (deterministic) policy, π', where:

$$\pi'(s) = \arg\max_a \mathbb{E}(r_{t+1} + \gamma \hat{V}(s_{t+1}) | s_t = s, a_t = a)$$

$$\pi'(s) = \arg\max_a \sum_{s'} t(s, a, s')(r(s, a, s') + \gamma \hat{V}(s'))$$

We can then alternate between the policy evaluation step, E, and a policy improvement step, I, until there is no further change in the policy:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

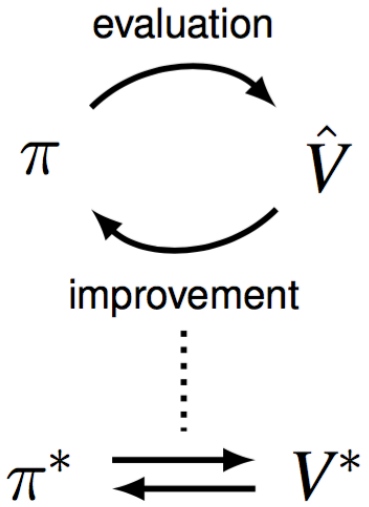
(this is similar to the EM algorithm)

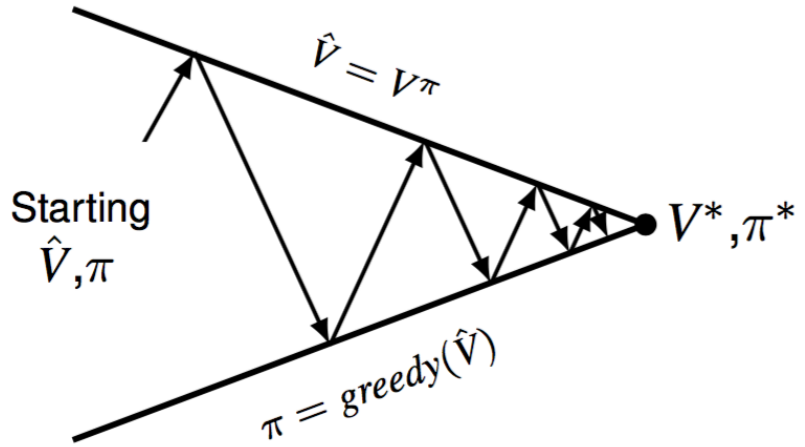
Policy Iteration algorithm

1: **procedure** POLICYITERATION(θ)
2: Initialise $\hat{V}(s) = 0$ and $\pi(s) \in A$ for all $s \in S$.
3: **repeat**
4: $\hat{V} \leftarrow$ POLICYEVALUATION(π,θ)
5: *stable* $\leftarrow$ *true*
6: **for all** $s \in S$ **do**
7: $b \leftarrow \pi(s)$
8: $\pi(s) \leftarrow \arg\max_a \sum_{s'} t(s,a,s') \left(r(s,a,s') + \gamma \hat{V}(s')\right)$
9: **if** $b \neq \pi(s)$ **then**
10: *stable* $\leftarrow$ *false*
11: **until** *stable* $=$ *true*

Dynamic Programming

- The order with which we update states does not matter, as long as we visit every state sufficiently often.
- The Policy Iteration algorithm is a type of Generalized Policy Iteration (GPI).
- Another example is **Value Iteration**, which replaces the policy evaluation step in Policy Iteration with a single backup of each state, making it more efficient.





