Markov Decision Process

Course: Symbolic AI, Leiden University

Lecturer: Thomas Moerland

Content

1. Markov Decision Process definition

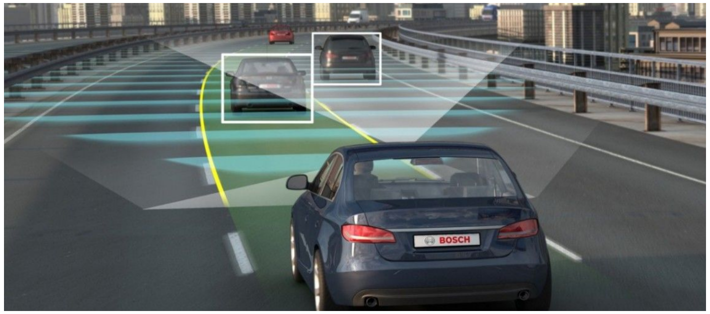
2. Cumulative reward and value

Break

3. Bellman Equation

4. Dynamic Programming

Sequential decision making



Many key problems in AI

Sequential decision making

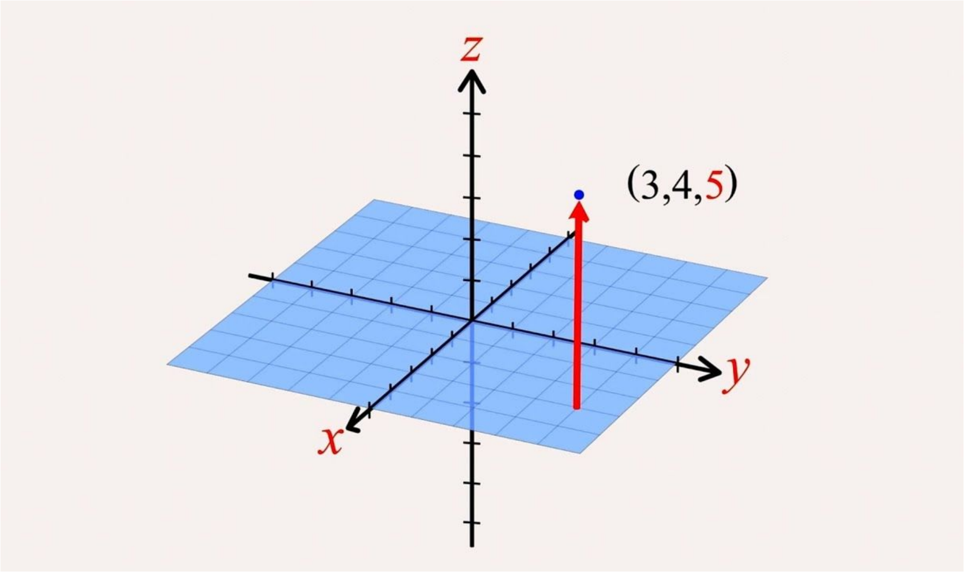
Many recent AI breakthroughs use this formulation

Recap

Recap: Symbolic AI & Search

Search = central theme in AI

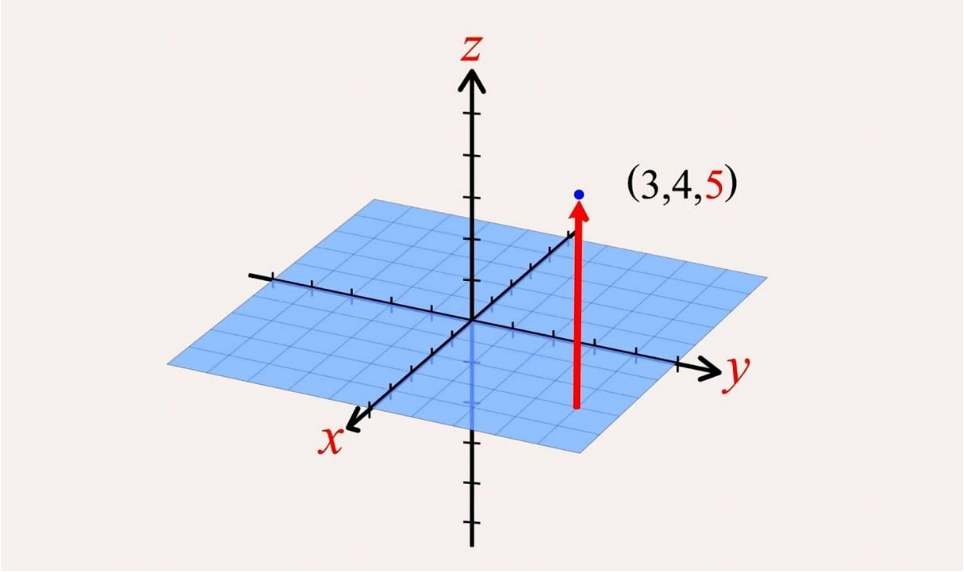
Search in parameter/solution space



Recap: Symbolic AI & Search

Search = central theme in AI

Search in parameter/solution space



In symbolic AI we focus on *discrete* solution spaces

Recap: Symbolic AI & Search

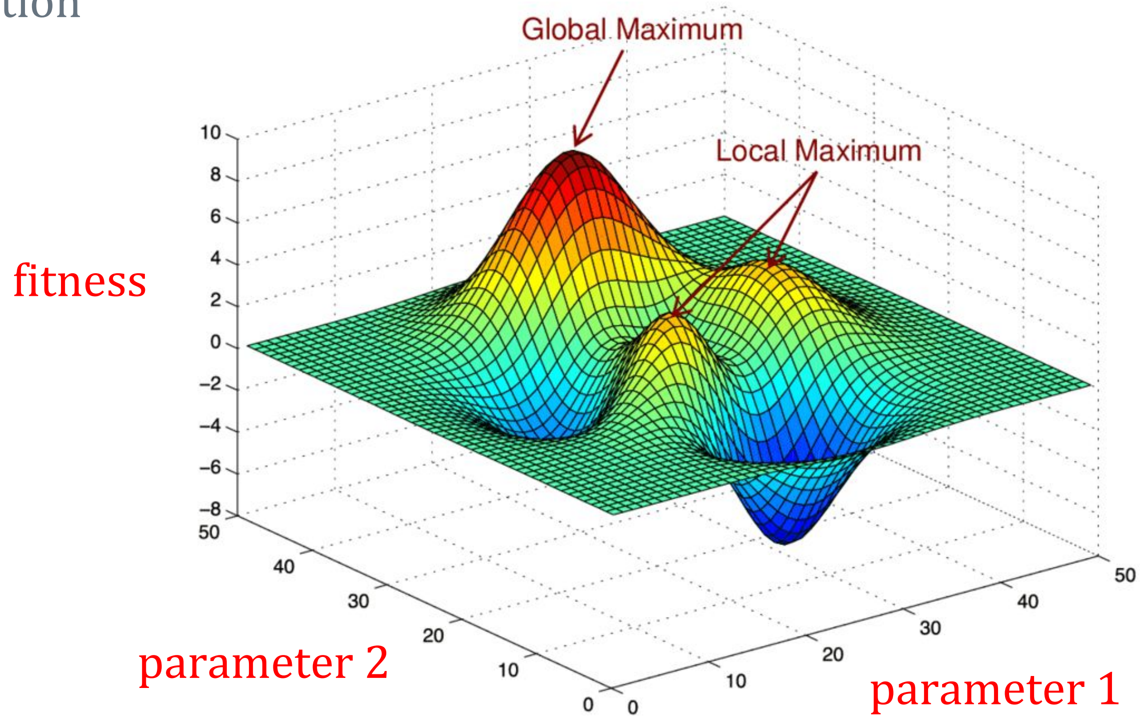
What defines whether a particular point in solution space is preferable?

1. Optimality/fitness function

Recap: Symbolic AI & Search

What defines whether a particular point in solution space is preferable?

1. Optimality/fitness function

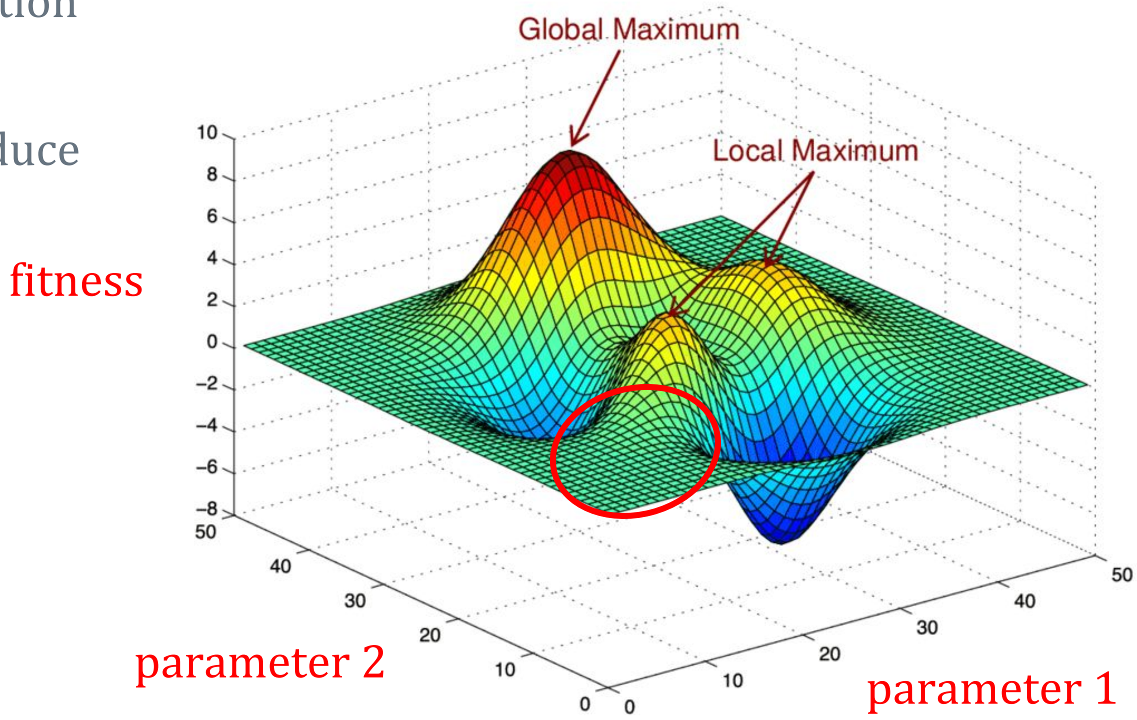


Recap: Symbolic AI & Search

What defines whether a particular point in solution space is preferable?

1. Optimality/fitness function

2. Constraints/logic to reduce feasible region



Recap: Sequential problems

Sequential problems:

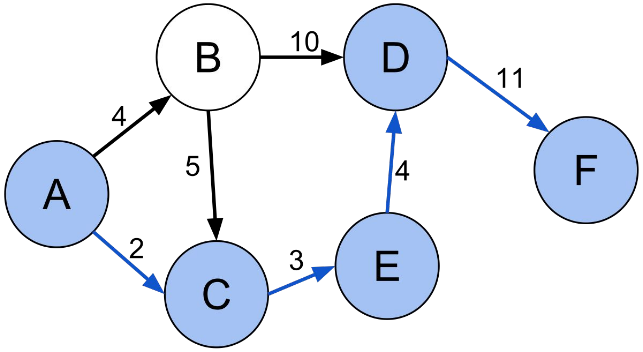
Solution space has a natural ordering

Recap: Sequential problems

Sequential problems:

Solution space has a natural ordering

- Usually sequential in *time.*
- Examples are shortest path problems.

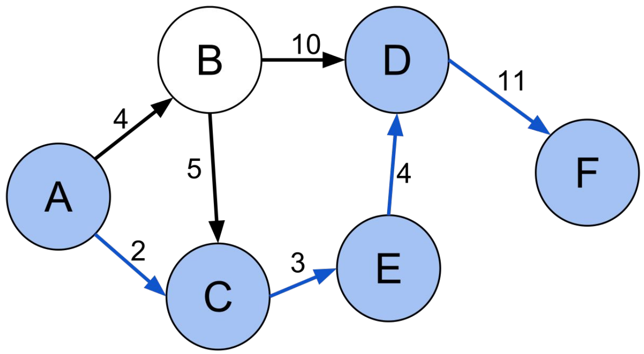


Recap: Sequential problems

Sequential problems:

Solution space has a natural ordering

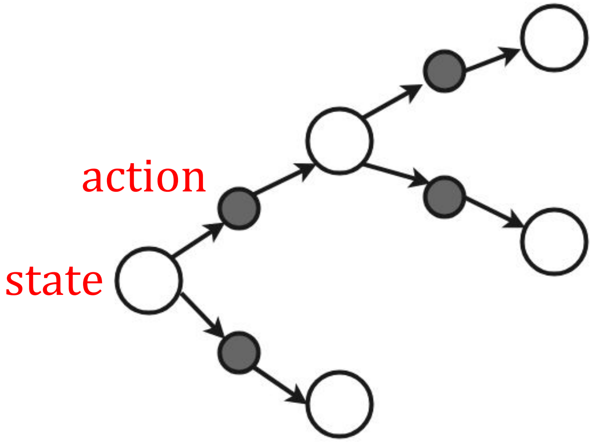
- Usually sequential in *time.*
- Examples are shortest path problems.



Solution space are the actions we take (A-C, then C-E, then E-D, etc.)

Natural sequential ordering

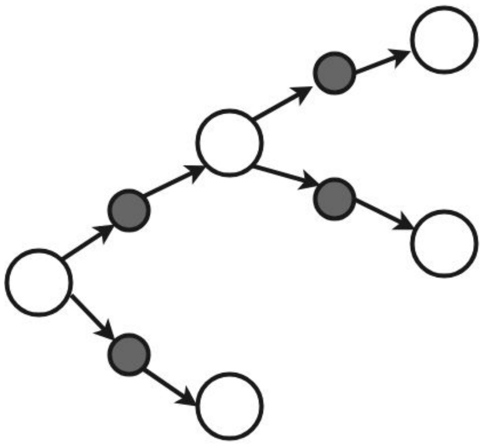
Today



Directed Graphs

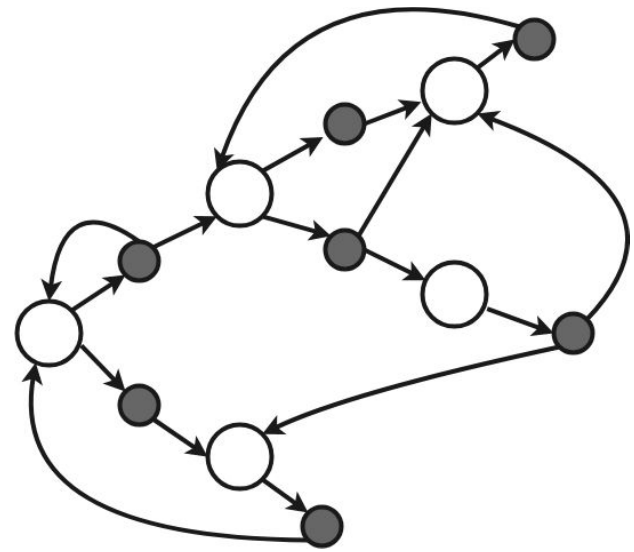
- Deterministic
- Single goal state

Today



Directed Graphs

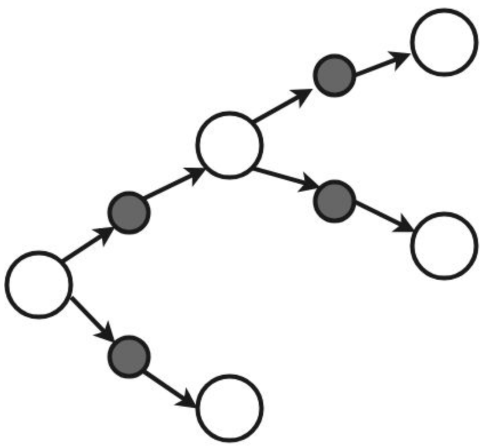
- Deterministic
- Single goal state



Markov Decision Process

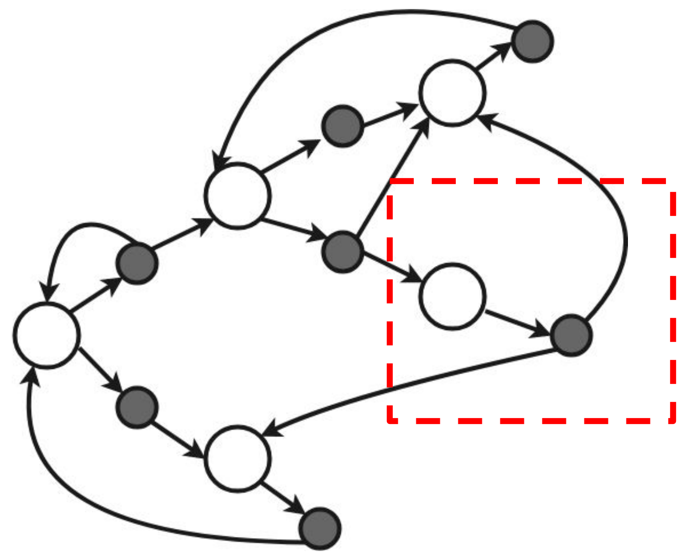
- Possibly stochastic
- Utility

Today



Directed Graphs

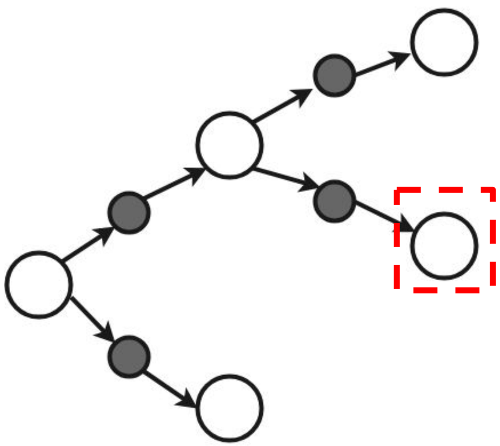
- Deterministic
- Single goal state



Markov Decision Process

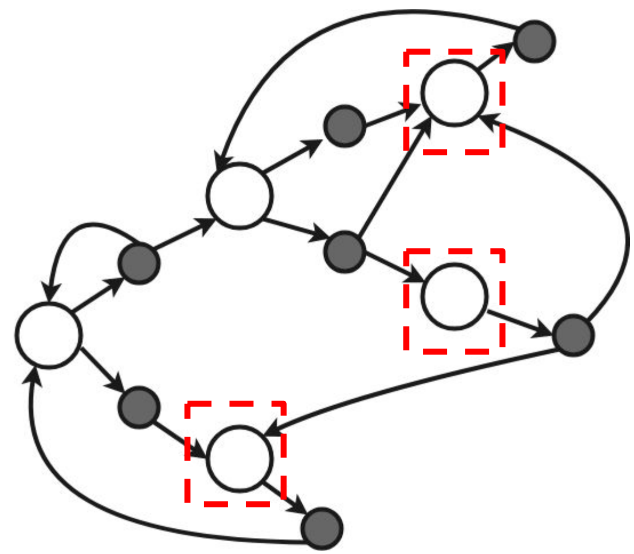
- Possibly stochastic
- Utility

Today



Directed Graphs

- Deterministic
- Single goal state



Markov Decision Process

- Possibly stochastic
- Utility (multiple goals possible)

Part 1:

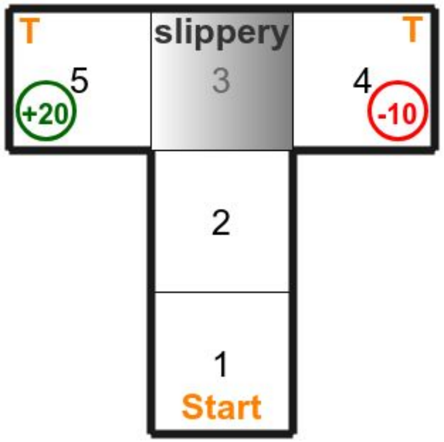
Markov Decision Process definition

MDP definition

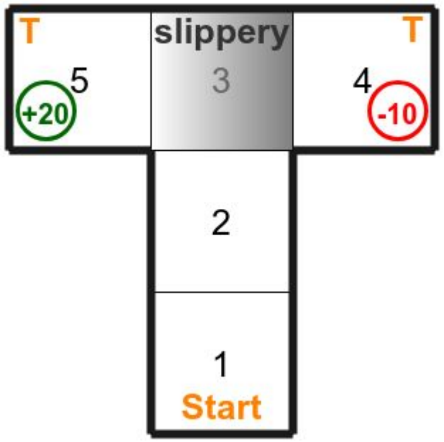
Markov Decision Process = very generic formulation

Many of the sequential problems you have seen so far
can be formulated as an MDP

Example problem



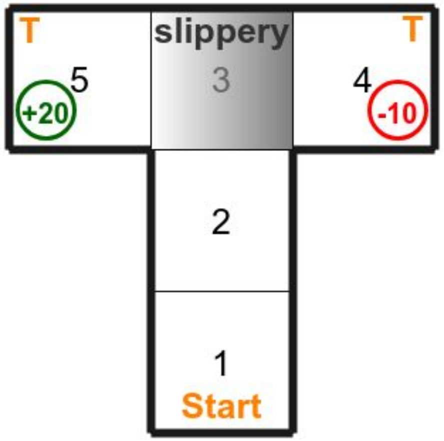
Example problem



In words:

- There are 5 states.
- We always start in state 1.
- In every state we have 4 available actions (up, down, left, right).
- Action (up, down, etc.) moves agent in that direction.