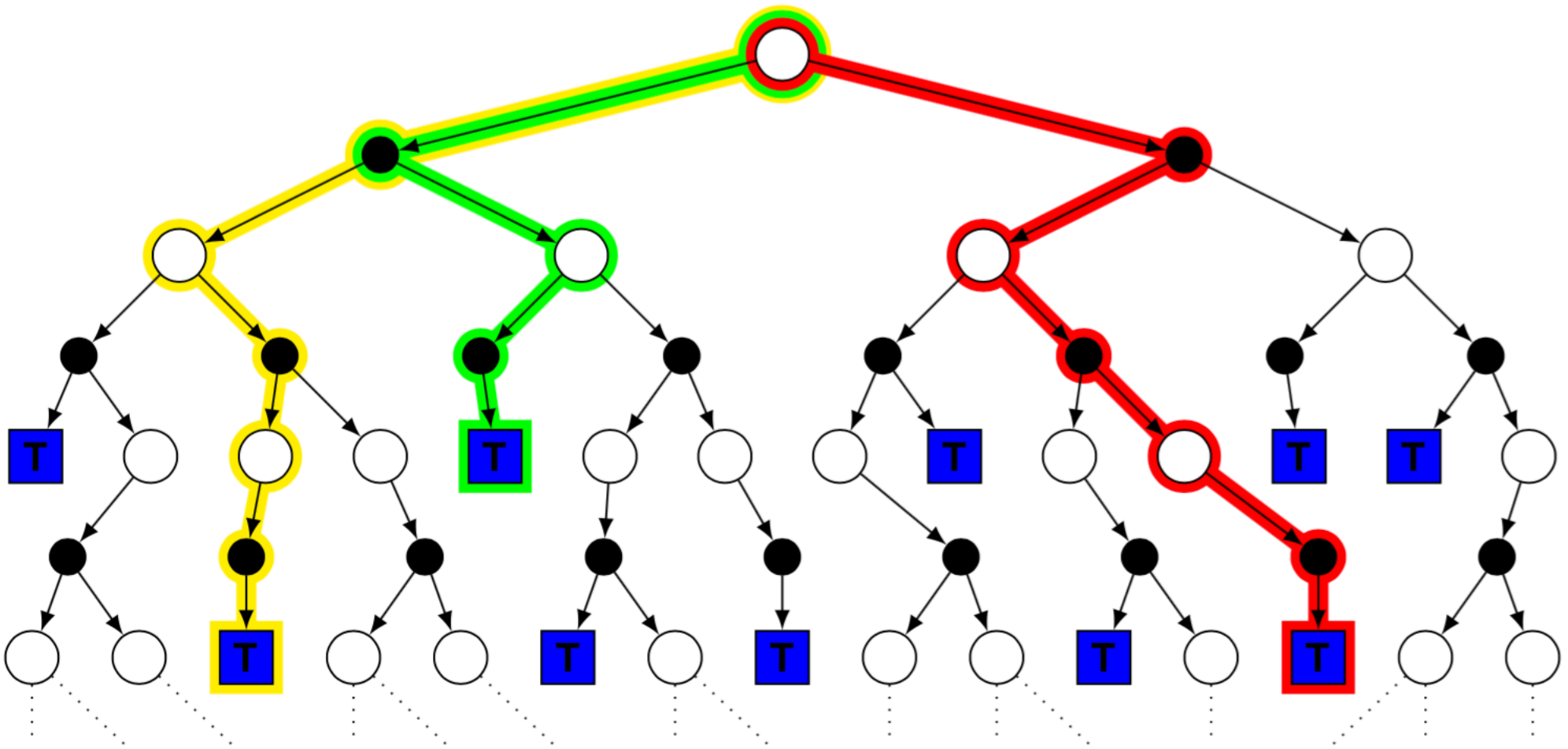
Sampling traces

Is there a better way of estimating $V^{\pi}(s)$?

We can use π to explore the MDP by interacting with it. If we sample a sufficient number of traces $\{\tau_i\}_1^N$ from state s, then

$$V^{\pi}(s) = \sum_{\text{all } \tau} p(\tau|\pi, s_0 = s)R(\tau) \approx \frac{1}{N}\sum_{i=1}^{N} R(\tau_i)$$

Monte Carlo exploration



First-visit Monte Carlo estimation algorithm

1: **procedure** MONTECARLOESTIMATION(π, N)
2: **Init**
3: $Returns(s) \leftarrow$ an empty list, for all $s \in S$.
4: **EndInit**
5: **for** N iterations **do**
6: Get trace, τ, using π.
7: **for all** s appearing in τ **do**
8: $R \leftarrow$ return from first appearance of s in τ.
9: Append R to $Returns(s)$
10: **for all** $s \in S$ **do**
11: $\hat{V}(s) \leftarrow$ average($Returns(s)$)
12: **return** $\hat{V}$