Example problem

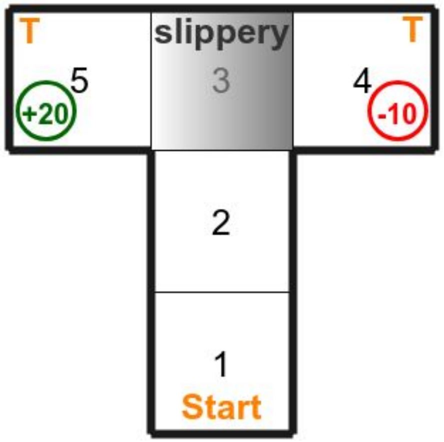


In words:

- There are 5 states.
- We always start in state 1.
- In every state we have 4 available actions (up, down, left, right).
- Action (up, down, etc.) moves agent in that direction.

- When we move into a wall, we stay at the same location.
- State 3 is 'slippery'. When we step on state 3, we have a 20% chance to slip to state 4.
- States 4 and 5 are terminal (episode ends).

Example problem



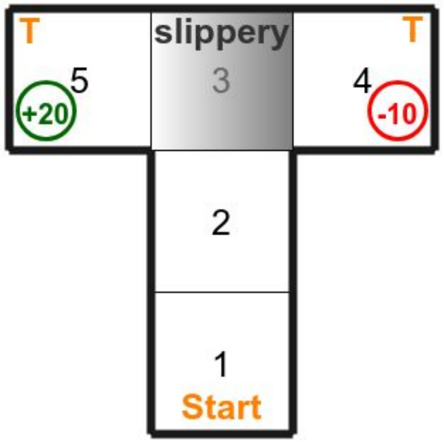
In words:

- There are 5 states.
- We always start in state 1.
- In every state we have 4 available actions (up, down, left, right).
- Action (up, down, etc.) moves agent in that direction.

- When we move into a wall, we stay at the same location.
- State 3 is 'slippery'. When we step on state 3, we have a 20% chance to slip to state 4.
- States 4 and 5 are terminal (episode ends).

- State 5 has reward of +20.
- State 4 has a penalty of -10.
- Every other action has a penalty of -1.

Example problem



In words:

- There are 5 states.
- We always start in state 1.
- In every state we have 4 available actions (up, down, left, right).
- Action (up, down, etc.) moves agent in that direction.

- When we move into a wall, we stay at the same location.
- State 3 is 'slippery'. When we step on state 3, we have a 20% chance to slip to state 4.
- States 4 and 5 are terminal (episode ends).

- State 5 has reward of +20.
- State 4 has a penalty of -10.
- Every other action has a penalty of -1.

A lot of text! And computers prefer numbers!
We want a more systematic problem definition protocol

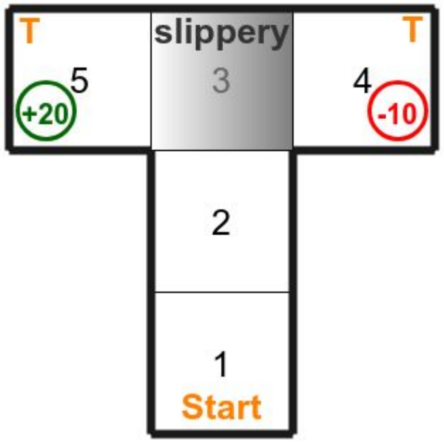
MDP definition

	Symbol	Description
1. State space	S	What are the possible observations?
2. Action space	A	What are the possible actions?
3. Transition function	T(s'|s,a)	How does the environment respond to my actions?
4. Reward function	R(s,a,s')	How good or bad is a certain transition?
5. Discount factor	γ	How much do we ignore long term benefit?
6. Initial state distribution	$p_0(s)$	Where do we start?

MDP definition

	Symbol	Description
1. State space	**S**	**What are the possible observations?**
2. Action space	**A**	**What are the possible actions?**
3. Transition function	T(s'|s,a)	How does the environment respond to my actions?
4. Reward function	R(s,a,s')	How good or bad is a certain transition?
5. Discount factor	γ	How much do we ignore long term benefit?
6. Initial state distribution	$p_0(s)$	Where do we start?

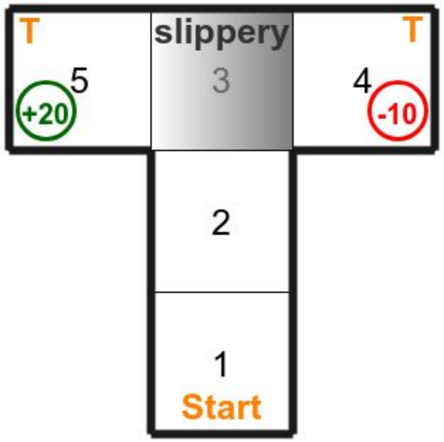
MDP definition: State + Action space



In words:

- There are 5 states.
- In every state we may move up, down, left or right.

MDP definition: State + Action space



In words:

- There are 5 states.
- In every state we may move up, down, left or right.

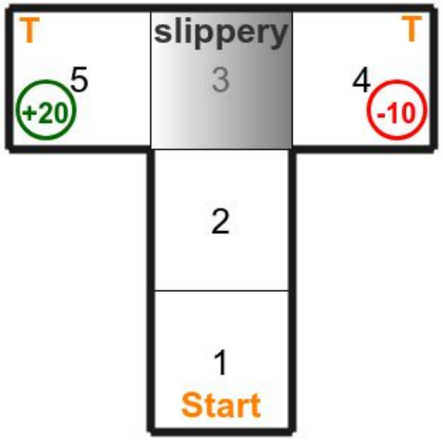
Formally:

- S = {1,2,3,4,5}
- A = {up,down,left,right} (which we assign 1,2,3,4 in a computer)
- Both are *sets*

MDP definition

	Symbol	Description
1. State space	S	What are the possible observations?
2. Action space	A	What are the possible actions?
3. Transition function	**T(s'|s,a)**	**How does the environment respond to my actions?**
4. Reward function	R(s,a,s')	How good or bad is a certain transition?
5. Discount factor	γ	How much do we ignore long term benefit?
6. Initial state distribution	$p_0(s)$	Where do we start?

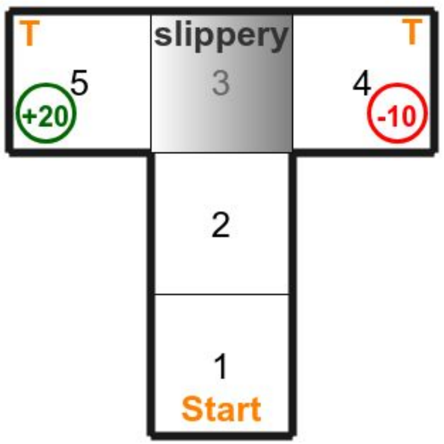
MDP definition: Transition Function



In words:

- Action (up, down, etc.) moves agent in that direction.
- When we move into a wall, we stay at the same location.
- State 3 is ‘slippery’. When we step on state 3, we have a 20% chance to slip to state 4.
- States 4 and 5 are terminal (episode ends).

MDP definition: Transition Function

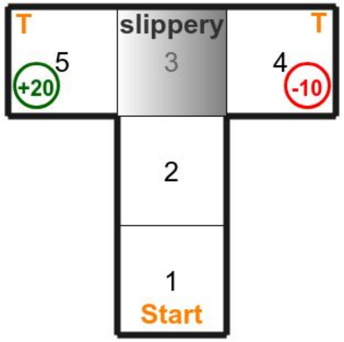


In words:

- Action (up, down, etc.) moves agent in that direction.
- When we move into a wall, we stay at the same location.
- State 3 is 'slippery'. When we step on state 3, we have a 20% chance to slip to state 4.
- States 4 and 5 are terminal (episode ends).

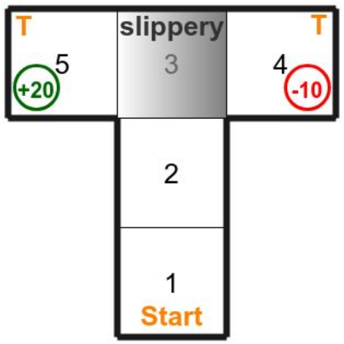
Formally:

- T(s'|s,a), a *conditional probability distribution*
- T: S x A $\rightarrow$ p(S)
- Represented as table/array of size |S|x|A|x|S| (= here: 5 x 4 x 5)



MDP definition: Transition function

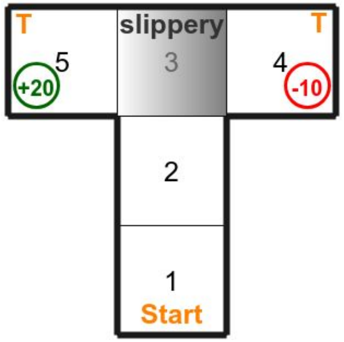
s	a	p(s'=1)	p(s'=2)	p(s'=3)	p(s'=4)	p(s'=5)
1	up	0	1	0	0	0
1	down	1	0	0	0	0
1	left	1	0	0	0	0
1	right	1	0	0	0	0
2	up	0	0	0.80	0.20	0
2	down	1	0	0	0	0
2	left	0	1	0	0	0
2	right	0	1	0	0	0
etc.						



MDP definition: Transition function

s	a	p(s'=1)	p(s'=2)	p(s'=3)	p(s'=4)	p(s'=5)
1	up	0	1	0	0	0
1	down	1	0	0	0	0
1	left	1	0	0	0	0
1	right	1	0	0	0	0
2	up	0	0	0.80	0.20	0
2	down	1	0	0	0	0
2	left	0	1	0	0	0
2	right	0	1	0	0	0
etc.						

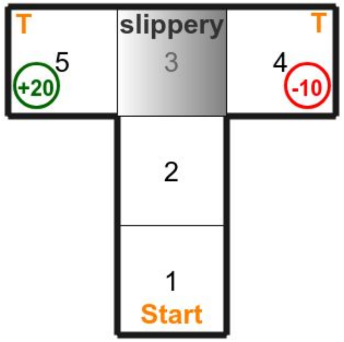
- Action (up, down, etc.) moves agent in that direction.



MDP definition: Transition function

s	a	p(s'=1)	p(s'=2)	p(s'=3)	p(s'=4)	p(s'=5)
1	up	0	1	0	0	0
1	down	1	0	0	0	0
1	left	1	0	0	0	0
1	right	1	0	0	0	0
2	up	0	0	0.80	0.20	0
2	down	1	0	0	0	0
2	left	0	1	0	0	0
2	right	0	1	0	0	0
etc.						

- When we move into a wall, we stay at the same location.



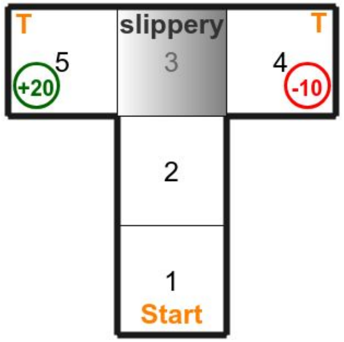
MDP definition: Transition function

s	a	p(s'=1)	p(s'=2)	p(s'=3)	p(s'=4)	p(s'=5)
1	up	0	1	0	0	0
1	down	1	0	0	0	0
1	left	1	0	0	0	0
1	right	1	0	0	0	0
2	up	0	0	0.80	0.20	0
2	down	1	0	0	0	0
2	left	0	1	0	0	0
2	right	0	1	0	0	0
etc.						

Probability distributions always sum to 1.

- State 3 is 'slippery'.

MDP definition: Transition function

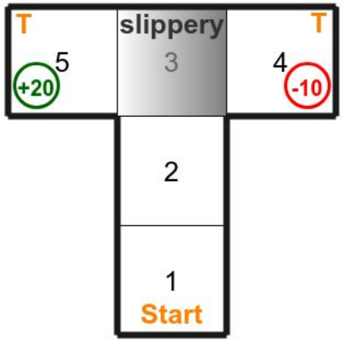


s	a	p(s'=1)	p(s'=2)	p(s'=3)	p(s'=4)	p(s'=5)
1	up	0	1	0	0	0
1	down	1	0	0	0	0
1	left	1	0	0	0	0
1	right	1	0	0	0	0
2	up	0	0	0.80	0.20	0
2	down	1	0	0	0	0
2	left	0	1	0	0	0
2	right	0	1	0	0	0
etc.						

Probability distributions always sum to 1.

If any action in the MDP can lead to multiple next states, the MDP is *stochastic*.

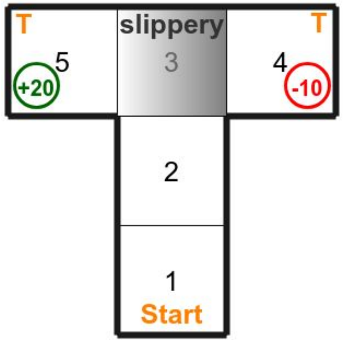
MDP definition: Transition function



Terminal states (state 4 and 5) -- episode ends

Two perspectives:

MDP definition: Transition function

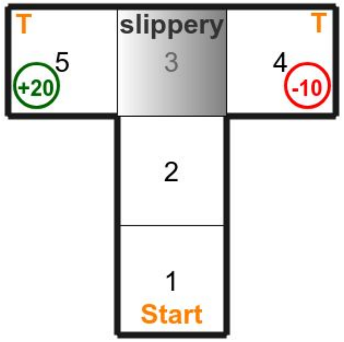


Terminal states (state 4 and 5) -- episode ends

Two perspectives:

1. No available actions.

MDP definition: Transition function



Terminal states (state 4 and 5) -- episode ends

<u>Two perspectives</u>:

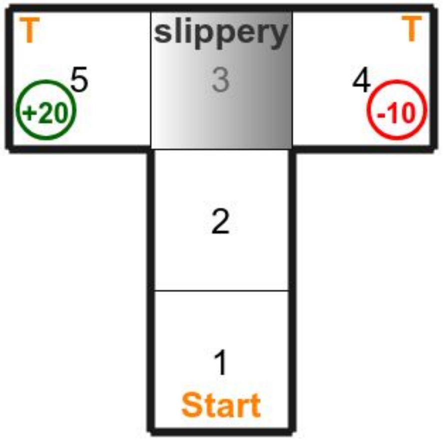
1. No available actions.
2. Absorbing state: all actions lead back to same state with reward 0.

s	a	p(s'=1)	p(s'=2)	p(s'=3)	p(s'=4)	p(s'=5)
4	up	0	0	0	1	0
4	down	0	0	0	1	0
4	left	0	0	0	1	0
4	right	0	0	0	1	0

MDP definition

	Symbol	Description
1. State space	S	What are the possible observations?
2. Action space	A	What are the possible actions?
3. Transition function	T(s'|s,a)	How does the environment respond to my actions?
4. Reward function	**R(s,a,s')**	**How good or bad is a certain transition?**
5. Discount factor	γ	How much do we ignore long term benefit?
6. Initial state distribution	$p_0(s)$	Where do we start?

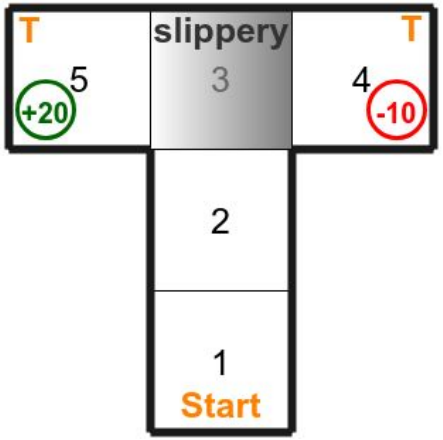
MDP definition: Reward function



In words:

- State 5 has reward of +20.
- State 4 has a penalty of -10.
- Every other action has a penalty of -1.

MDP definition: Reward function



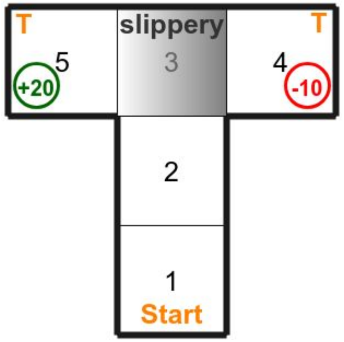
In words:

- State 5 has reward of +20.
- State 4 has a penalty of -10.
- Every other action has a penalty of -1.

Formally:

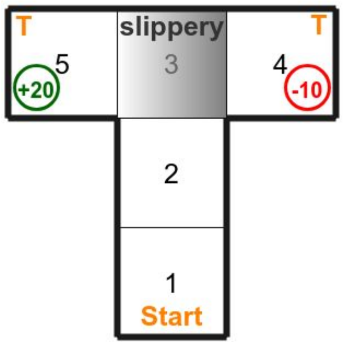
- R(s,a,s'): a *function*
- What is the reward of taking action a in state s, and reaching state s'.
- Again need table/array of size |S|x|A|x|S| (= here 5 x 4 x 5)
- Each entry a reward (real number)

MDP definition: Reward function



s	a	s'	R(s,a,s')
1	up	1	-1
1	up	2	-1
1	up	3	-1
1	up	4	-1
1	up	5	-1
..	..	..	..
..	..	..	..
3	left	5	+20

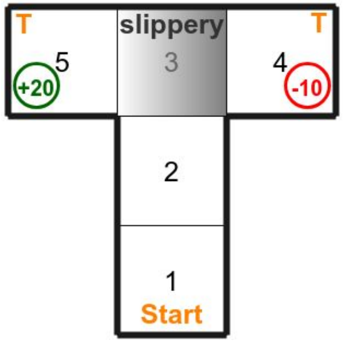
MDP definition: Reward function



s	a	s'	R(s,a,s')
1	up	1	-1
1	up	2	-1
1	up	3	-1
1	up	4	-1
1	up	5	-1
..	..	..	..
..	..	..	..
3	left	5	+20

Some transition not even possible.