The game of blackjack

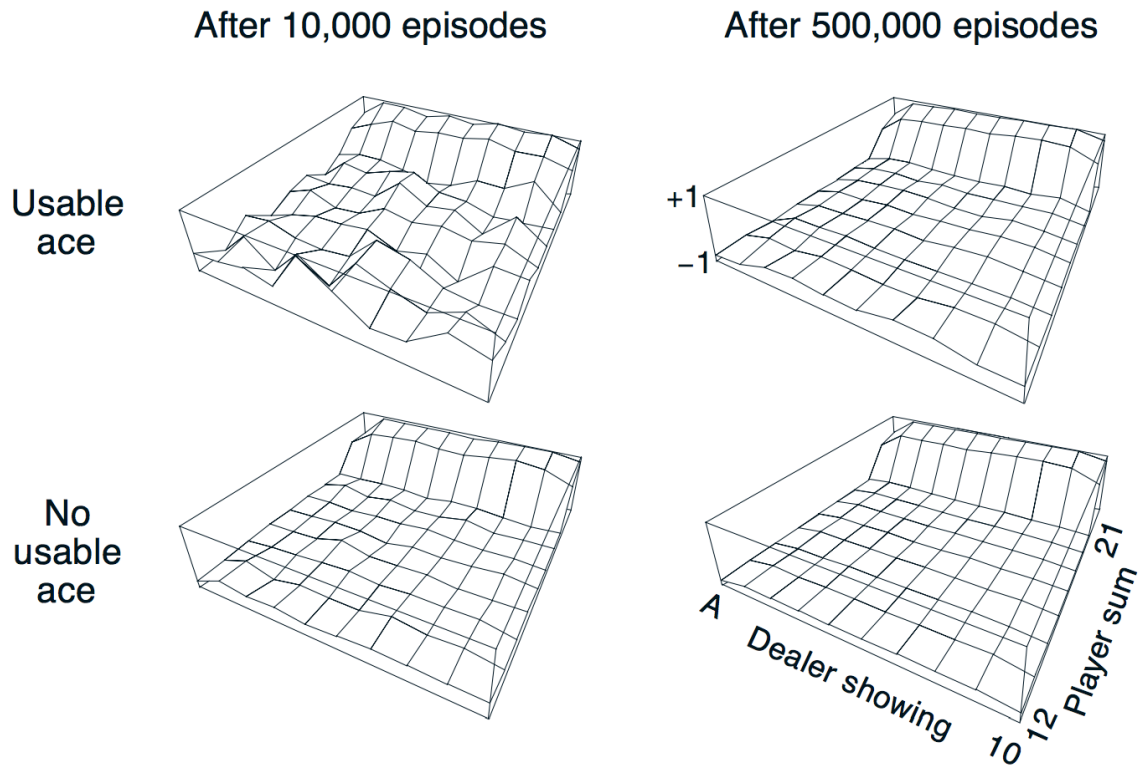
Rules:

- Play against the dealer.
- A hand's value is the sum of the cards' values, with J = Q = K = 10 and A = 1 or 11.
- You want to get closer to 21 than the dealer, but no higher or you bust.
- Start with 2 cards. Actions are hit (get another card) or stick (end your round).

Example of Monte Carlo estimation

Consider the game of blackjack. Is this an MDP?

We can use Monte Carlo estimation with a policy that only sticks on 20 or 21.

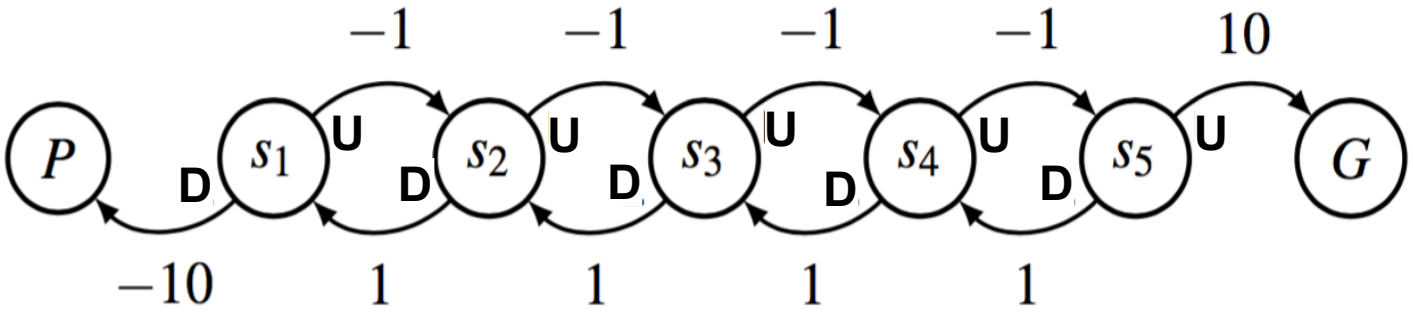


(Sutton and Barto)

First-visit Monte Carlo estimation on stair climbing MDP

MC estimation for our stair climbing MDP with $\pi(s, D) = \pi(s, U) = \frac{1}{2}$ and $\gamma = 0.9$.