MDP definition: Reward function

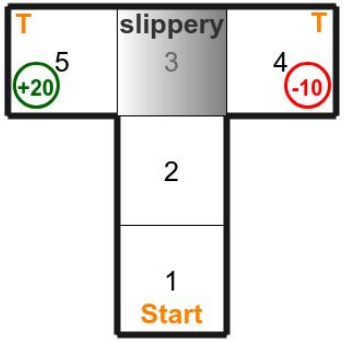


s	a	s'	R(s,a,s')
1	up	1	-1
1	up	2	-1
1	up	3	-1
1	up	4	-1
1	up	5	-1
..	..	..	..
..	..	..	..
3	left	5	+20

Often R(s,a,s') is defined as:

- R(s,a) (only current state/action)
- R(s') (only which state we reach)

MDP definition: Reward function

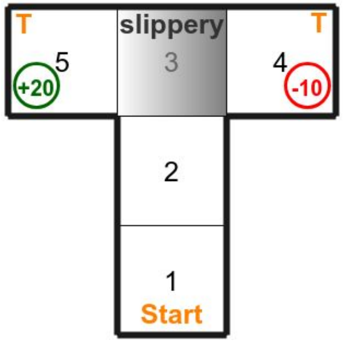


s'	R(s')
1	-1
2	-1
3	-1
4	-10
5	+20

Often R(s,a,s') is defined as:

- R(s,a) (only current state/action)
- **R(s') (only which state we reach)**

MDP definition: Reward function



s'	R(s')
1	-1
2	-1
3	-1
4	-10
5	+20

Often R(s,a,s') is defined as:

- R(s,a) (only current state/action)
- **R(s') (only which state we reach)**

<u>Cost minimization</u> (often in path planning):

- Cost = negated reward, i.e., C(s,a,s') = - R(s,a,s')
- Then: cost *minimization* equivalent to reward *maximization*.

MDP definition

	Symbol	Description
1. State space	S	What are the possible observations?
2. Action space	A	What are the possible actions?
3. Transition function	T(s'|s,a)	How does the environment respond to my actions?
4. Reward function	R(s,a,s')	How good or bad is a certain transition?
5. Discount factor	γ	**How much do we ignore long term benefit?**
6. Initial state distribution	$p_0(s)$	Where do we start?

MDP definition

	Symbol	Description
1. State space	S	What are the possible observations?
2. Action space	A	What are the possible actions?
3. Transition function	T(s'|s,a)	How does the environment respond to my actions?
4. Reward function	R(s,a,s')	How good or bad is a certain transition?
5. Discount factor	γ	**How much do we ignore long term benefit?**
6. Initial state distribution	$p_0(s)$	Where do we start?

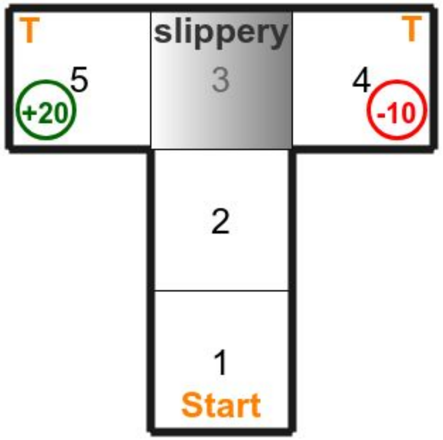
γ is a scalar in [0,1]

We discuss this later

MDP definition

	Symbol	Description
1. State space	S	What are the possible observations?
2. Action space	A	What are the possible actions?
3. Transition function	T(s'|s,a)	How does the environment respond to my actions?
4. Reward function	R(s,a,s')	How good or bad is a certain transition?
5. Discount factor	γ	How much do we ignore long term benefit?
6. Initial state distribution	$\mathbf{p_0(s)}$	**Where do we start?**

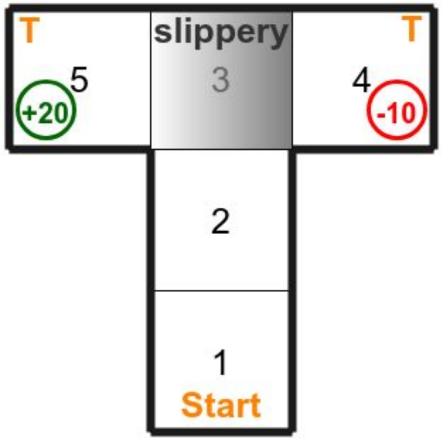
MDP definition: Initial State Distribution



In words:

- We always start in state 1.

MDP definition: Initial State Distribution



In words:

- We always start in state 1.

Formally:

- Probability distribution over S

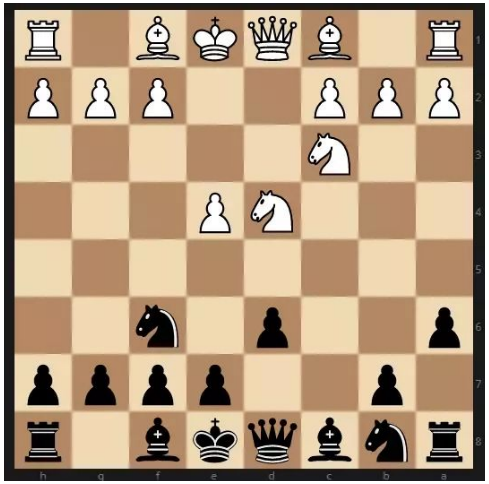
s	$p_0(s)$
1	1
2	0
3	0
4	0
5	0

Markov property

The future is independent of the past given the present

=

The present state gives all information about the system



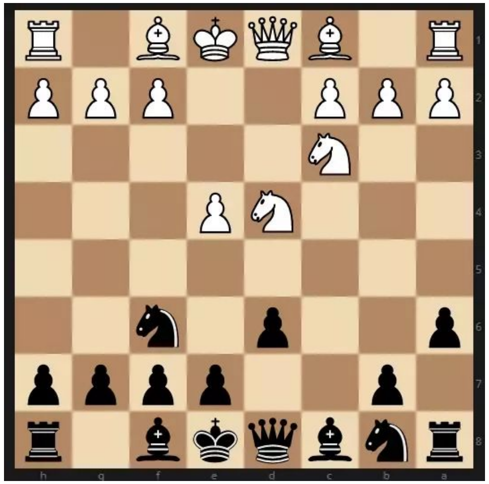
Markovian

Markov property

The future is independent of the past given the present

=

The present state gives all information about the system



Markovian

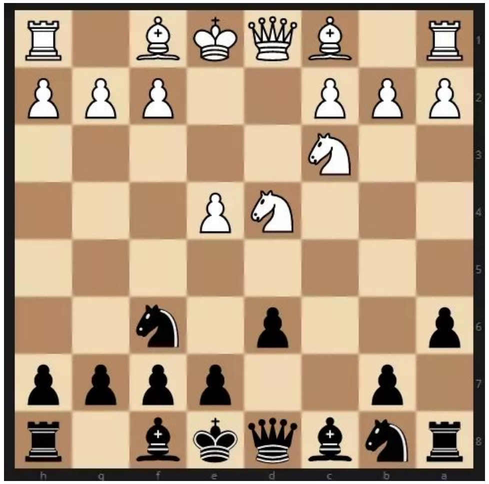
Non-Markovian/
partially observable

Markov property

The future is independent of the past given the present

=

The present state gives all information about the system



Markovian

Non-Markovian/
partially observable

Not for
today

Policy: how to act in the MDP

Policy: $\pi(a|s)$

- *A conditional probability distribution:*
- For each state s, gives a probability distribution over the actions.
- $\pi: S \rightarrow p(A)$

Policy: how to act in the MDP

Policy: $\pi(a|s)$

- *A conditional probability distribution:*
- For each state s, gives a probability distribution over the actions.
- $\pi: S \longrightarrow p(A)$

- Not part of the MDP problem definition.
- But actually our potential solution to the problem.

Policy: how to act in the MDP

Policy: $\pi(a|s)$

Example:

s	π(a=up)	π(a=down)	π(a=left)	π(s'=right)
1	0	0.5	0	0.5
2	1	0	0	0
3	0.3	0	0.3	0.4
4	-	-	-	-
5	-	-	-	-

Table of size |S|x|A|

Policy: how to act in the MDP

Policy: $\pi(a|s)$

Example:

s	π(a=up)	π(a=down)	π(a=left)	π(s'=right)
1	0	0.5	0	0.5
2	1	0	0	0
3	0.3	0	0.3	0.4
4	-	-	-	-
5	-	-	-	-

State 4 and 5 terminal, so no action possible.

Policy: how to act in the MDP

Policy: $\pi(a|s)$

Example:

s	π(a=up)	π(a=down)	π(a=left)	π(s’=right)
1	0	1	0	0
2	1	0	0	0
3	0	0	1	0
4	-	-	-	-
5	-	-	-	-

Special case:

deterministic policy

(always select one action in a state)

Policy: how to act in the MDP

Policy: $\pi(a|s)$

Example:

s	π(a=up)	π(a=down)	π(a=left)	π(s'=right)
1	0	1	0	0
2	1	0	0	0
3	0	0	1	0
4	-	-	-	-
5	-	-	-	-

Special case:

deterministic policy

(always select one action in a state)

Write $\pi(s)$, e.g.:

$\pi(s=2)$="up"

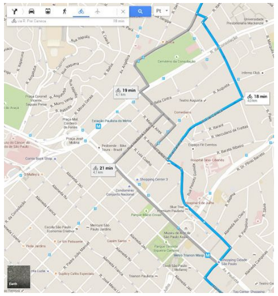
MDP overview

	Symbol	Description
1. State space	S	What are the possible observations?
2. Action space	A	What are the possible actions?
3. Transition function	T(s'|s,a)	How does the environment respond to my actions?
4. Reward function	R(s,a,s')	How good or bad is a certain transition?
5. Discount factor	γ	How much do we ignore long term benefit?
6. Initial state distribution	$p_0(s)$	Where do we start?

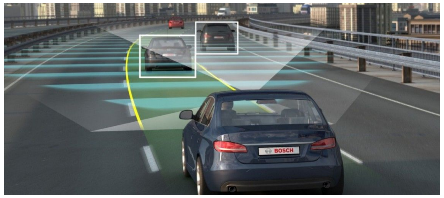
Policy	$\pi(a\|s)$	How do we act in the environment

MDP overview

Problem definition applicable to a variety of settings



Path planning with single goal

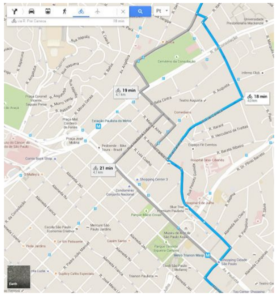


Stochasticity

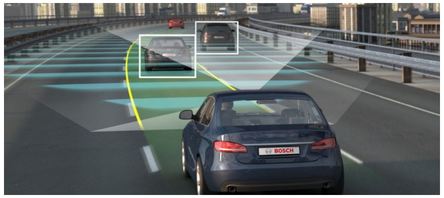
Multiple goals

MDP overview

Problem definition applicable to a variety of settings



Path planning with single goal



Stochasticity

Multiple goals

Of course our goal is to find a good policy!

But how do we define "good"?

Part 2:

Cumulative return & Value

Trace

We can act in the MDP by taking actions. This generates a *trace:*

$$h_t = \{s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, ..., a_{t+n}, r_{t+n}, s_{t+n+1}\}.$$

Here: subscripts are time index, e.g. $r_0 = R(s_0,a_0,s_1)$

Trace

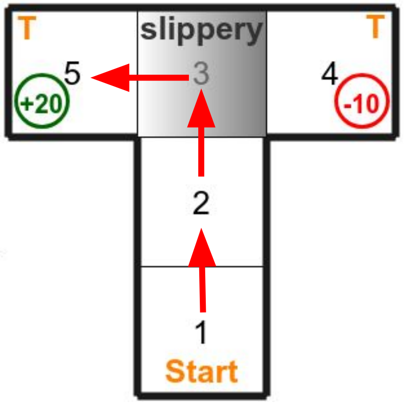
We can act in the MDP by taking actions. This generates a *trace:*

$$h_t = \{s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, \ldots, a_{t+n}, r_{t+n}, s_{t+n+1}\}.$$

Here: subscripts are time index, e.g. $r_0 = R(s_0, a_0, s_1)$

Example trace:

$\{s_0{=}1, a_0{=}\text{up}, r_0{=}\text{-}1, s_1{=}2, a_1{=}\text{up}, r_1{=}\text{-}1, s_2{=}3, a_2{=}\text{left}, r_2{=}20, s_3{=}5\}$

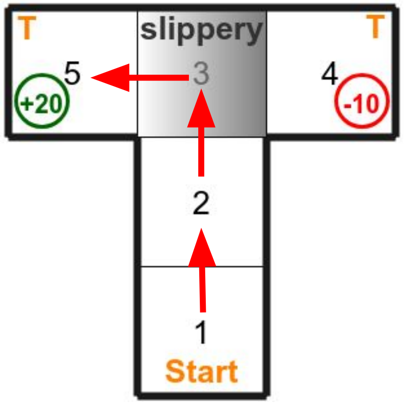


Trace

We can act in the MDP by taking actions. This generates a *trace:*

$$h_t = \{s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, \ldots, a_{t+n}, r_{t+n}, s_{t+n+1}\}.$$

Here: subscripts are time index, e.g. $r_0 = R(s_0, a_0, s_1)$



Example trace:

$\{s_0{=}1, a_0{=}\text{up}, r_0{=}\text{-}1, s_1{=}2, a_1{=}\text{up}, r_1{=}\text{-}1, s_2{=}3, a_2{=}\text{left}, r_2{=}20, s_3{=}5\}$

There are many rewards in a trace

Return (= cumulative reward)

We want to get the highest total reward!

Cumulative reward (=return)

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + ... + \gamma^n \cdot r_{t+n}$$

Return (= cumulative reward)

We want to get the highest total reward!

<u>Cumulative reward (=return)</u>

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

First reward

Second reward

Third reward

Return (= cumulative reward)

We want to get the highest total reward!

Cumulative reward (=return)

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

Future rewards discounted by γ∈ [0, 1]

We will mostly ignore discounting, and fix γ=1

Return (= cumulative reward)

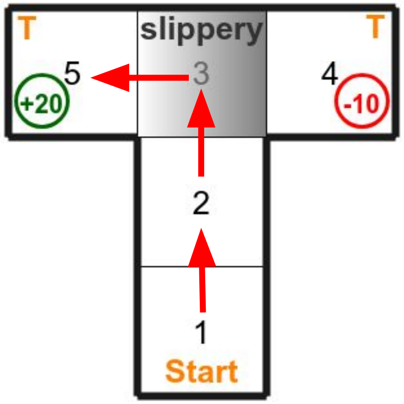
We want to get the highest total reward!

Cumulative reward (=return)

$$
\begin{aligned}
G_t &= r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n} \\
&= \sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i}
\end{aligned}
$$

= summation notation for cumulative reward

Return (= cumulative reward)



<u>Example trace:</u>

$\{s_0{=}1, a_0{=}\text{up}, r_0{=}\text{-}1, s_1{=}2, a_1{=}\text{up}, r_1{=}\text{-}1, s_2{=}3, a_2{=}\text{left}, r_2{=}20, s_3{=}5\}$

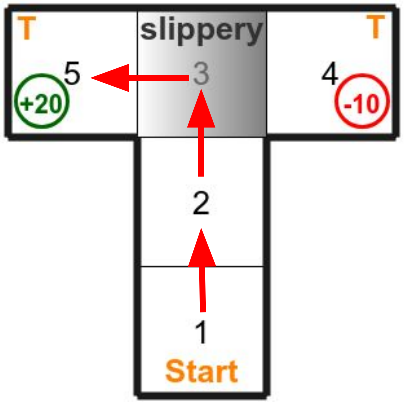
Q: What is the cumulative reward of this trace (assume $\gamma = 1.0$)?

A: G_0 =

Hint:

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

Return (= cumulative reward)



Example trace:

$$\{s_0{=}1, a_0{=}\text{up}, r_0{=}\text{-}1, s_1{=}2, a_1{=}\text{up}, r_1{=}\text{-}1, s_2{=}3, a_2{=}\text{left}, r_2{=}20, s_3{=}5\}$$

Q: What is the cumulative reward of this trace (assume $\gamma = 1.0$)?

A: $G_0 = -1 + 1.0 \cdot -1 + 1.0^2 \cdot 20$

$= (-1) + (-1) + 20$

$= 18$

Hint:

$$G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

Value/Utility

We will not always observe the same trace from a state:

(environment and policy can be stochastic)

Value

(= utility)

the *average cumulative reward* we expected to get,

when starting in state s, and following policy π

Value (= expected cumulative reward)

$$V^{\pi}(s) = \mathbb{E}_{\pi,T}\Big[\sum_{i=0}^{\infty} \gamma^{i} \cdot r_{t+i} | s_t = s\Big]$$

Value (= expected cumulative reward)

$$V^{\pi}(s) = \mathbb{E}_{\pi,T}\Big[\underbrace{\sum_{i=0}^{\infty} \gamma^{i} \cdot r_{t+i}}_{\text{Cumulative reward}} \Big| s_t = s\Big]$$

Value of state s given policy π

Cumulative reward

Value (= expected cumulative reward)

$$V^{\pi}(s) = \mathbb{E}_{\pi,T}\big[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} | s_t = s\big]$$

Value of state s given policy π

Cumulative reward

Expectation over all traces induced by policy π and environment T

Recap: Expectation

Definition

- Discrete variable X with distribution p(X)
- Expectation of function f(·) of X:

$$\mathbb{E}_{X\sim p(X)}[f(X)] = \sum_{x \in X}[f(x) \cdot p(x)]$$

" the average of a random variable"

Example

Recap: Expectation

Definition

- Discrete variable X with distribution p(X)
- Expectation of function f(·) of X:

$$\mathbb{E}_{X \sim p(X)}[f(X)] = \sum_{x \in X} [f(x) \cdot p(x)]$$

Simply weight every outcome by its probability of occuring

Example

Recap: Expectation

Definition

- Discrete variable X with distribution p(X)
- Expectation of function f(·) of X:

$$\mathbb{E}_{X\sim p(X)}[f(X)] = \sum_{x\in X} \underline{[f(x)\cdot p(x)]}$$

Simply weight every outcome by its probability of occuring

Example

x	p(x)	f(x)
1	0.2	22.0
2	0.3	13.0
3	0.5	7.4

Q: *Compute E[f(X)]*

A:

Recap: Expectation

Definition

- Discrete variable X with distribution p(X)
- Expectation of function f(·) of X:

$$\mathbb{E}_{X\sim p(X)}[f(X)] = \sum_{x\in X}\underline{[f(x)\cdot p(x)]}$$

Simply weight every outcome
by its probability of occuring

Example

x	p(x)	f(x)
1	0.2	22.0
2	0.3	13.0
3	0.5	7.4

Q: *Compute E[f(X)]*

A:

$$\begin{aligned}\mathbb{E}_x[f(x)] &= 0.2\cdot 22.0 + 0.3\cdot 13.0 + 0.5\cdot 7.4\\ &= 12.0\end{aligned}$$

Value (= expected cumulative reward)

$$V^\pi(s) = \mathbb{E}_{\pi,T}\Big[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} | s_t = s\Big]$$

Sum over all traces, multiplied by their probability of occuring

Value (= expected cumulative reward)

$$V^{\pi}(s) = \mathbb{E}_{\pi,T}\Big[\sum_{i=0}^{\infty}\gamma^{i}\cdot r_{t+i}\,\Big|\,s_t = s\Big]$$

Sum over all traces, multiplied by their probability of occuring

Q: Imagine, *given a certain policy* π, *we can get two traces from state s.*

- *The first trace has return G=6 and occurs 60% of times.*
- *The second trace has return G = 9 and occurs 40% of times.*

Compute V(s)

Value (= expected cumulative reward)

$$V^{\pi}(s) = \mathbb{E}_{\pi,T}\big[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} | s_t = s\big]$$

Sum over all traces, multiplied by their probability of occuring

Q: Imagine, *given a certain policy π, we can get two traces from state s.*

- *The first trace has return G=6 and occurs 60% of times.*
- *The second trace has return G = 9 and occurs 40% of times.*

Compute V(s)

A: 0.6 · 6 + 0.4 · 9 = 3.6 + 3.6 = 7.2

Value = function

- Value is a function!
    - For every possible state s, there is one $V^{\pi}(s)$, e.g.:
    - Can be represented as a table.

s	$V^{\pi}(s)$
1	4.5
2	7.3
3	2.3
4	0
5	0

Value = function

- Value is a function!
  - For every possible state s, there is one $V^\pi(s)$, e.g.:
  - Can be represented as a table.

s	$V^\pi(s)$
1	4.5
2	7.3
3	2.3
4	0
5	0

- Every policy has its own associated value function: $V^\pi(s)$
  - We sometimes omit π for simplicity, and write $V(s)$

Value = function

- Value is a function!
    - For every possible state s, there is one $V^\pi(s)$, e.g.:
    - Can be represented as a table.

s	$V^\pi(s)$
1	4.5
2	7.3
3	2.3
4	0
5	0

- Every policy has its own associated value function: $V^\pi(s)$
    - We sometimes omit π for simplicity, and write $V(s)$

- The value of a terminal state is by definition 0!