First generate a trace, for instance: $\tau = s_3, U, -1, s_4, D, 1, s_3, U, -1, s_4, U, -1, s_5, U, 10, G$



$\hat{V}_0$:	0.0	0.0	0.0	0.0	0.0
$\hat{V}_1$:	0.0	0.0	4.92	6.58	10.

and after many traces…

$\hat{V}_\infty$:	−6.90	−3.10	0.0	3.10	6.90

Monte Carlo overview

- Methods that collect traces by exploration are collectively called Monte Carlo.
- There is no need to access the model of the MDP.
- Can be first state visit or every state visit.

How do we update our policy (choose better actions)?

Another way to choose actions

The Q function (or state-action value function) is the expected return given a state and action:

$$Q^{\pi}(s,a) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^{k} r_{t+1} \,|\pi, s_0 = s, a_0 = a\right)$$

Relating the functions V^{π} and Q^{π}:

$$V^{\pi}(s) = \sum_{a} \pi(s,a)\, Q^{\pi}(s,a)$$

$$Q^{\pi}(s,a) = \sum_{s'} t(s,a,s')(r(s,a,s') + \gamma\, V^{\pi}(s'))$$

Using the state-action value function

- The Q function can be used to choose the best action at a particular state.
- We can define the **greedy** policy as:

$$\pi_{\text{greedy}}(s) = \arg\max_a Q^{\pi}(s, a)$$

- We no longer need the model of the MDP, just the Q function.
- And we can learn estimates $\hat{Q}$ with Monte Carlo, in the same way as we learned $\hat{V}$.

First visit Monte Carlo Q estimation

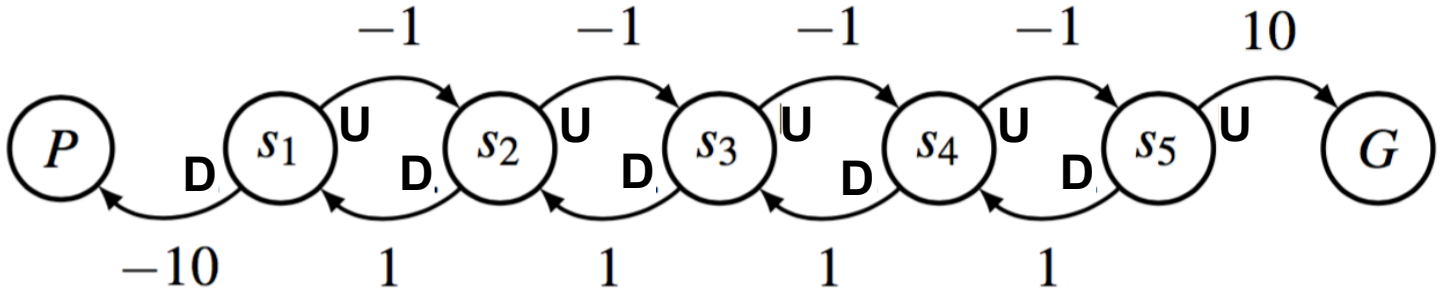
State-action value estimation is almost identical to state value estimation.

1: **procedure** MONTECARLOESTIMATION(π, N)
2: **Init**
3: $Returns(s, a) \leftarrow$ an empty list, for all $s \in S$ and $a \in A$.
4: **EndInit**
5: **for** N iterations **do**
6: Get trace, τ, using π.
7: **for all** (s,a) appearing in τ **do**
8: $R \leftarrow$ return from first appearance of (s, a) in τ.
9: Append R to $Returns(s, a)$
10: **for all** $s \in S$ and $a \in A$ **do**
11: $\hat{Q}(s, a) \leftarrow$ average($Returns(s, a)$)
12: **return** $\hat{Q}$

Monte Carlo Q estimation

On stair climbing MDP, with $\pi(s, D) = \pi(s, U) = \frac{1}{2}$, $\gamma = 0.9$, and first trace

$$\tau = s_3, U, -1, s_4, D, 1, s_3, U, -1, s_4, U, -1, s_5, U, 10, G$$



	(D,U)	(D,U)	(D,U)	(D,U)	(D,U)
$\hat{Q}_0$:	$(0,0)$	$(0,0)$	$(0,0)$	$(0,0)$	$(0,0)$
$\hat{Q}_1$:	$(0,0)$	$(0,0)$	$(0, 4.92)$	$(6.58, 8.)$	$(0, 10.)$

and after many traces…

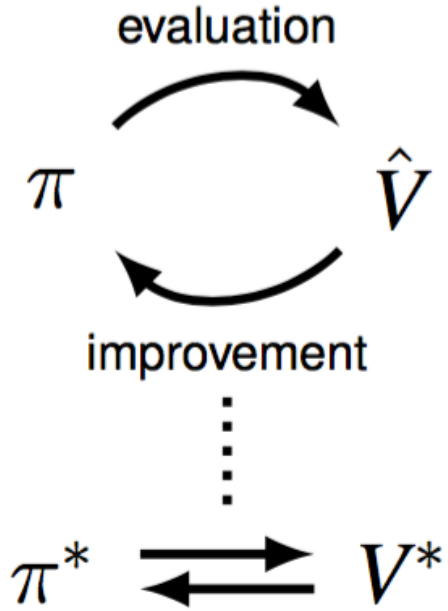
$\hat{Q}_\infty$:	(-10., -3.8)	(-5.2, -1.0)	(-1.8, 1.8)	(1.0, 5.2)	(3.8, 10.)

What is the greedy policy w.r.t. $\hat{Q}_\infty$?

What if we had started with policy $\pi(s) = D$ for all $s \in S$?

A return to Policy Iteration

- Policy Iteration (repeated estimation and improvement) can work for Q values, just as it works for V values.
- But if we use Monte Carlo estimation, we must be careful that we continue to explore all states and actions. Why?
- And why is this a problem if we are using greedy policies?



ε-soft policies

Consider a policy where at each state visit, we follow a deterministic policy with probability $(1-\varepsilon)$, $0 < \varepsilon \ll 1$, and act randomly with probability ε.

- For any ε-soft policy, π,

$$\pi(s,a) = \begin{cases} 1-\varepsilon+\dfrac{\varepsilon}{|A|} & \text{the dominant action, } a\text{, at } s \\ \dfrac{\varepsilon}{|A|} & \text{otherwise} \end{cases}$$

- All ε-soft policies are guaranteed to explore all states and actions.
- Dominant actions are taken far more often.

ε-greedy policies

The ε-greedy policy is the closest ε-soft policy to a greedy policy.

- For $\hat{Q}$, the ε-greedy policy, $\pi = \varepsilon\text{−greedy}(\hat{Q})$, is given by

$$\pi(s,a) = \begin{cases} 1 - \varepsilon + \dfrac{\varepsilon}{|A|} & \text{for } a = \arg\max_{a'} \hat{Q}(s,a') \\ \dfrac{\varepsilon}{|A|} & \text{otherwise} \end{cases}$$

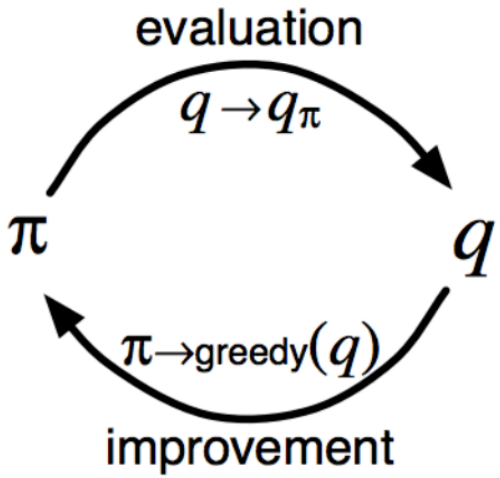
- We can use Policy Improvement with ε-greedy policies to get the best such policy $\tilde{\pi}^*$ (we write $\tilde{Q}^{\tilde{\pi}^*} = \tilde{Q}^*$).
- If ε is small, then the absolute best policy is $\pi^* = \text{greedy}(\tilde{Q}^*)$.

ε-soft Policy Iteration

Interleave Monte Carlo estimation, E, with periodic ε-greedy policy improvement, I_ε, to find the best ε-greedy policy, $\tilde{\pi}^*$.

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{I_\varepsilon} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{I_\varepsilon} \dots \xrightarrow{I_\varepsilon} \tilde{\pi}^* \xrightarrow{E} \tilde{Q}^* \xrightarrow{I} \pi^*$$

Methods that follow the policy they are estimating are known as **on-policy**.