State-action value Q(s,a)

State value

$$V^{\pi}(s) = \mathbb{E}_{\pi,T}\left[\sum_{i=0}^{\infty} \gamma^{i} \cdot r_{t+i} | s_t = s\right]$$

State-action value Q(s,a)

State value

$$V^{\pi}(s) = \mathbb{E}_{\pi,T}\left[\sum_{i=0}^{\infty} \gamma^{i} \cdot r_{t+i} | s_t = s\right]$$

State-action value

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi,T}\left[\sum_{i=0}^{\infty} \gamma^{i} \cdot r_{t+i} | s_t = s, a_t = a\right]$$

State-action value Q(s,a)

State value

$$V^\pi(s) = \mathbb{E}_{\pi,T}\big[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} | s_t = s\big]$$

State-action value

We also condition on the first action!

$$Q^\pi(s,a) = \mathbb{E}_{\pi,T}\big[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} | s_t = s, a_t = a\big]$$

Representation of V(s) and Q(s,a) in memory

<u>State values</u>

V(s)

s=1	V=3
s=2	-4
s=3	9
s=4	...
s=5	

Vector of size |S|

Representation of V(s) and Q(s,a) in memory

State values

V(s)

s=1	V=3
s=2	-4
s=3	9
s=4	...
s=5	

Vector of size |S|

State-action values

Q(s,a)

	a=up	**a=down**	**a=left**	**a=right**
s=1	Q=5	3	...	
s=2	9	4	...	
s=3	4	2	...	
s=4	...	...		
s=5				

Matrix of size |S| x |A|

Example computation of Q(s,a)

Q: Imagine, we are in state 3, take action “up”, and afterwards follow policy π.

- 20% of times we then observe a return of 10.
- 40% of times we then observe a return of -4.
- 40% of times we then observe a return of 2.

Compute Q^π(s=3,a=“up”)

A:

Hint

$$Q^\pi(s,a) = \mathbb{E}_{\pi,T}\left[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} | s_t = s, a_t = a\right]$$

Example computation of Q(s,a)

Q: Imagine, we are in state 3, take action "up", and afterwards follow policy π.

- 20% of times we then observe a return of 10.
- 40% of times we then observe a return of -4.
- 40% of times we then observe a return of 2.

Compute $Q^\pi(s=3,a=\text{"up"})$

A: $Q^\pi(s=3,a=\text{"up"})$ = $0.2 \cdot 10 + 0.4 \cdot (-4) + 0.4 \cdot 2$

= $2 - 1.6 + 0.8$

= 1.2

Hint

$$Q^\pi(s,a) = \mathbb{E}_{\pi,T}\left[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i} | s_t = s, a_t = a\right]$$

Optimal value function

Main idea of MDPs: we want to find a policy with the highest possible value

Optimal value function

Main idea of MDPs: we want to find a policy with the highest possible value

- Each possible policy has an associated value function.

- Theorem:
  One of these possible value functions is better than all others, i.e., it is the **optimal value function V*(s)**

$$V^\star(s) \geq V^\pi(s), \quad \text{for all } \pi \in \Pi, s \in \mathcal{S}$$

Optimal value function and optimal policy

Main idea of MDPs: we want to find a policy with the highest possible value

- Each possible policy has an associated value function.

- Theorem:
  One of these possible value functions is better than all others, i.e., it is the **optimal value function V*(s)**

$$V^\star(s) \geq V^\pi(s), \quad \text{for all } \pi \in \Pi, s \in \mathcal{S}$$

- A policy that achieves this optimal value function is **an optimal policy π*(a|s)**

$$\pi^\star(s) = \arg\max_\pi V^\pi(s)$$

Optimal value function and optimal policy

Main idea of MDPs: we want to find a policy with the highest possible value

- Each possible policy has an associated value function.

- Theorem:
  One of these possible value functions is better than all others, i.e., it is the **optimal value function V*(s)**

$$V^\star(s) \geq V^\pi(s), \quad \text{for all } \pi \in \Pi, s \in \mathcal{S}$$

- A policy that achieves this optimal value function is **an optimal policy π*(a|s)**

$$\pi^\star(s) = \arg\max_\pi V^\pi(s)$$

We want to find π*(a|s)

(or V*(s) / Q*(s,a))

Summary

<u>Problem definition</u>: Markov Decision Process (MDP)

<u>Solution space</u>: Policy $\pi(a|s)$

<u>Solution criterion</u>: Value/utility (= expected cumulative reward)

<u>Optimality</u>: One optimal value function $V^*(s)$ / $Q^*(s,a)$, achieved by the optimal policy $\pi^*(a|s)$

Summary

Problem definition: Markov Decision Process (MDP)

Solution space: Policy $\pi(a|s)$

Solution criterion: Value/utility (= expected cumulative reward)

Optimality: One optimal value function $V^*(s)$ / $Q^*(s,a)$, achieved by the optimal policy $\pi^*(a|s)$

After the break we will try to find $\pi^*(a|s)$ / $V^*(s)$ / $Q^*(s,a)$

Break

Part 3:

Bellman Equation

Bellman Equation

Value function can be written as a *recursive* formula

$$V(s) = \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$

Bellman Equation

Value function can be written as a *recursive* formula

$$
\begin{aligned}
V(s) &= \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right] \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) \Big[\sum_{s' \in \mathcal{S}} T(s'|s,a) \left[r + \gamma \cdot V(s') \right] \Big]
\end{aligned}
$$

Bellman Equation

Value function can be written as a *recursive* formula

$$V(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\mathbb{E}_{s'\sim T(\cdot|a,s)}\left[r + \gamma \cdot V(s')\right]$$
$$= \sum_{a\in\mathcal{A}} \pi(a|s)\Big[\sum_{s'\in\mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]\Big]$$

Value at state s is a function of values at next states s'!

Bellman Equation

Value function can be written as a *recursive* formula

$$\begin{aligned} V(s) &= \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \Big[\sum_{s' \in \mathcal{S}} T(s'|s,a) \left[r + \gamma \cdot V(s') \right] \Big] \end{aligned}$$

Bellman equation is a *functional equation*

(the unknown quantity is a function)

Bellman Equation

Bellman equation

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$

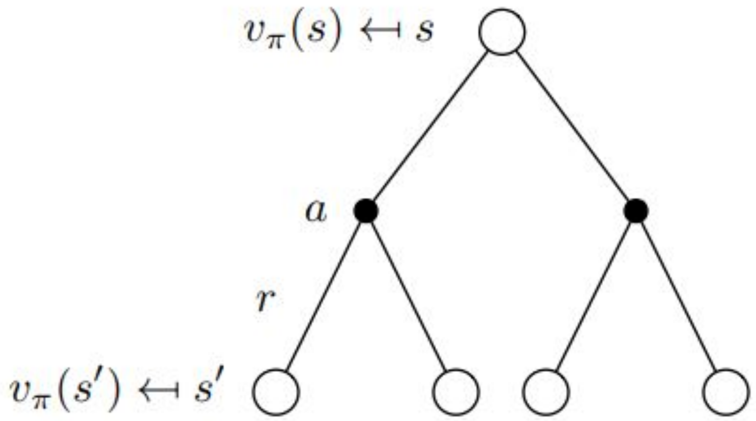
Back-up diagram

Bellman Equation

Bellman equation

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$

Back-up diagram

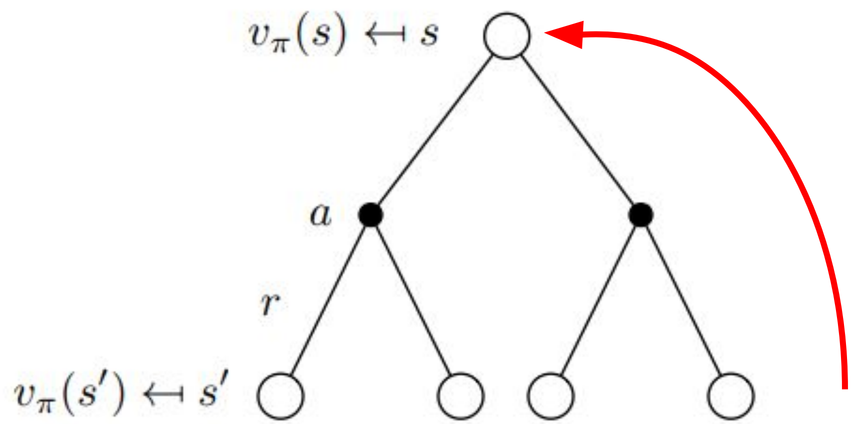


Bellman Equation

Bellman equation

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$

Back-up diagram



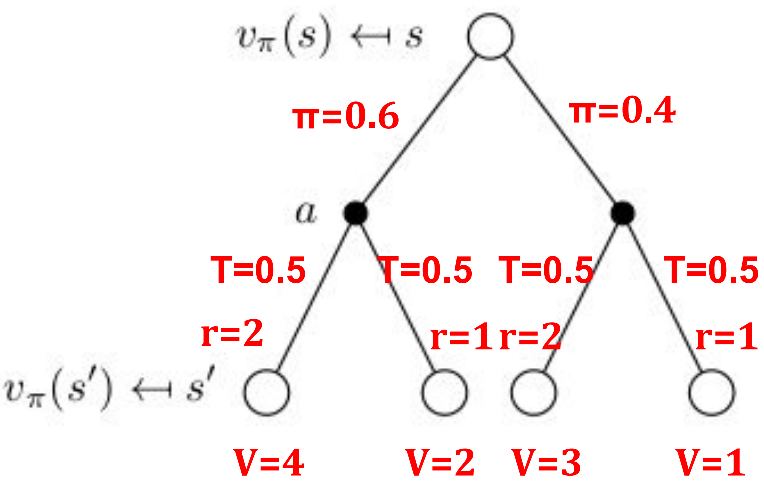
If we knew V(s’),

we can compute V(s)

=

“back-up”

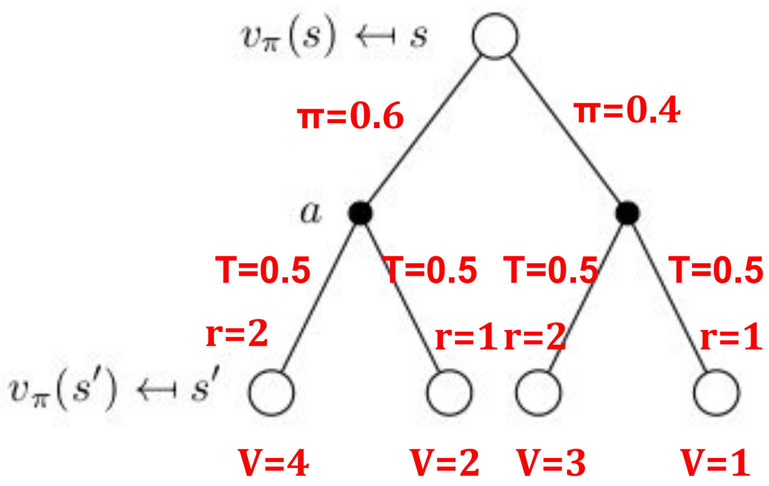
Bellman Equation



Example

Policy, transition function, rewards and next state value estimates: see picture

Bellman Equation



Example

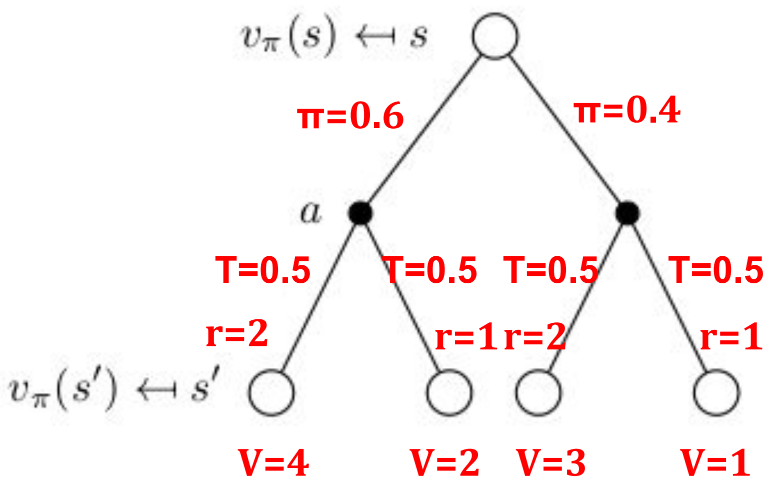
Policy, transition function, rewards and next state value estimates: see picture

Bellman equation:

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$

Q: Compute V(s) (assume γ=1.0)

Bellman Equation



Example

Policy, transition function, rewards and next state value estimates: see picture

Bellman equation:

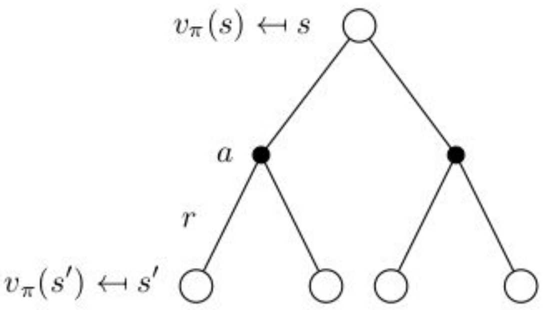
$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$

Q: Compute V(s) (assume γ=1.0)

A: V(s) = 0.6*(0.5*(2 + 1.0*4) + 0.5*(1 + 1.0*2)) +
0.4*(0.5*(2 + 1.0*3) + 0.5*(1 + 1.0*1))
= 0.6* 4.5 + 0.4*3.5 = 4.1

Bellman Equation for state-action values

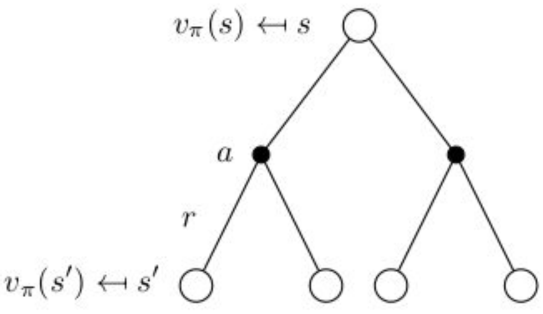
$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$



Back-up diagram for V

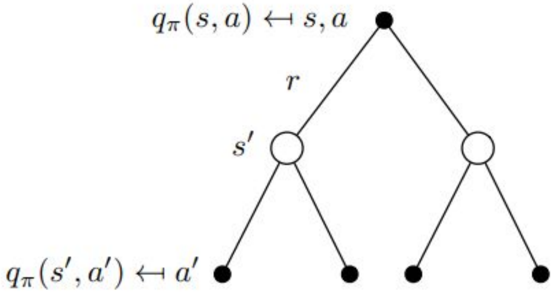
Bellman Equation for state-action values

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$



Back-up diagram for V

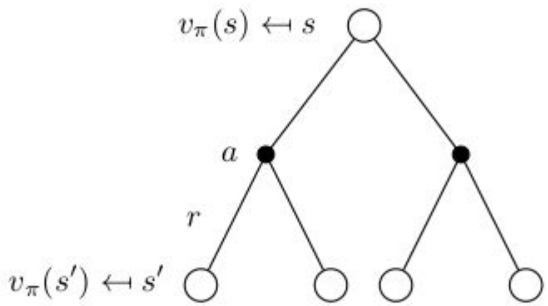
$$\begin{aligned} Q(s,a) &= \mathbb{E}_{s' \sim T}\left[r + \gamma \cdot \mathbb{E}'_{a \sim \pi}[Q(s',a')]\right] \\ &= \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot \sum_{a' \in \mathcal{A}} [\pi(a|s) \cdot Q(s',a')]\right] \end{aligned}$$



Back-up diagram for Q

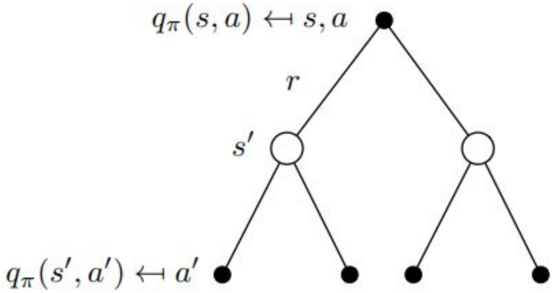
Bellman Equation for state-action values

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$



Back-up diagram for V

$$\begin{aligned} Q(s,a) &= \mathbb{E}_{s' \sim T}\big[r + \gamma \cdot \mathbb{E}_{'a \sim \pi}[Q(s',a')]\big] \\ &= \sum_{s' \in \mathcal{S}} T(s'|s,a)\big[r + \gamma \cdot \sum_{a' \in \mathcal{A}} [\pi(a|s) \cdot Q(s',a')]\big] \end{aligned}$$



Back-up diagram for Q

Essentially the same equation, only:

- Represent the function at different points.
- Therefore the summation order in the Bellman equation switches.

Relation between V(s) and Q(s,a)

- Are two ways of storing the same value function for a given π
- Can therefore be rewritten into eachother

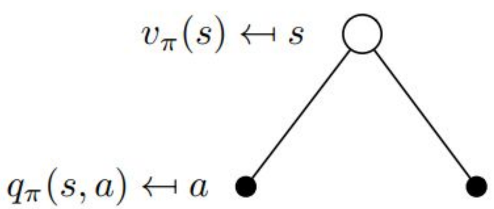
Relation between V(s) and Q(s,a)

- Are two ways of storing the same value function for a given π
- Can therefore be rewritten into eachother

	$\mathbf{Q} \rightarrow \mathbf{V}$	$\mathbf{V} \rightarrow \mathbf{Q}$
Equation		
Back-up diagram		

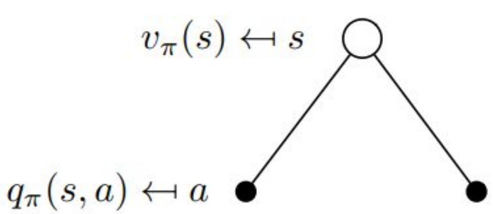
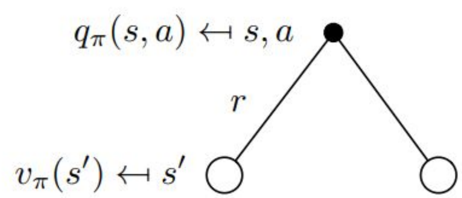
Relation between V(s) and Q(s,a)

- Are two ways of storing the same value function for a given π
- Can therefore be rewritten into eachother

	Q → V	**V → Q**
Equation	$V(s) = \mathbb{E}_{a\sim\pi}[Q(s,a)] = \sum_{a\in\mathcal{A}}[\pi(a\|s)\cdot Q(s,a)]$	
Back-up diagram		

Relation between V(s) and Q(s,a)

- Are two ways of storing the same value function for a given π
- Can therefore be rewritten into eachother

	Q → V	**V → Q**
Equation	$$V(s) = \mathbb{E}_{a\sim\pi}[Q(s,a)] = \sum_{a\in\mathcal{A}} [\pi(a\|s)\cdot Q(s,a)]$$	$$Q(s,a) = \mathbb{E}_{s'\sim T}[r+\gamma V(s')] = \sum_{s'\in\mathcal{S}} T(s'\|s,a)[r+\gamma V(s')]$$
Back-up diagram		

Relation between V(s) and Q(s,a)

Example:

- Two actions in s
- Random policy
- Q(a=1|s) = 10
- Q(a=2|s) = 20

Question: Compute V(s)

Relation between V(s) and Q(s,a)

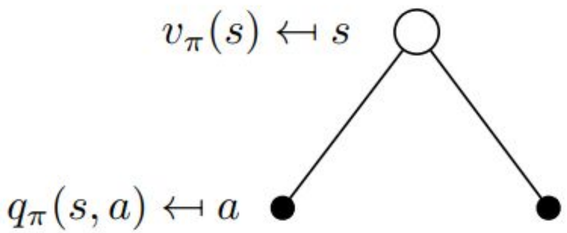
Example:

- Two actions in s
- Random policy
- Q(a=1|s) = 10
- Q(a=2|s) = 20

Question: Compute V(s)

Q to V equation

$$V(s) = \mathbb{E}_{a \sim \pi}[Q(s,a)]$$
$$= \sum_{a \in \mathcal{A}} [\pi(a|s) \cdot Q(s,a)]$$



Q to V back-up diagram

Relation between V(s) and Q(s,a)

Example:

- Two actions in s
- Random policy
- Q(a=1|s) = 10
- Q(a=2|s) = 20

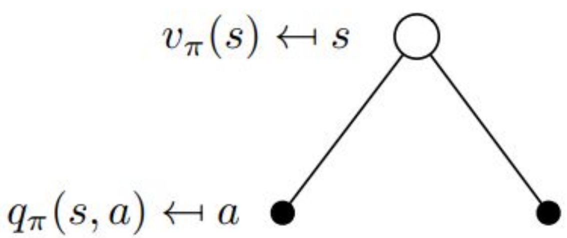
Question: Compute V(s)

Answer:

V(s) = 0.5 * 10 + 0.5*20 = 15

Q to V equation

$$V(s) = \mathbb{E}_{a \sim \pi}[Q(s,a)] \\ = \sum_{a \in \mathcal{A}} [\pi(a|s) \cdot Q(s,a)]$$



Q to V back-up diagram

Part 4

Dynamic Programming

Dynamic Programming

General concept (not only applicable to MDPs)

Key idea:

Dynamic Programming

General concept (not only applicable to MDPs)

Key idea:

- Break a large problem into smaller subproblems.