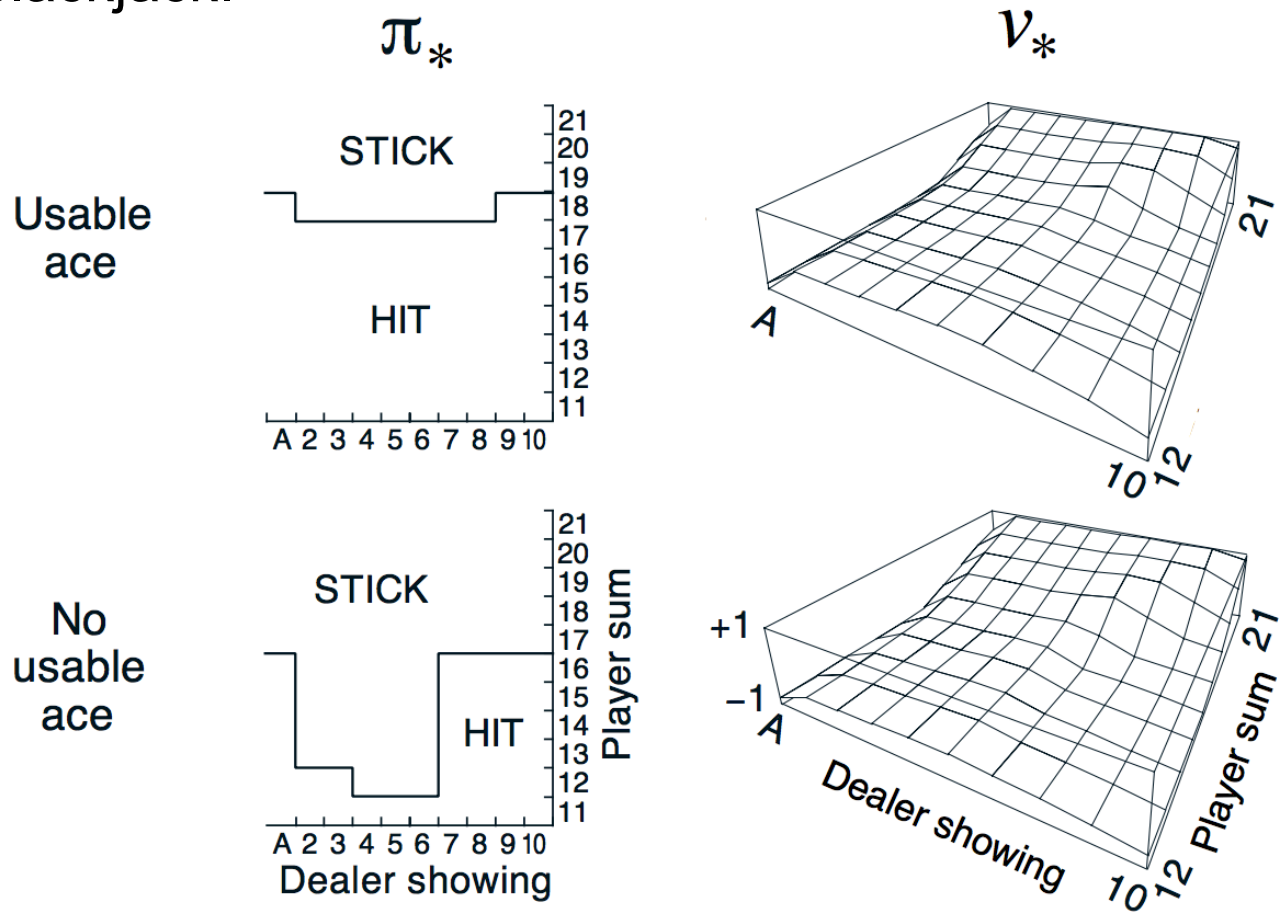


On-policy Monte Carlo batch optimization

1: **procedure** MONTECARLOBATCHOPTIMISATION(n, N)
2: **Init**
3: $\hat{Q}(s, a) \leftarrow$ arbitrary value, for all $s \in S$, $a \in A$.
4: $\pi \leftarrow \varepsilon$-*greedy*$(\hat{Q})$
5: **EndInit**
6: **for** n batches **do**
7: $\hat{Q} \leftarrow$ MONTECARLOESTIMATION(π, N)
8: $\pi \leftarrow \varepsilon$-*greedy*$(\hat{Q})$
9: **return** *greedy*$(\hat{Q})$

Example of Monte Carlo optimization

We can use Monte Carlo optimization to find the optimal policy in the game of blackjack.



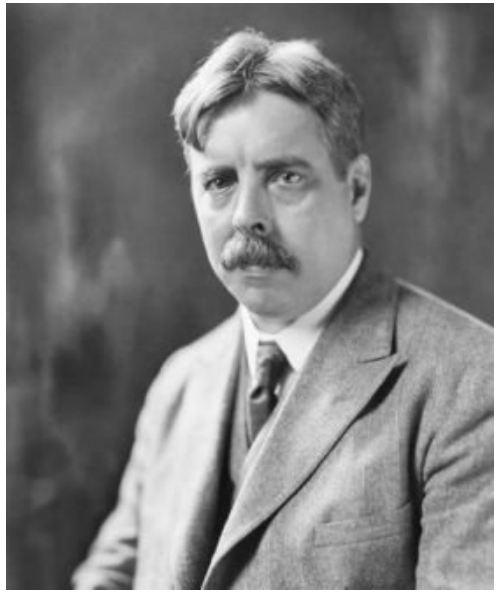
(Sutton and Barto)