- Efficiently store and reuse intermediate results.

- Repeatedly solving the small subproblem solves the overall problem.

Dynamic Programming

General concept (not only applicable to MDPs)

Key idea:

- Break a large problem into smaller subproblems.
- Efficiently store and reuse intermediate results.
- Repeatedly solving the small subproblem solves the overall problem.

In context of MDP: a central algorithm to solve for the optimal policy

Dynamic Programming

<u>Iterate two procedure:</u>

1) Given a policy, how do we find the associated value function?

= **policy evaluation**: from π to $V^{\pi}(s)$

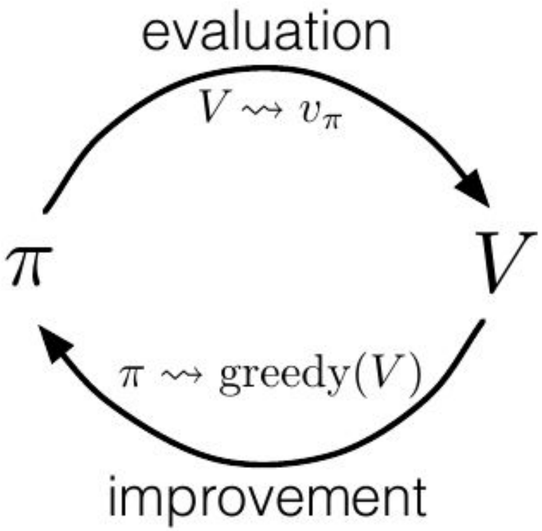
Dynamic Programming

Iterate two procedure:

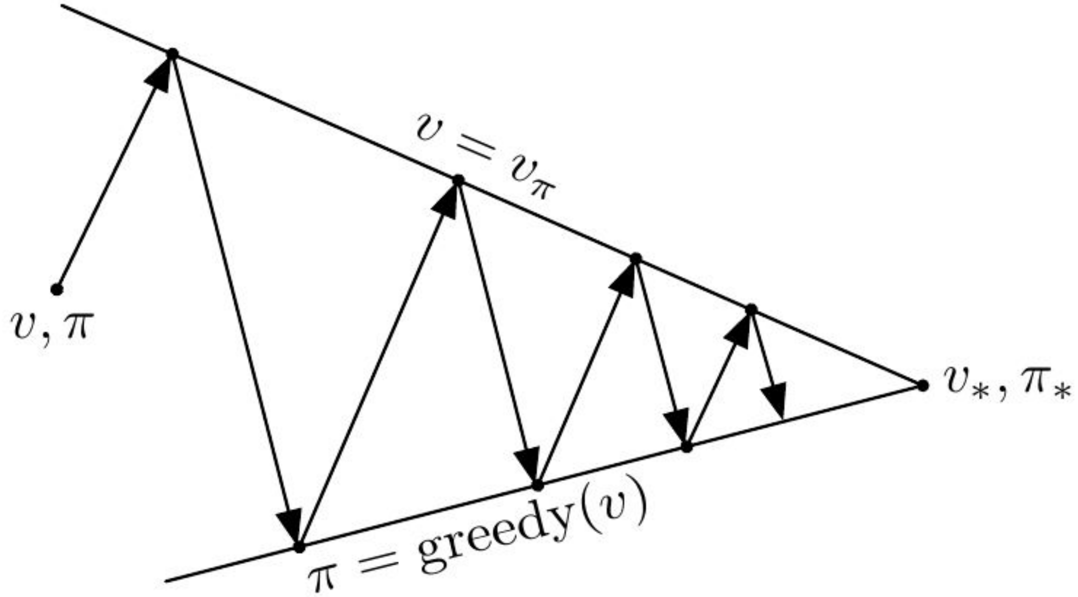
1) Given a policy, how do we find the associated value function?
= **policy evaluation**: from π to $V^{\pi}(s)$

2) Given the value function, how do we find an improved policy?
= **policy improvement**: from $V^{\pi}(s)$ to improved π

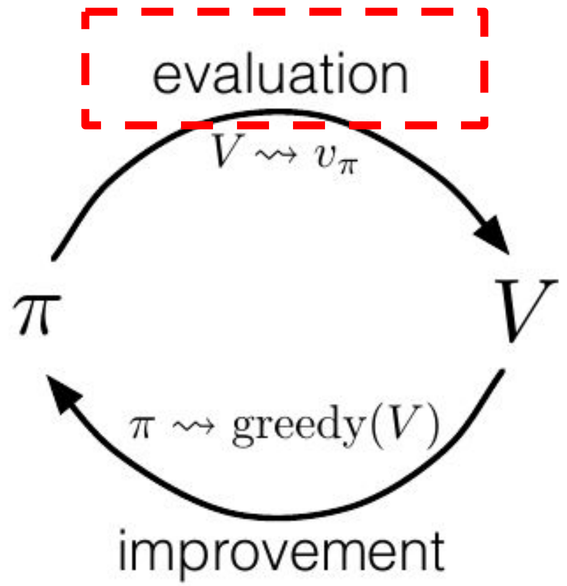
Dynamic Programming



Dynamic Programming



Dynamic Programming



Policy Evaluation

Given a policy π, find associated value function $V^{\pi}(s)$

Policy Evaluation

Given a policy π, find associated value function $V^{\pi}(s)$

- Bellman equation:

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$

  - For every s we have one such equation.
  - Therefore: *system of |S| linear equations*.

Policy Evaluation

Given a policy π, find associated value function $V^{\pi}(s)$

- Bellman equation:

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$

    - For every s we have one such equation.
    - Therefore: *system of |S| linear equations*.

- <u>Can we analytically solve for V(s)?</u>
    - Can be done, but only for small problems
    - (needs $O(|S|^3)$) matrix inverse)

Policy Evaluation

Given a policy π, find associated value function $V^{\pi}(s)$

- Bellman equation:

$$V(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} T(s'|s,a)\Big(r_t + \gamma \cdot V(s')\Big)$$

  - For every s we have one such equation.
  - Therefore: *system of |S| linear equations*.

- Can we analytically solve for V(s)? No
- <u>Therefore: iterative solution</u>

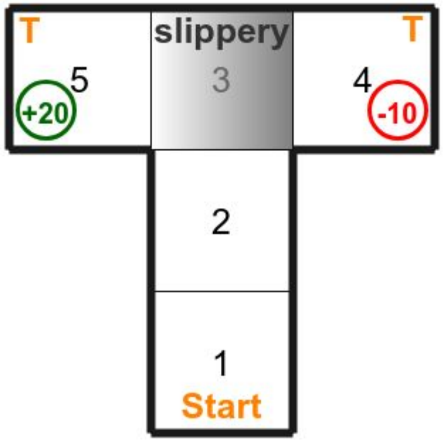
Policy Evaluation

Algorithm 1: Policy evaluation

Input: Policy $\pi(a|s)$, small threshold θ
Result: Value function $V^\pi(s)$
Initialization: A value table $V(s)$ with arbitrary entries, except $V(terminal) = 0$;
repeat
  $\Delta \leftarrow 0$
  for *each* $s \in \mathcal{S}$ **do**
    $x \leftarrow V(s)$ `/* Old value for s */`
    $V(s) \leftarrow \mathbb{E}_{a\sim\pi(\cdot|s)}\mathbb{E}_{s'\sim T(\cdot|a,s)}\left[r_t + \gamma \cdot V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |x - V(s)|)$ `/* Update the max error */`
  end
until $\Delta < \theta$;
Return $V(s)$

Full algorithm in lecture notes
We walk through an example step-by-step

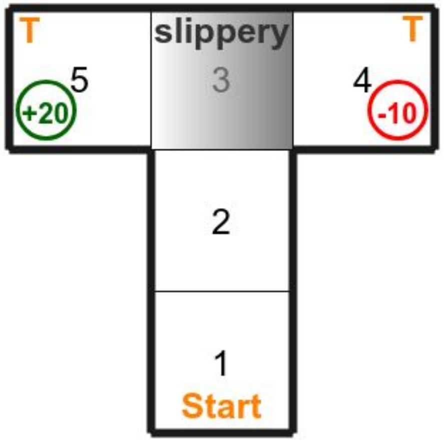
Policy Evaluation: example



Input: Policy

Example: In every state 50% “up” and 50% “left”.

Policy Evaluation: example



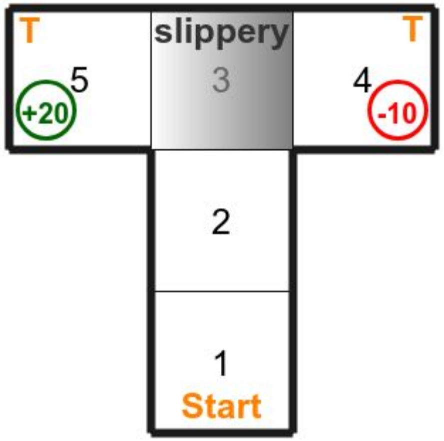
Input: Policy

Example: In every state 50% “up” and 50% “left”.

s	p(a=up)	p(a=down)	p(a=left)	p(s’=right)
1	0.5	0	0.5	0
2	0.5	0	0.5	0
3	0.5	0	0.5	0
4	-	-	-	-
5	-	-	-	-

Terminal states do not have a policy

Policy Evaluation: example

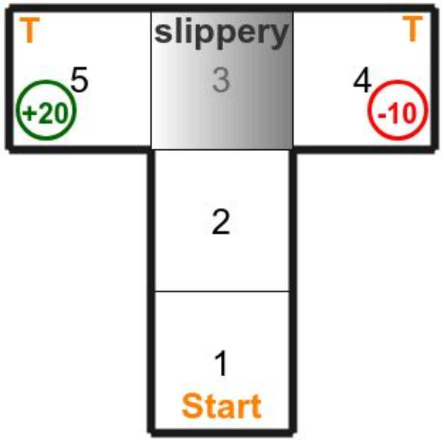


Initialisation: Value table

Example:

s	V(s)
s=1	0
s=2	0
s=3	0
s=4	0
s=5	0

Policy Evaluation: example



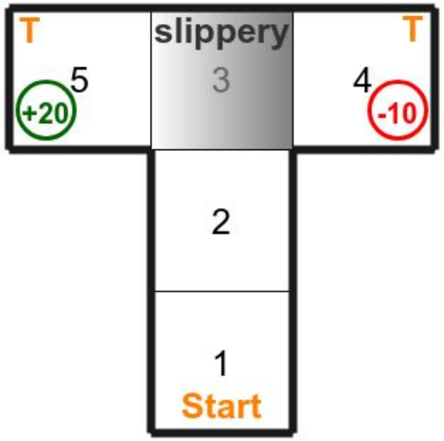
Initialisation: Value table

Example:

s	V(s)
s=1	0
s=2	0
s=3	0
s=4	0
s=5	0

Initialize terminal states to 0

Policy Evaluation: example



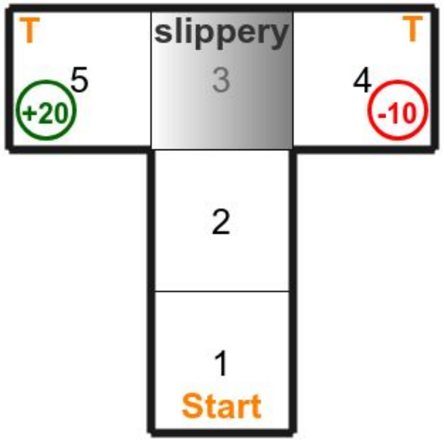
Initialisation: Value table

Example:

s	V(s)
s=1	0
s=2	0
s=3	0
s=4	0
s=5	0

All other states can be randomly initialized

Policy Evaluation: example



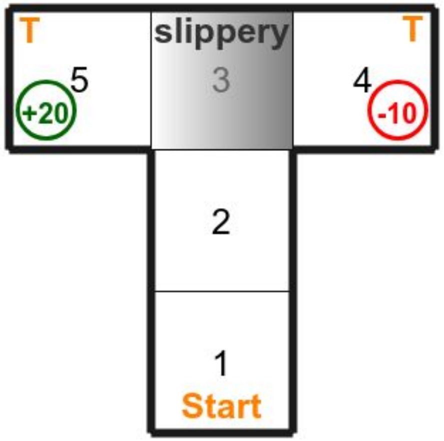
Initialisation: Value table

Example:

s	V(s)
s=1	2
s=2	3
s=3	6
s=4	0
s=5	0

All other states can be randomly initialized

Policy Evaluation: example

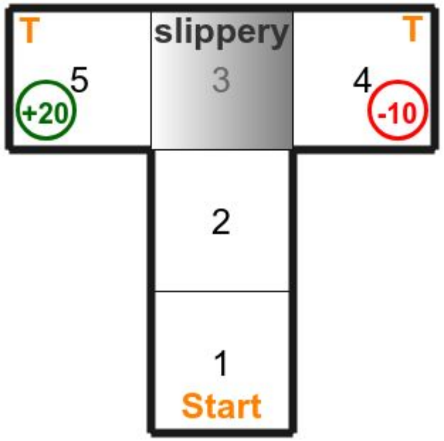


Algorithm:

1. Loop through all states
2. Update each state according to Bellman equation
3. Repeat until convergence.

<u>Bellman update:</u> $V(s) \leftarrow \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r_t + \gamma \cdot V(s') \right]$

Policy Evaluation



Algorithm:

1. Loop through all states
2. Update each state according to Bellman equation
3. Repeat until convergence.

Bellman update: $V(s) \leftarrow \mathbb{E}_{a\sim\pi(\cdot|s)}\mathbb{E}_{s'\sim T(\cdot|a,s)}\left[r_t + \gamma \cdot V(s')\right]$

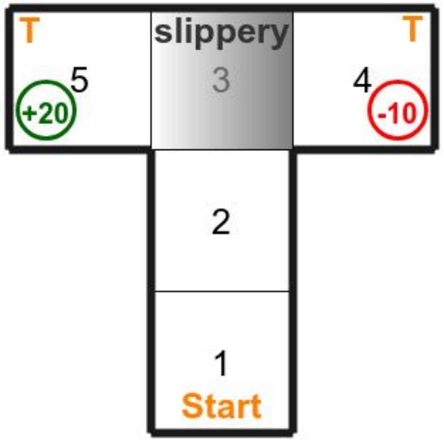
Example: Policy --> 50% up, 50% left, assume $\gamma = 1.0$

Q: *Update state 1*?

A: V(s=1) =

s	V(s)
1	0
2	0
3	0
4	0
5	0

Policy Evaluation



Algorithm:
1. Loop through all states
2. Update each state according to Bellman equation
3. Repeat until convergence.

Bellman update: $V(s) \leftarrow \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r_t + \gamma \cdot V(s') \right]$

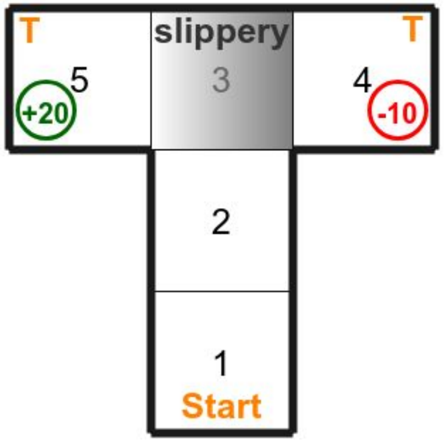
Example: Policy --> 50% up, 50% left, assume γ = 1.0

Q: *Update state 1*?

A: V(s=1) = 0.5 * (-1 + 1.0 * 0) + 0.5 * (-1 + 1.0 * 0) = -1

s	V(s)
1	0
2	0
3	0
4	0
5	0

Policy Evaluation



Algorithm:

1. Loop through all states
2. Update each state according to Bellman equation
3. Repeat until convergence.

<u>Bellman update:</u> $V(s) \leftarrow \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r_t + \gamma \cdot V(s') \right]$

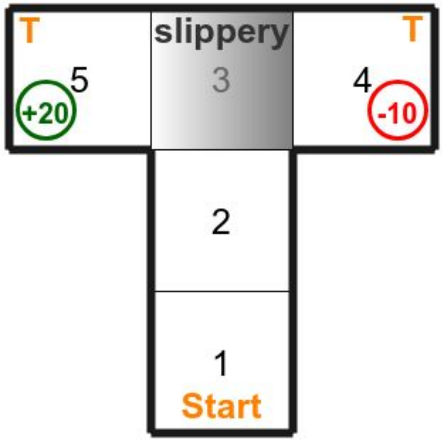
Example: Policy --> 50% up, 50% left, assume $\gamma = 1.0$

s	V(s)
1	-1
2	0
3	0
4	0
5	0

Q: *Update state 1*?

A: V(s=1) = 0.5 * (-1 + 1.0 * 0) + 0.5 * (-1 + 1.0 * 0) = -1

Policy Evaluation



Algorithm:
1. Loop through all states
2. Update each state according to Bellman equation
3. Repeat until convergence.

Bellman update: $V(s) \leftarrow \mathbb{E}_{a\sim\pi(\cdot|s)}\mathbb{E}_{s'\sim T(\cdot|a,s)}\left[r_t + \gamma \cdot V(s')\right]$

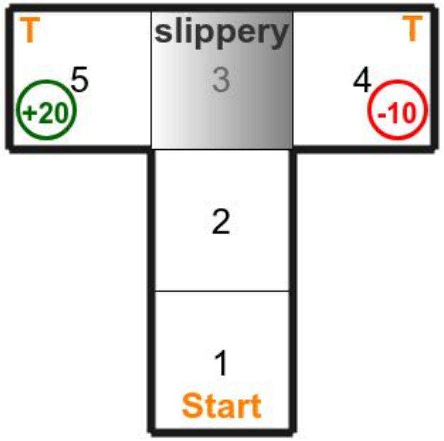
Example: Policy --> 50% up, 50% left, assume γ = 1.0

Q: *Update state 2*? (**stochastic!**)

A: V(s=2) =

s	V(s)
1	-1
2	0
3	0
4	0
5	0

Policy Evaluation



Algorithm:
1. Loop through all states
2. Update each state according to Bellman equation
3. Repeat until convergence.

<u>Bellman update:</u> $V(s) \leftarrow \mathbb{E}_{a\sim\pi(\cdot|s)}\mathbb{E}_{s'\sim T(\cdot|a,s)}\left[r_t + \gamma \cdot V(s')\right]$

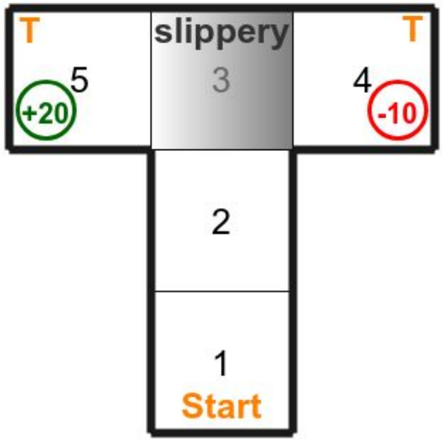
Example: Policy --> 50% up, 50% left, assume γ = 1.0

s	V(s)
1	-1
2	0
3	0
4	0
5	0

Q: *Update state 2*? (**stochastic!**)

A: V(s=2) = 0.5 * (0.8 * (-1 + 1.0*0) + 0.2 * (-10 + 1.0 * 0))
+ 0.5 * (-1 + 1.0 * 0)
= 0.5*(-0.8 - 2.0) + 0.5*-1 = -1.4 - 0.5 = -1.9

Policy Evaluation



Algorithm:

1. Loop through all states
2. Update each state according to Bellman equation
3. Repeat until convergence.

<u>Bellman update:</u> $V(s) \leftarrow \mathbb{E}_{a \sim \pi(\cdot|s)} \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r_t + \gamma \cdot V(s') \right]$

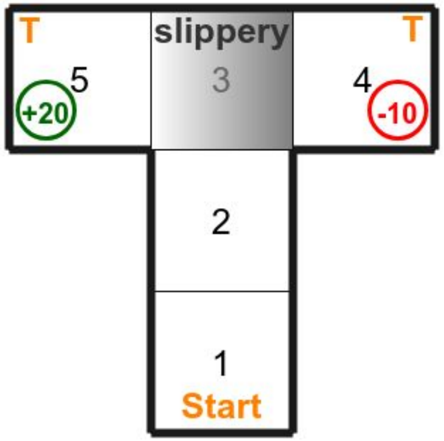
Example: Policy --> 50% up, 50% left, assume γ = 1.0

Q: *Update state 2*? (**stochastic!**)

A: V(s=2) = 0.5 * (0.8 * (-1 + 1.0*0) + 0.2 * (-10 + 1.0 * 0)) + 0.5 * (-1 + 1.0 * 0)
= 0.5*(-0.8 - 2.0) + 0.5*-1 = -1.4 - 0.5 = -1.9

s	V(s)
1	-1
2	-1.9
3	0
4	0
5	0

Policy Evaluation



Algorithm:

1. Loop through all states
2. Update each state according to Bellman equation
3. Repeat until convergence.

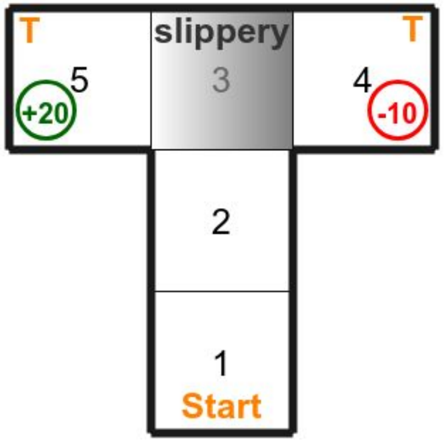
Bellman update: $V(s) \leftarrow \mathbb{E}_{a\sim\pi(\cdot|s)}\mathbb{E}_{s'\sim T(\cdot|a,s)}\left[r_t + \gamma \cdot V(s')\right]$

Example: Policy --> 50% up, 50% left, assume $\gamma = 1.0$

s	V(s)
1	-1
2	-1.9
3	0
4	0
5	0

Update state 3, update state 1, 2, 3, 1, 2, 3 etc.

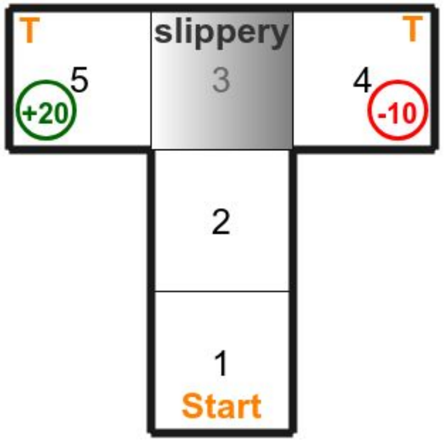
Policy Evaluation



Informal Python code:

```
In [1]: import numpy as np
   ...: V = np.zeros(3) # Initialize values
   ...:
   ...: # Policy evaluation (for policy with 50% left and 50% up)
   ...: for i in range(30):
   ...:     V[0] = 0.5*(-1 + 1.0 * V[0]) + 0.5*(-1 + 1.0 * V[1]) # effect left + effect up
   ...:     V[1] = 0.5 * (-1 + 1.0 * V[1]) + 0.5 * (0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0)) # effect left + effect up
   ...:     V[2] = 0.5 * (20 + 1.0 * 0) + 0.5 * (0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0)) # effect left + effect up
   ...:
   ...: print(np.round(V,1)) # value function of 50/50 left/up policy
```

Policy Evaluation

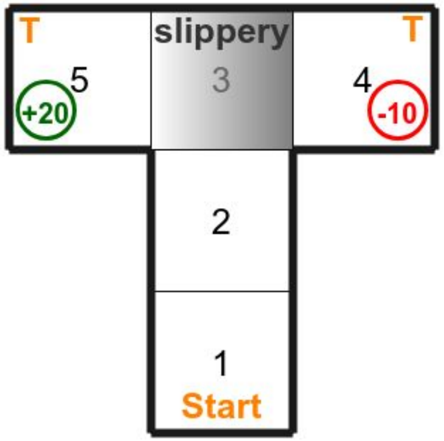


Informal Python code:

```
In [1]: import numpy as np
   ...: V = np.zeros(3) # Initialize values
   ...:
   ...: # Policy evaluation (for policy with 50% left and 50% up)
   ...: for i in range(30):
   ...:     V[0] = 0.5*(-1 + 1.0 * V[0]) + 0.5*(-1 + 1.0 * V[1]) # effect left + effect up
   ...:     V[1] = 0.5 * (-1 + 1.0 * V[1]) + 0.5 * (0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0)) # effect left + effect up
   ...:     V[2] = 0.5 * (20 + 1.0 * 0) + 0.5 * (0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0)) # effect left + effect up
   ...:
   ...: print(np.round(V,1)) # value function of 50/50 left/up policy
```

Python indexing starts at 0
So V[0] refers to V(s=1)

Policy Evaluation



Informal Python code:

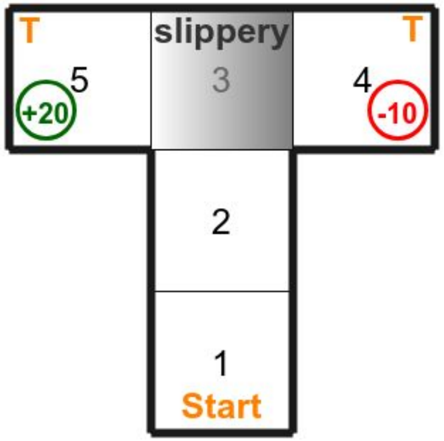
```
In [1]: import numpy as np
   ...: V = np.zeros(3) # Initialize values
   ...:
   ...: # Policy evaluation (for policy with 50% left and 50% up)
   ...: for i in range(30):
   ...:     V[0] = 0.5*(-1 + 1.0 * V[0]) + 0.5*(-1 + 1.0 * V[1]) # effect left + effect up
   ...:     V[1] = 0.5 * (-1 + 1.0 * V[1]) + 0.5 * (0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0)) # effect left + effect up
   ...:     V[2] = 0.5 * (20 + 1.0 * 0) + 0.5 * (0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0)) # effect left + effect up
   ...:
   ...: print(np.round(V,1)) # value function of 50/50 left/up policy
```

Python indexing starts at 0
So V[0] refers to V(s=1)

No stopping criteria (for readability)

30 epochs is more than enough to converge

Policy Evaluation



Informal Python code:

```
In [1]: import numpy as np
   ...: V = np.zeros(3) # Initialize values
   ...:
   ...: # Policy evaluation (for policy with 50% left and 50% up)
   ...: for i in range(30):
   ...:     V[0] = 0.5*(-1 + 1.0 * V[0]) + 0.5*(-1 + 1.0 * V[1]) # effect left + effect up
   ...:     V[1] = 0.5 * (-1 + 1.0 * V[1]) + 0.5 * (0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0)) # effect left + effect up
   ...:     V[2] = 0.5 * (20 + 1.0 * 0) + 0.5 * (0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0)) # effect left + effect up
   ...:
   ...: print(np.round(V,1)) # value function of 50/50 left/up policy
```

Python indexing starts at 0
So V[0] refers to V(s=1)

No stopping criteria (for readability)

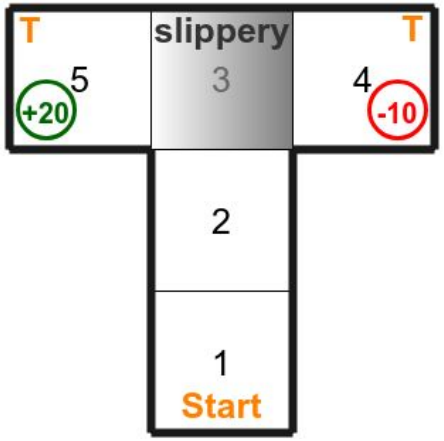
30 epochs is more than enough to converge

$$V(s) \leftarrow \mathbb{E}_{a\sim\pi(\cdot|s)}\mathbb{E}_{s'\sim T(\cdot|a,s)}\left[r_t + \gamma \cdot V(s')\right]$$

Manually wrote this equation for every state

Only wrote the terms for which pi(a|s) and T(s'|s,a) are positive

Policy Evaluation



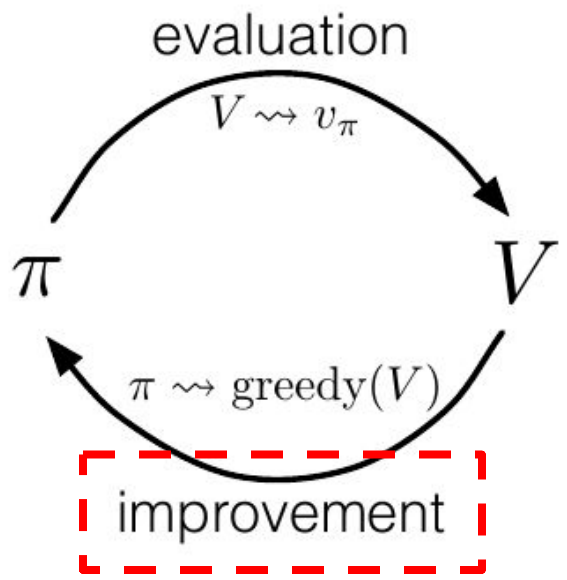
Informal Python code (without stopping criteria):

```
In [1]: import numpy as np
   ...: V = np.zeros(3) # Initialize values
   ...:
   ...: # Policy evaluation (for policy with 50% left and 50% up)
   ...: for i in range(30):
   ...:     V[0] = 0.5*(-1 + 1.0 * V[0]) + 0.5*(-1 + 1.0 * V[1]) # effect left + effect up
   ...:     V[1] = 0.5 * (-1 + 1.0 * V[1]) + 0.5 * (0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0)) # effect left + effect up
   ...:     V[2] = 0.5 * (20 + 1.0 * 0) + 0.5 * (0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0)) # effect left + effect up
   ...:
   ...: print(np.round(V,1)) # value function of 50/50 left/up policy
[ 5.7  7.7 14.3]
```

So this is the value function belonging to a 50/50 left/up policy →

s	V(s)
1	5.7
2	7.7
3	14.3
4	0
5	0

Dynamic Programming



Policy improvement

Given a $V^{\pi}(s)$, how can we find an improved π?

Policy improvement

Given a $V^\pi(s)$, how can we find an improved π?

Main insights:

- The optimal policy is always *greedy*.
- There is always one action with the highest value
  (or multiple with equally high value estimate)

Policy improvement

Given a $V^{\pi}(s)$, how can we find an improved π?

Main insights:

- The optimal policy is always *greedy*.
- There is always one action with the highest value
  (or multiple with equally high value estimate)

Policy improvement:

- Simply acting greedy with respect to $V^{\pi}(s)$ or $Q^{\pi}(s,a)$

For Q:

For V:

Policy improvement

Given a $V^\pi(s)$, how can we find an improved π?

Main insights:

- The optimal policy is always *greedy*.
- There is always one action with the highest value
  (or multiple with equally high value estimate)

Policy improvement:

- Simply acting greedy with respect to $V^\pi(s)$ or $Q^\pi(s,a)$

For Q: $$\pi(s) \leftarrow \arg\max_a Q(s,a)$$

For V:

Policy improvement

Given a $V^{\pi}(s)$, how can we find an improved π?

Main insights:

- The optimal policy is always *greedy*.
- There is always one action with the highest value
  (or multiple with equally high value estimate)

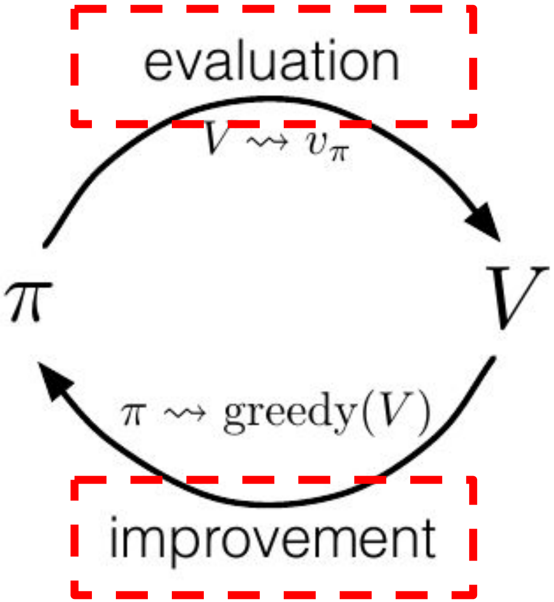
Policy improvement:

- Simply acting greedy with respect to $V^{\pi}(s)$ or $Q^{\pi}(s,a)$

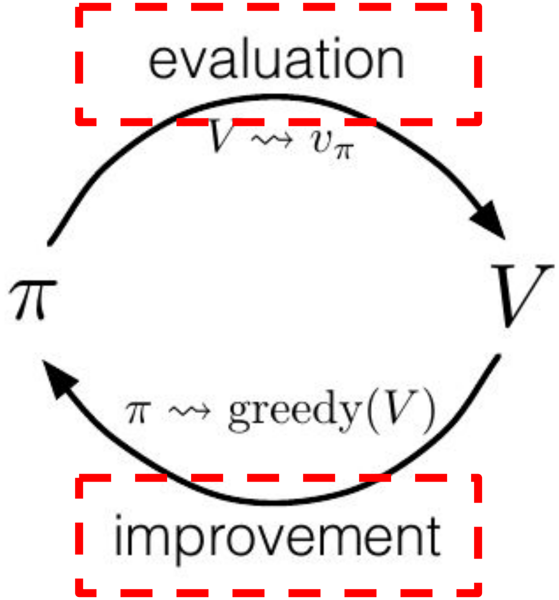
For Q: $$\pi(s) \leftarrow \arg\max_a Q(s,a)$$

For V: $$\pi(s) \leftarrow \arg\max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$

Dynamic Programming

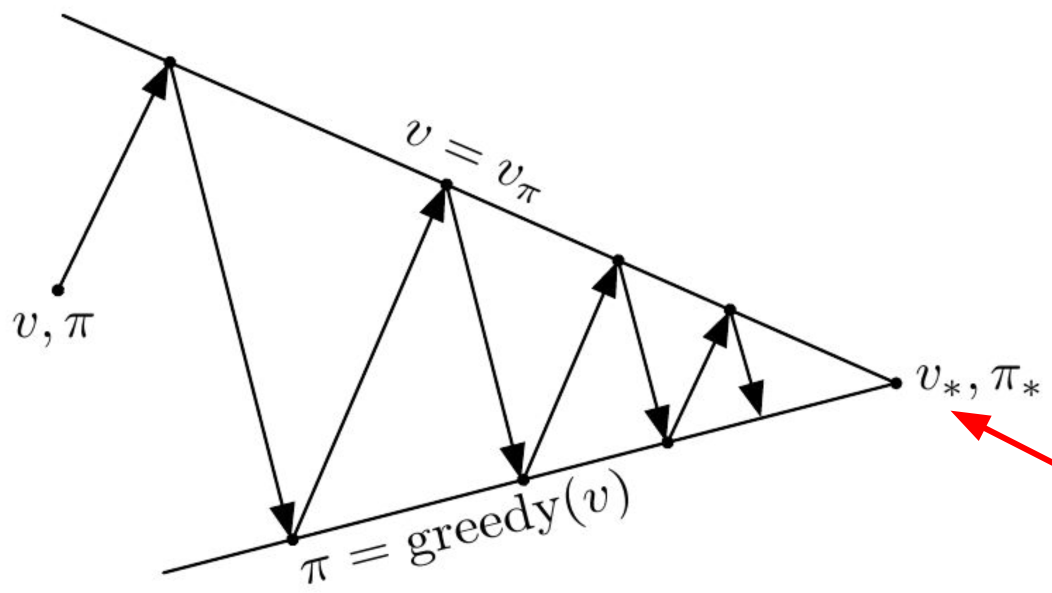


Dynamic Programming



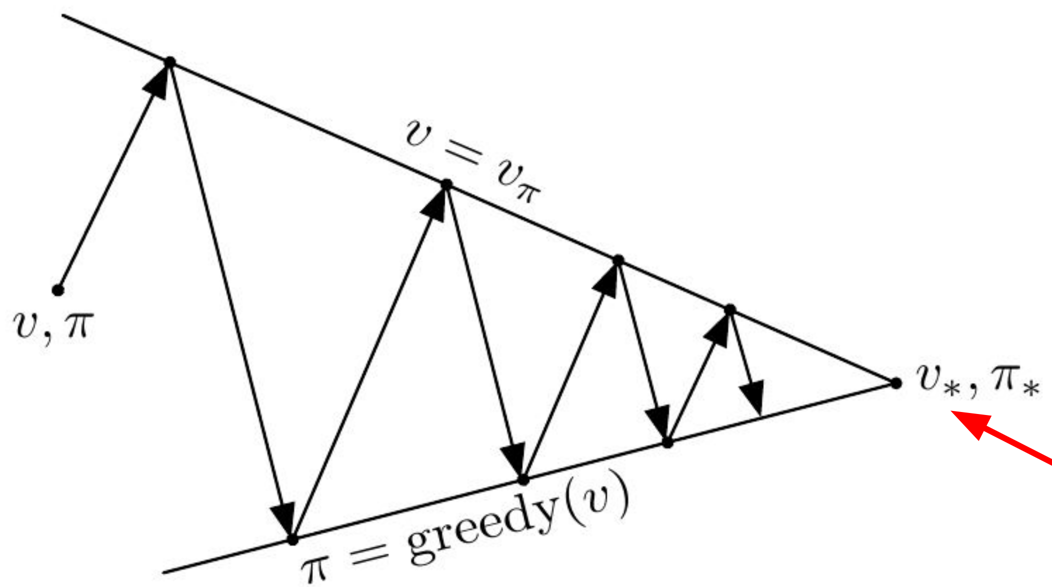
Dynamic Programming
=
iterating policy evaluation and policy improvement

Dynamic Programming



Converges to the optimal value function and policy

Dynamic Programming



Converges to the optimal value function and policy

But *how* may we iterate both?