Summary

- Markov Decision Processes (MDPs) describe a class of control problems.
- The state (state-action) value function predicts the future return from a given state (and action).
- Policy Evaluation uses Bellman updates to estimate the value function.
- Policy Improvement (greedily) chooses a new policy based on the value function.
- Policy Iteration combines these steps to converge on the optimal policy (an example of GPI).

Summary

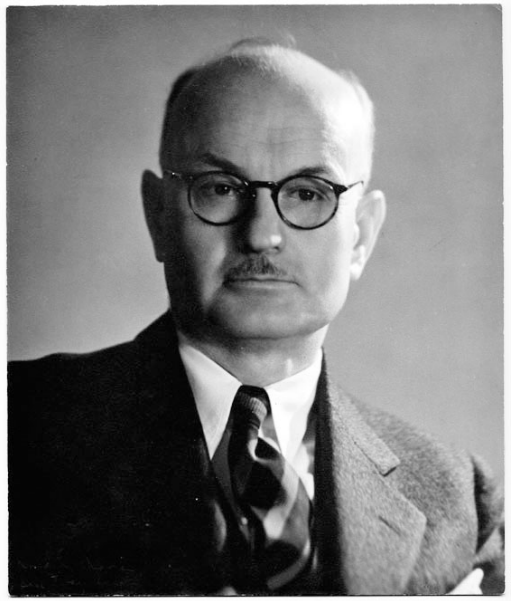
- Monte Carlo estimation can also be used for policy evaluation.
- Monte Carlo estimation learns $\hat{Q}$ (or $\hat{V}$) from experience.
- The algorithm must continually explore all states and actions.
- ε-soft policies can be used for on-policy learning.
- Note: we can also perform off-policy learning, meaning we use one fixed policy to explore the environment, but optimize a different policy (more on this tomorrow).



Edward Thorndike

vs.

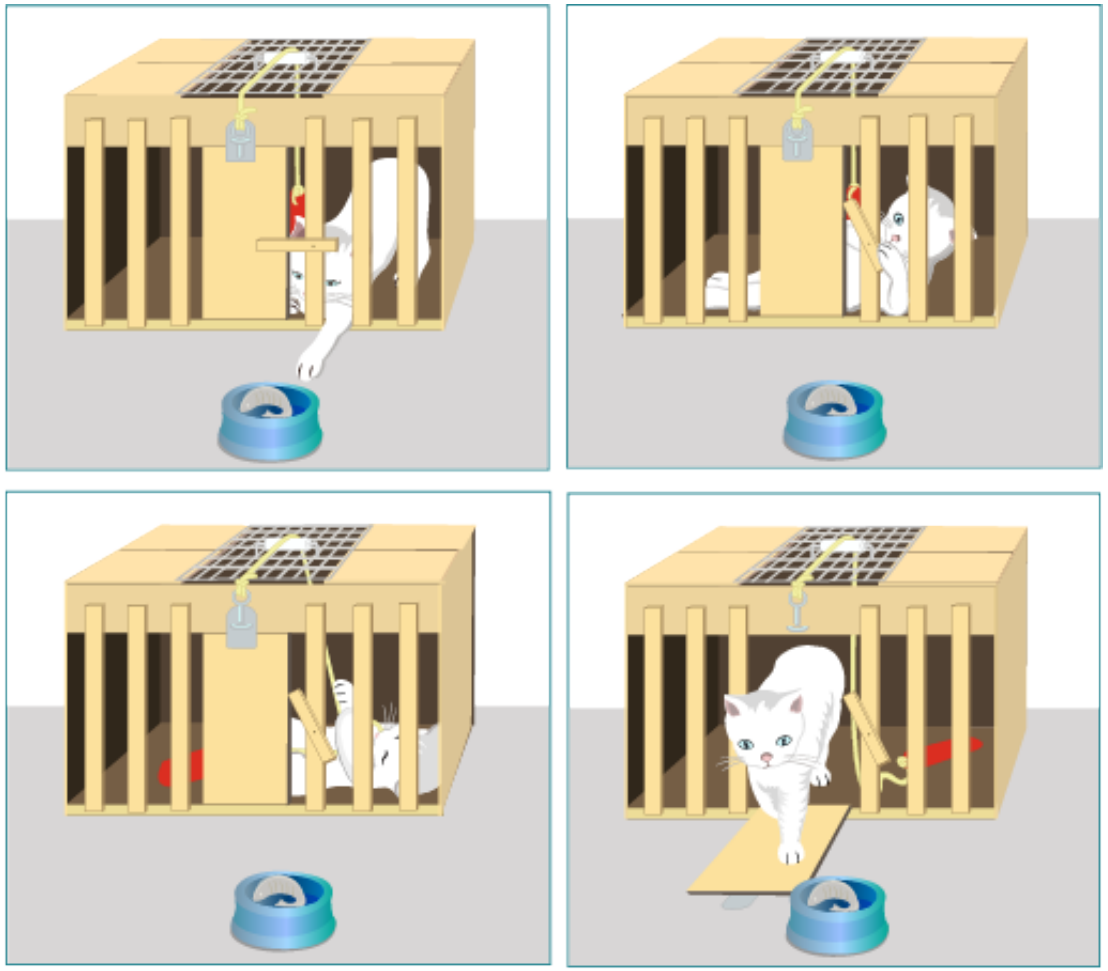
Edward Tolman



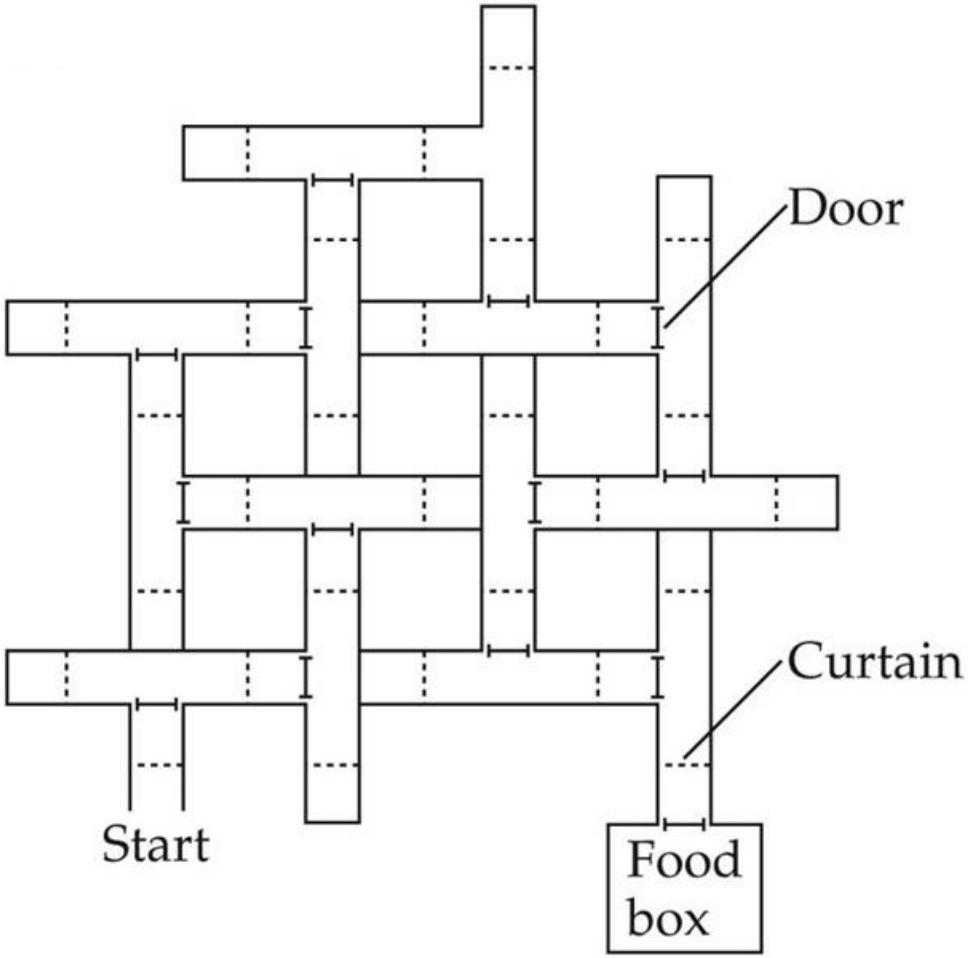
- Behaviorist view
- Operant conditioning
- “Law of effect”
- Stimulus-response habits
- No reward anticipation

- Cognitive view
- Goal-driven behavior
- Cognitive maps of the spatial environment
- “Expectancies” of reward
- Latent learning

Thorndike's box



Tolman's maze



结束