Dynamic Programming

Two approaches

Policy iteration

Value iteration

Dynamic Programming

Two approaches

Policy iteration

1. Policy evaluation: *until convergence*
2. Policy improvement

Value iteration

1. Policy evaluation: *for 1 cycle*
2. Policy improvement

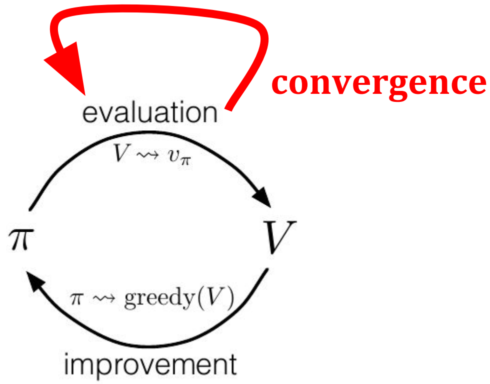
Dynamic Programming

Two approaches

Policy iteration

1. Policy evaluation: *until convergence*

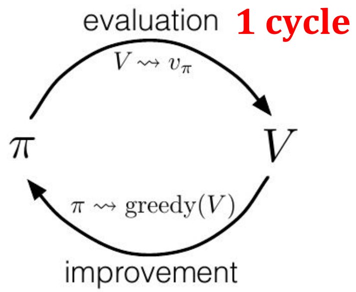
2. Policy improvement



Value iteration

1. Policy evaluation: *for 1 cycle*

2. Policy improvement



Dynamic Programming

Two approaches

Policy iteration

1. Policy evaluation: *until convergence*
2. Policy improvement

See lecture notes (only conceptually)

Value iteration

1. Policy evaluation: *for 1 cycle*
2. Policy improvement

Covered here (and for practical)

Value iteration

Loop until convergence:

1. Policy evaluation (1 cycle):

   For all s in state space:

$$V(s) \leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$

2. Policy improvement:

   For all s in state space:

$$\pi(s) \leftarrow \arg\max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$

Value iteration

Loop until convergence:

1. Policy evaluation (1 cycle):

   For all s in state space:

   $$V(s) \leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$

2. Policy improvement:

   For all s in state space:

   $$\pi(s) \leftarrow \arg\max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$

But:

When we represent the policy in a smart way

-

we can write these two equations in one line!

Implicit policies

<u>Explicit policy representation:</u>

Table with mapping: $\mathbf{s} \longrightarrow \mathbf{p(A)}$

Implicit policies

Implicit policy representation:

Derive policy from value table (only store value table as solution)

Example:

Implicit policies

Implicit policy representation:

Derive policy from value table (only store value table as solution)

Example:

Value table

s	V(s)
1	2
2	4
3	3

Implicit policies

<u>Implicit policy representation</u>:

Derive policy from value table (only store value table as solution)

Example:

Value table → *Policy (function of value table)*

s	V(s)
1	2
2	4
3	3

$$\pi(s) = \arg\max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$

Example for greedy policy

Implicit policies

<u>Implicit policy representation</u>:

Derive policy from value table (only store value table as solution)

Example:

State-action value table → *Policy (function of value table)*

	a=up	a=down	..
s=1	Q(s,a)=5	3	
s=2	9	4	
s=3	4	2	

Implicit policies

<u>Implicit policy representation</u>:

Derive policy from value table (only store value table as solution)

Example:

State-action value table → *Policy (function of value table)*

	a=up	a=down	..
s=1	Q(s,a)=5	3	
s=2	9	4	
s=3	4	2	

$$\pi(s) \leftarrow \arg\max_a Q(s,a)$$

Example for greedy policy

Value iteration

Two ingredients:

- Alternate policy evaluation (1 sweep) and policy improvement (1 sweep)
- Implicitly represent the policy with a value table.

Effect: can write the update *as a single equation*!

Value iteration

Two ingredients:

- Alternate policy evaluation (1 sweep) and policy improvement (1 sweep)
- Implicitly represent the policy with a value table.

Effect: can write the update *as a single equation*!

Algorithm:

Loop until convergence:

For each s in state space:

$$V(s) \leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$

Value iteration

Two ingredients:

- Alternate policy evaluation (1 sweep) and policy improvement (1 sweep)
- Implicitly represent the policy with a value table.

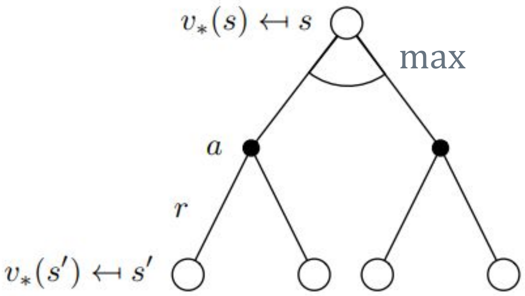
Effect: can write the update *as a single equation*!

Algorithm:

Loop until convergence:

For each s in state space:

$$V(s) \leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$



Back-up diagram

Value iteration

Two ingredients:

- Alternate policy evaluation (1 sweep) and policy improvement (1 sweep)
- Implicitly represent the policy with a value table.

Effect: can write the update *as a single equation*!

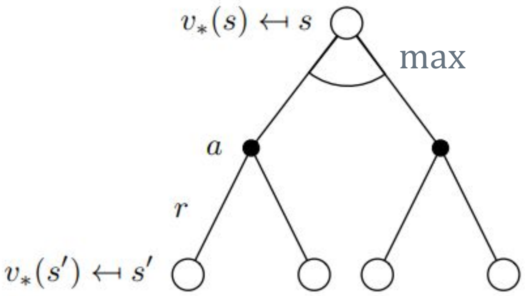
Algorithm:

Loop until convergence:

For each s in state space:

$$V(s) \leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$

Returns optimal value function V*(s)!



Back-up diagram

Value iteration

<u>Two ingredients</u>:

- Alternate policy evaluation (1 sweep) and policy improvement (1 sweep)
- Implicitly represent the policy with a value table.

Effect: can write the update *as a single equation*!

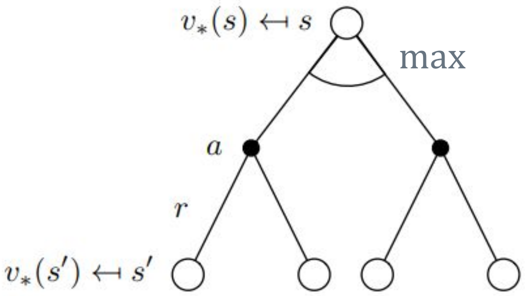
<u>Algorithm:</u>

Very straightforward algorithm!

Loop until convergence:

For each s in state space:

$$V(s) \leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$



Back-up diagram

Returns optimal value function V*(s)!

Value iteration

Algorithm 3: Value iteration

Input: Some deterministic policy $\pi(s)$, small threshold θ
Result: The optimal greedy policy $\pi^\star(s)$
Initialization: A value table $V(s)$ with arbitrary entries, except $V(s = terminal) = 0$
repeat
  $\Delta \leftarrow 0$
  for *each* $s \in \mathcal{S}$ **do**
    $x \leftarrow V(s)$
    $V(s) \leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)}\left[r + \gamma \cdot V(s')\right]$ `/* Eq. 18 */`
    $\Delta \leftarrow \max(\Delta, |x - V(s)|)$
  end
until $\Delta < \theta$;
$\pi^\star(s) = \arg\max_a \mathbb{E}_{s' \sim T(\cdot|a,s)}\left[r + \gamma \cdot V(s')\right] \quad \forall s \in \mathcal{S}$
Return $\pi^\star(s)$

Full pseudocode in lecture notes

Q-Value iteration

= same algorithm but with state-action values!

Q-Value iteration

= same algorithm but with state-action values!

<u>Algorithm</u>

Loop until convergence:

    For each s in state space:

        For each a in action space:

$$Q(s,a) \leftarrow \mathbb{E}_{s' \sim T}\big[r_t + \gamma \max_{a} Q(s', a')\big]\big]$$

Returns optimal state-action value function Q*(s,a)!

Q-Value iteration

= same algorithm but with state-action values!

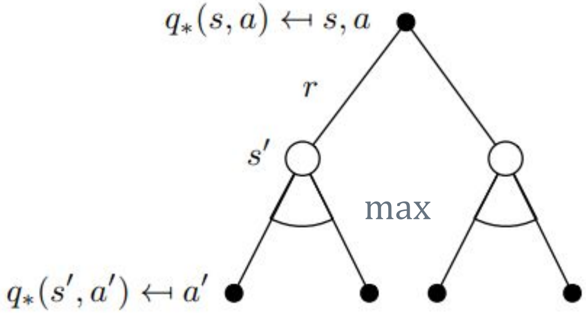
<u>Algorithm</u>

Loop until convergence:

For each s in state space:

For each a in action space:

$$Q(s,a) \leftarrow \mathbb{E}_{s'\sim T}\left[r_t + \gamma \max_a Q(s',a')\right]]$$



Back-up diagram

Returns optimal state-action value function Q*(s,a)!

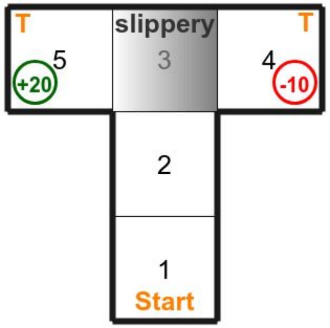
Q-Value iteration

Algorithm 4: Q-value iteration

Input: Some deterministic policy $\pi(s)$, small threshold θ
Result: The optimal greedy policy $\pi^\star(s)$
Initialization: A state-action value table $Q(s,a)$ with arbitrary entries, except $Q(s = terminal, a) = 0, \forall a$
repeat
  $\Delta \leftarrow 0$
  for *each* $s \in \mathcal{S}$ **do**
    for *each* $a \in \mathcal{A}$ **do**
      $x \leftarrow Q(s,a)$
      $Q(s,a) \leftarrow \mathbb{E}_{s' \sim T}\left[r_t + \gamma \max_{a'} Q(s', a')\right]$ `/* Eq.` $\boxed{19}$ `*/`
      $\Delta \leftarrow \max(\Delta, |x - Q(s,a)|)$
    end
  end
until $\Delta < \theta$;
$\pi^\star(s) = \arg\max_a Q(s,a) \quad \forall s \in \mathcal{S}$
Return $\pi^\star(s)$

Full pseudocode in lecture notes

Value iteration example



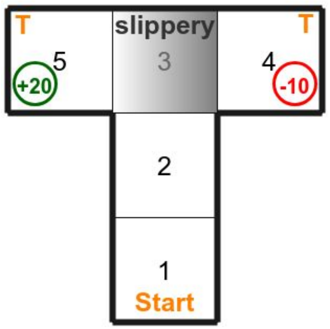
Task

s	V*(s)
1	?
2	?
3	?

Value table

Small problem: can reason what optimal value should be

Value iteration example



Task

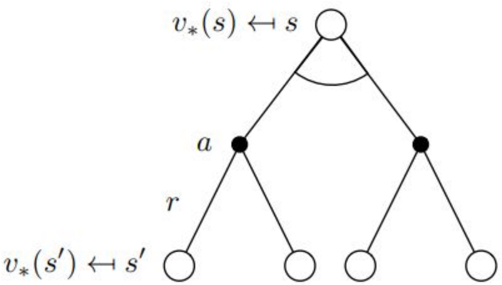
s	V*(s)
1	?
2	?
3	?

Value table

Small problem: can reason what optimal value should be

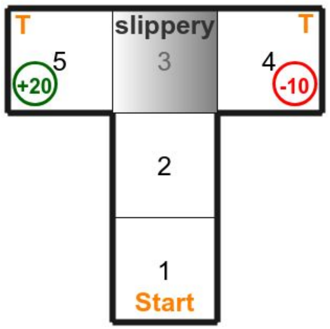
Q: Compute V*(s=3).

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

Value iteration example

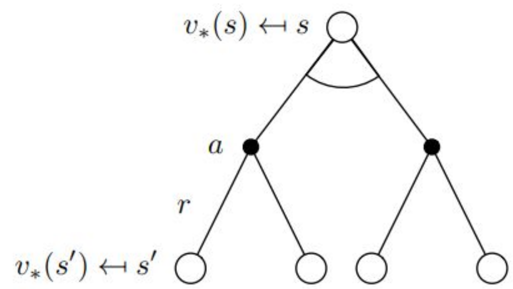


Task

s	V*(s)
1	?
2	?
3	?

Value table

Hints



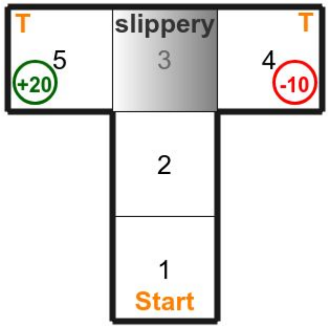
$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

<u>Small problem: can reason what optimal value should be</u>

Q: Compute V*(s=3).

A: Best action = left

Value iteration example



Task

s	V*(s)
1	?
2	?
3	?

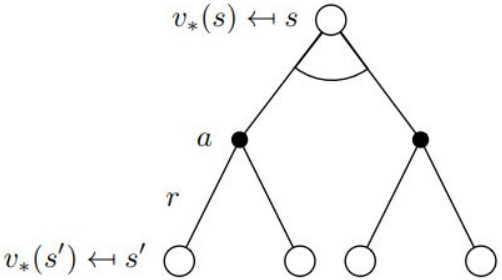
Value table

Small problem: can reason what optimal value should be

Q: Compute V*(s=3).

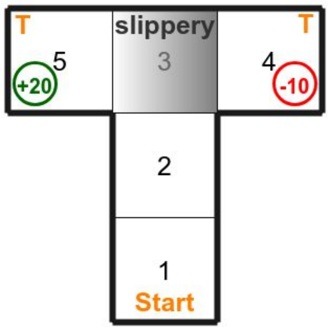
A: Best action = left
This always reaches state 5

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

Value iteration example



Task

s	V*(s)
1	?
2	?
3	?

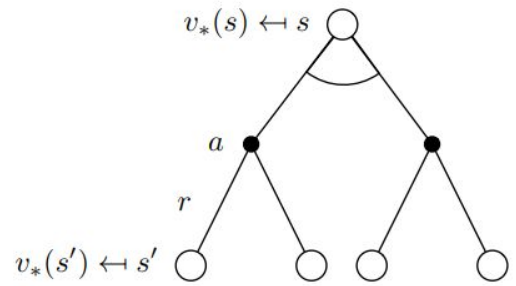
Value table

<u>Small problem: can reason what optimal value should be</u>

Q: Compute V*(s=3).

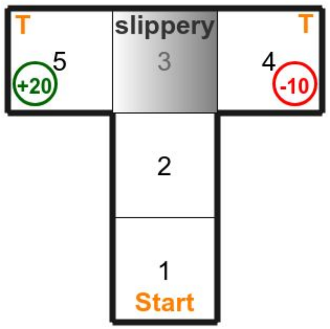
A: Best action = left
This always reaches state 5
Reward = 20 and terminates (so V(5)=0)

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

Value iteration example

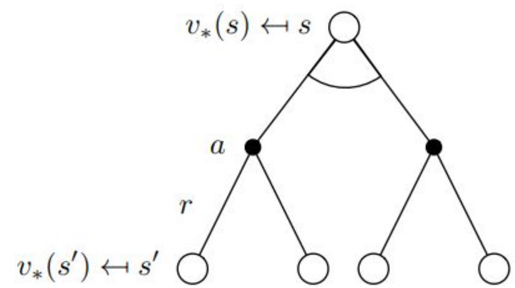


Task

s	V*(s)
1	?
2	?
3	?

Value table

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

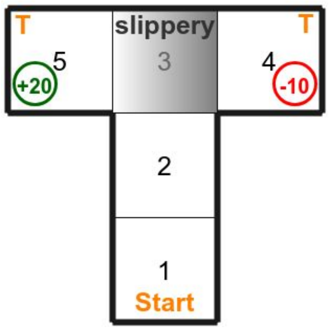
<u>Small problem: can reason what optimal value should be</u>

Q: Compute V*(s=3).

A: Best action = left
This always reaches state 5
Reward = 20 and terminates (so V(5)=0)

V*(3) = 20 + 1.0 * 0 = 20

Value iteration example

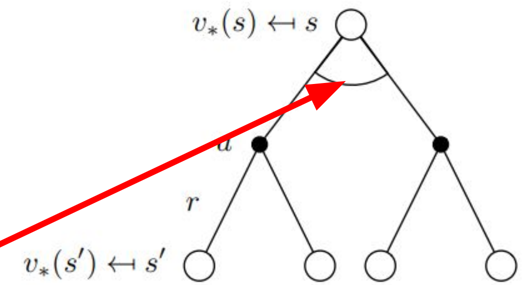


Task

s	V*(s)
1	?
2	?
3	?

Value table

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

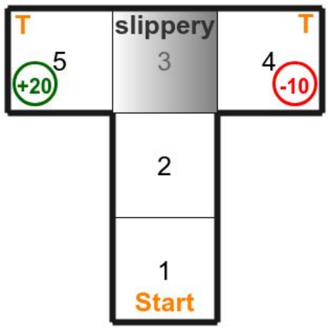
<u>Small problem: can reason what optimal value should be</u>

Q: Compute V*(s=3).

A: Best action = left
This always reaches state 5
Reward = 20 and terminates (so V(5)=0)

V*(3) = 20 + 1.0 * 0 = 20

Value iteration example

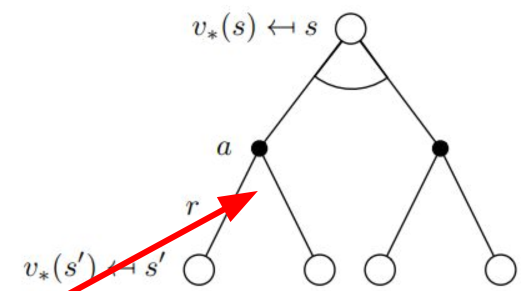


Task

s	V*(s)
1	?
2	?
3	?

Value table

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in S} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

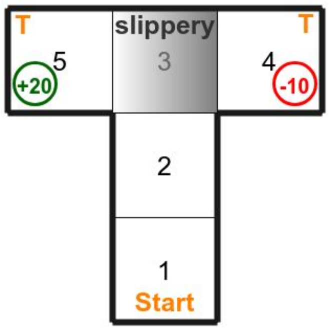
<u>Small problem: can reason what optimal value should be</u>

Q: Compute V*(s=3).

A: Best action = left
This always reaches state 5
Reward = 20 and terminates (so V(5)=0)

V*(3) = 20 + 1.0 * 0 = 20

Value iteration example

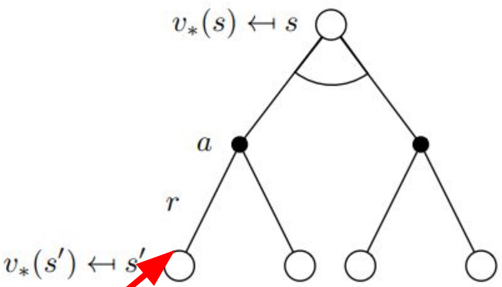


Task

s	V*(s)
1	?
2	?
3	?

Value table

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

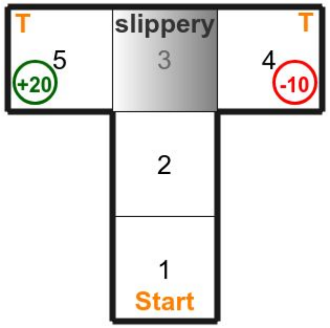
<u>Small problem: can reason what optimal value should be</u>

Q: Compute V*(s=3).

A: Best action = left
This always reaches state 5
Reward = 20 and terminates (so V(5)=0)

V*(3) = 20 + 1.0 * 0 = 20

Value iteration example



Task

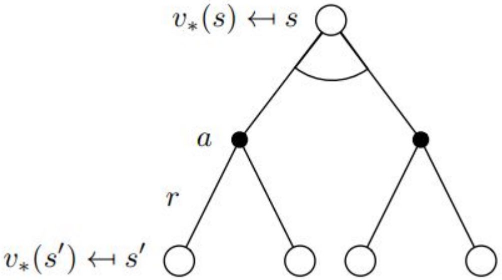
s	V*(s)
1	?
2	?
3	20.0

Value table

<u>Small problem: can reason what optimal value should be</u>

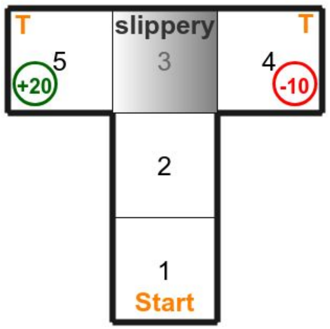
Q: Compute V*(s=2).

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

Value iteration example

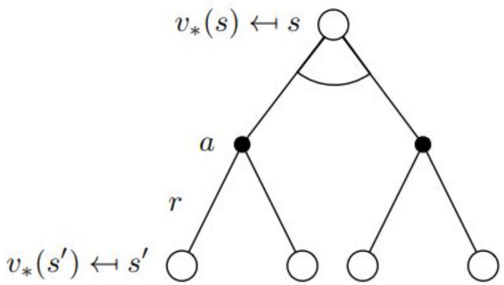


Task

s	V*(s)
1	?
2	?
3	20.0

Value table

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

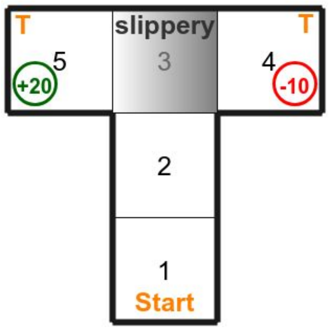
<u>Small problem: can reason what optimal value should be</u>

Q: Compute V*(s=2).

A: Best action = up
80% reaches s=3 with r=-1
20% slips and reaches s=4 with r=-10

V*(2) = 0.8(-1 + 1.0*20) + 0.2(-10+1.0*0) = 13.2

Value iteration example



Task

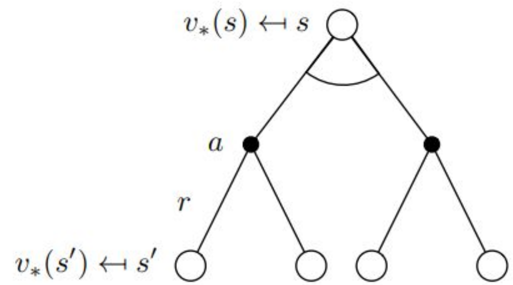
s	V*(s)
1	?
2	13.2
3	20.0

Value table

Small problem: can reason what optimal value should be

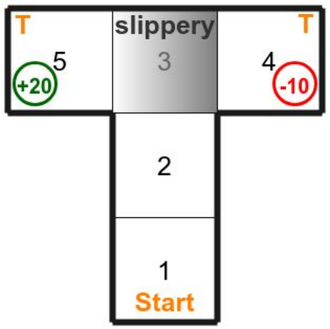
Q: Compute V*(s=1).

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

Value iteration example

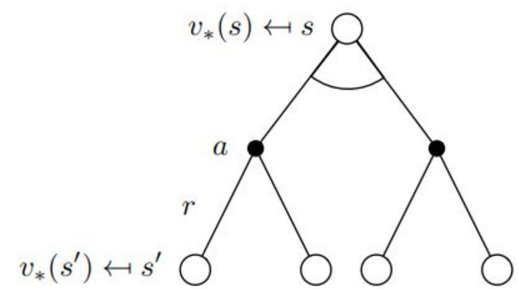


Task

s	V*(s)
1	?
2	13.2
3	20.0

Value table

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\big[r + \gamma \cdot V(s')\big]$$

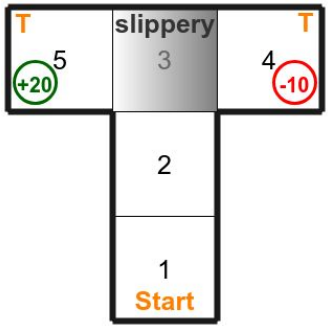
<u>Small problem: can reason what optimal value should be</u>

Q: Compute V*(s=1).

A: Best action = up
Always reaches s=2, with r=-1.

V*(1) = -1 + 1.0 * 13.2 = 12.2

Value iteration example



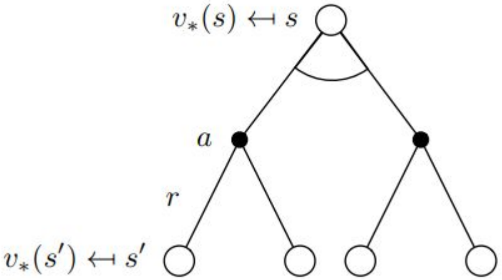
Task

s	V*(s)
1	12.2
2	13.2
3	20.0

Value table

Does value iteration give the same solution?

Hints



$$V(s) \leftarrow \max_a \sum_{s' \in \mathcal{S}} T(s'|s,a)\left[r + \gamma \cdot V(s')\right]$$

Value iteration example

```
In [1]: import numpy as np
   ...: V = np.zeros(3) # Initialize values
   ...:
   ...: # Value iteration (policy evaluation + policy improvement = dynamic programming)
   ...: for i in range(30):
   ...:     V[0] = np.max([-1 + 1.0 * V[1], # effect of up in state 1
   ...:                   -1 + 1.0 * V[0], # effect of down in state 1
   ...:                   -1 + 1.0 * V[0], # effect of left in state 1
   ...:                   -1 + 1.0 * V[0]]) # effect of right in state 1
   ...:     V[1] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0), # effect of up in state 2
   ...:                   -1 + 1.0 * V[0], # etc.
   ...:                   -1 + 1.0 * V[1],
   ...:                   -1 + 1.0 * V[1]])
   ...:     V[2] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0),
   ...:                   -1 + 1.0 * V[1],
   ...:                   20 + 1.0 * 0,
   ...:                   -10 + 1.0*0])
   ...:
   ...: print(np.round(V,1)) # optimal value function
```

Value iteration example

Initialize value vector

```
In [1]: import numpy as np
   ...: V = np.zeros(3) #initialize values
   ...:
   ...: # Value iteration (policy evaluation + policy improvement = dynamic programming)
   ...: for i in range(30):
   ...:     V[0] = np.max([-1 + 1.0 * V[1], # effect of up in state 1
   ...:                    -1 + 1.0 * V[0], # effect of down in state 1
   ...:                    -1 + 1.0 * V[0], # effect of left in state 1
   ...:                    -1 + 1.0 * V[0]]) # effect of right in state 1
   ...:     V[1] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0), # effect of up in state 2
   ...:                    -1 + 1.0 * V[0], # etc.
   ...:                    -1 + 1.0 * V[1],
   ...:                    -1 + 1.0 * V[1]])
   ...:     V[2] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0),
   ...:                    -1 + 1.0 * V[1],
   ...:                    20 + 1.0 * 0,
   ...:                    -10 + 1.0*0])
   ...:
   ...: print(np.round(V,1)) # optimal value function
```

Value iteration example

Initialize value vector

```
In [1]: import numpy as np
   ...: V = np.zeros(3) # initialize values
   ...:
   ...: # Value iteration (policy evaluation + policy improvement = dynamic programming)
   ...: for i in range(30):
   ...:     V[0] = np.max([-1 + 1.0 * V[1], # effect of up in state 1
   ...:                    -1 + 1.0 * V[0], # effect of down in state 1
   ...:                    -1 + 1.0 * V[0], # effect of left in state 1
   ...:                    -1 + 1.0 * V[0]]) # effect of right in state 1
   ...:     V[1] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0), # effect of up in state 2
   ...:                    -1 + 1.0 * V[0], # etc.
   ...:                    -1 + 1.0 * V[1],
   ...:                    -1 + 1.0 * V[1]])
   ...:     V[2] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0),
   ...:                    -1 + 1.0 * V[1],
   ...:                    20 + 1.0 * 0,
   ...:                    -10 + 1.0*0])
   ...:
   ...: print(np.round(V,1)) # optimal value function
```

Manually wrote Bellman optimality equation

$$V(s) \leftarrow \max_a \mathbb{E}_{s' \sim T(\cdot|a,s)} \left[r + \gamma \cdot V(s') \right]$$

Only wrote the terms for which T(s'|s,a) is positive

Value iteration example

```
In [1]: import numpy as np
   ...: V = np.zeros(3) # Initialize values
   ...:
   ...: # Value iteration (policy evaluation + policy improvement = dynamic programming)
   ...: for i in range(30):
   ...:     V[0] = np.max([-1 + 1.0 * V[1], # effect of up in state 1
   ...:                   -1 + 1.0 * V[0], # effect of down in state 1
   ...:                   -1 + 1.0 * V[0], # effect of left in state 1
   ...:                   -1 + 1.0 * V[0]]) # effect of right in state 1
   ...:     V[1] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0), # effect of up in state 2
   ...:                   -1 + 1.0 * V[0], # etc.
   ...:                   -1 + 1.0 * V[1],
   ...:                   -1 + 1.0 * V[1]])
   ...:     V[2] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0),
   ...:                   -1 + 1.0 * V[1],
   ...:                   20 + 1.0 * 0,
   ...:                   -10 + 1.0*0])
   ...:
   ...: print(np.round(V,1)) # optimal value function
[12.2 13.2 20. ]
```

Yes!
Exactly finds the same optimal values

Value iteration example

```
In [1]: import numpy as np
   ...: V = np.zeros(3) # Initialize values
   ...:
   ...: # Value iteration (policy evaluation + policy improvement = dynamic programming)
   ...: for i in range(30):
   ...:     V[0] = np.max([-1 + 1.0 * V[1], # effect of up in state 1
   ...:                    -1 + 1.0 * V[0], # effect of down in state 1
   ...:                    -1 + 1.0 * V[0], # effect of left in state 1
   ...:                    -1 + 1.0 * V[0]]) # effect of right in state 1
   ...:     V[1] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0), # effect of up in state 2
   ...:                    -1 + 1.0 * V[0], # etc.
   ...:                    -1 + 1.0 * V[1],
   ...:                    -1 + 1.0 * V[1]])
   ...:     V[2] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0),
   ...:                    -1 + 1.0 * V[1],
   ...:                    20 + 1.0 * 0,
   ...:                    -10 + 1.0*0])
   ...:
   ...: print(np.round(V,1)) # optimal value function
[12.2 13.2 20. ]
```

Yes!
Exactly finds the same optimal values

Principle works - applicable to larger problems

Value iteration example

```
In [1]: import numpy as np
   ...: V = np.zeros(3) # Initialize values
   ...:
   ...: # Value iteration (policy evaluation + policy improvement = dynamic programming)
   ...: for i in range(30):
   ...:     V[0] = np.max([-1 + 1.0 * V[1], # effect of up in state 1
   ...:                      -1 + 1.0 * V[0], # effect of down in state 1
   ...:                      -1 + 1.0 * V[0], # effect of left in state 1
   ...:                      -1 + 1.0 * V[0]]) # effect of right in state 1
   ...:     V[1] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0), # effect of up in state 2
   ...:                      -1 + 1.0 * V[0], # etc.
   ...:                      -1 + 1.0 * V[1],
   ...:                      -1 + 1.0 * V[1]])
   ...:     V[2] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0),
   ...:                      -1 + 1.0 * V[1],
   ...:                      20 + 1.0 * 0,
   ...:                      -10 + 1.0*0])
   ...:
   ...: print(np.round(V,1)) # optimal value function
[12.2 13.2 20. ]
```

Sloppy example:

- No generic code (separation of algorithm and environment)
- No stopping criteria

Value iteration example

```
In [1]: import numpy as np
   ...: V = np.zeros(3) # Initialize values
   ...:
   ...: # Value iteration (policy evaluation + policy improvement = dynamic programming)
   ...: for i in range(30):
   ...:     V[0] = np.max([-1 + 1.0 * V[1], # effect of up in state 1
   ...:                   -1 + 1.0 * V[0], # effect of down in state 1
   ...:                   -1 + 1.0 * V[0], # effect of left in state 1
   ...:                   -1 + 1.0 * V[0]]) # effect of right in state 1
   ...:     V[1] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0), # effect of up in state 2
   ...:                   -1 + 1.0 * V[0], # etc.
   ...:                   -1 + 1.0 * V[1],
   ...:                   -1 + 1.0 * V[1]])
   ...:     V[2] = np.max([0.8 * (-1 + 1.0* V[2]) + 0.2 *(-10 + 1.0*0),
   ...:                   -1 + 1.0 * V[1],
   ...:                   20 + 1.0 * 0,
   ...:                   -10 + 1.0*0])
   ...:
   ...: print(np.round(V,1)) # optimal value function
[12.2 13.2 20. ]
```

Sloppy example:

- No generic code (separation of algorithm and environment)
- No stopping criteria

We will do this the right way in the assignment

Discussion: DP versus Search

Dynamic Programming

Tree/Graph Search

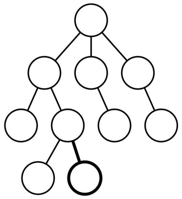
Discussion: DP versus Search

Dynamic Programming | Tree/Graph Search

- Global solution

s	V*(s)
1	12.2
2	13.2

- Local solution



Discussion: DP versus Search

Dynamic Programming

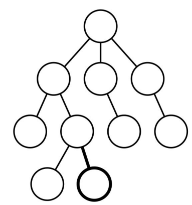
- Global solution

s	V*(s)
1	12.2
2	13.2

- High memory requirement: O(|S|) or O(|S|x|A|)

Tree/Graph Search

- Local solution



- Memory requirement varies, but usually lower than DP

Discussion: DP versus Search

Dynamic Programming | Tree/Graph Search

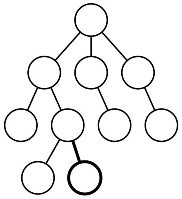
- Global solution

s	V*(s)
1	12.2
2	13.2

- High memory requirement: O(|S|) or O(|S|x|A|)

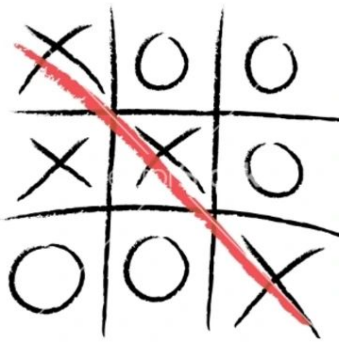
Curse of dimensionality:
Number of unique states grows exponentially in problem size (number of variables that the state describes)

- Local solution



- Memory requirement varies, but usually lower than DP

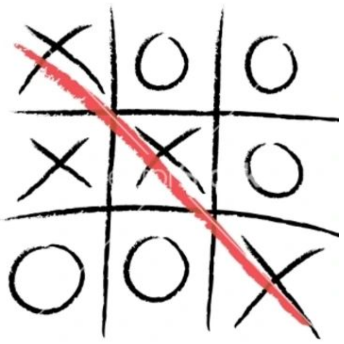
Curse of dimensionality



Imagine our task is a bit more complex, like a game of Tic-tac-toe.

- A state represents a combination of 9 variables (each location on the board)
- Each variable can take three values (X, O or empty)

Curse of dimensionality



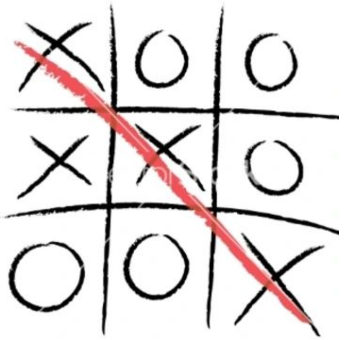
Imagine our task is a bit more complex, like a game of Tic-tac-toe.

- A state represents a combination of 9 variables (each location on the board)
- Each variable can take three values (X, O or empty)

Q: How many unique states are there?

A:

Curse of dimensionality



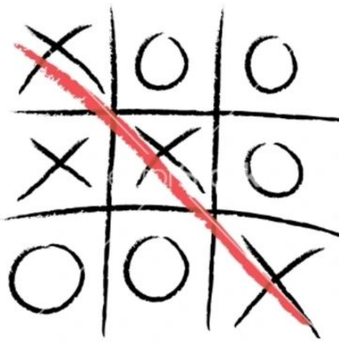
Imagine our task is a bit more complex, like a game of Tic-tac-toe.

- A state represents a combination of 9 variables (each location on the board)
- Each variable can take three values (X, O or empty)

Q: How many unique states are there?

A: $3^9 = 19.683$

Curse of dimensionality



Imagine our task is a bit more complex, like a game of Tic-tac-toe.

- A state represents a combination of 9 variables (each location on the board)
- Each variable can take three values (X, O or empty)

Q: How many unique states are there?

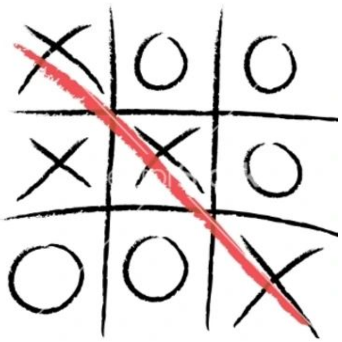
A: $3^9 = 19.683$

Now imagine we make the board slightly bigger, to 4x4

Q: How many unique states are there now?

A:

Curse of dimensionality



Imagine our task is a bit more complex, like a game of Tic-tac-toe.

- A state represents a combination of 9 variables (each location on the board)
- Each variable can take three values (X, O or empty)

Q: How many unique states are there?

A: $3^9 = 19.683$

Now imagine we make the board slightly bigger, to 4x4

Q: How many unique states are there now?

A: $3^{16} = 43.046.721$

<u>The size of the state space grows exponentially in the number of underlying variables.</u>

Summary

- **Markov Decision Process**

- **Bellman equation**

- **Dynamic Programming**

Summary

- **Markov Decision Process**
  - Powerful (generic) paradigm to define sequential tasks.
  - Can deal with multiple goals (and trading-off between them through utility/value).
  - Can deal with stochastic dynamics.

- **Bellman equation**

- **Dynamic Programming**

Summary

- **Markov Decision Process**
    - Powerful (generic) paradigm to define sequential tasks.
    - Can deal with multiple goals (and trading-off between them through utility/value).
    - Can deal with stochastic dynamics.

- **Bellman equation**
    - Recursive relation between state/state-action values.
    - Fundamental principle below many MDP algorithms.

- **Dynamic Programming**

Summary

- **Markov Decision Process**
    - Powerful (generic) paradigm to define sequential tasks.
    - Can deal with multiple goals (and trading-off between them through utility/value).
    - Can deal with stochastic dynamics.

- **Bellman equation**
    - Recursive relation between state/state-action values.
    - Fundamental principle below many MDP algorithms.

- **Dynamic Programming**
    - Group of algorithms to solve for the optimal value/policy in a MDP.
    - Fundamental ideas for most other MDP algorithms.

Summary

- MDP formulation
- Cumulative reward & value
- Bellman equation
- Dynamic programming

<u>Key principle below:</u>

- Search & planning
- Reinforcement learning (not in this course)

Summary

- MDP formulation
- Cumulative reward & value
- Bellman equation
- Dynamic programming

Key principle below:

- Search & planning
- Reinforcement learning (not in this course)

A lot of current AI research is into MDP problem formulations

Summary

- MDP formulation
- Cumulative reward & value
- Bellman equation
- Dynamic programming

Key principle below:

- Search & planning
- Reinforcement learning (not in this course)

A lot of current AI research is into MDP problem formulations

e.g., AlphaGo Zero:

Solution = search (MCTS) + learning

Line between symbolic and subsymbolic AI can be thin!

Assignment

1. Study the lecture notes
   a. Summarizes the material of this lecture
   b. Gives extra explanation and examples

Assignment

1. Study the lecture notes
   a. Summarizes the material of this lecture
   b. Gives extra explanation and examples

2. Make MDP assignment
   a. Coding exercises: implement value iteration and Q-value iteration
   b. Reflection exercises: answer in pdf.

Good luck!