Reinforcement Learning

Solving MDPs from samples

Cathy Wu

1.041/1.200 Transportation: Foundations and Methods

Readings

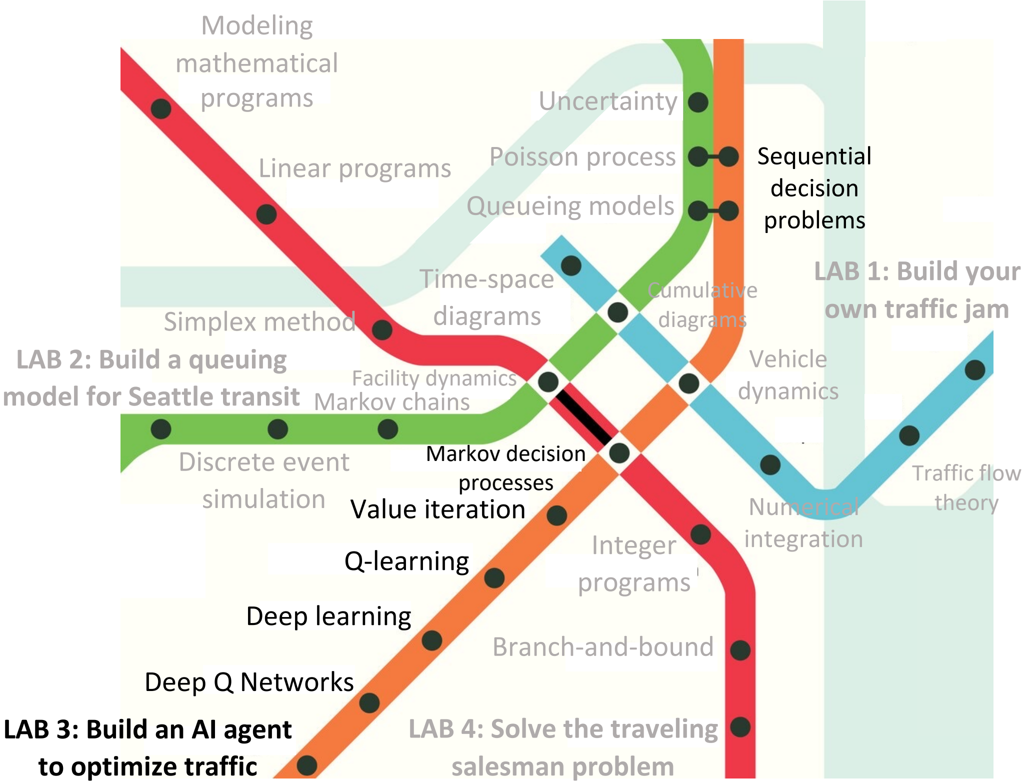
1. Miller, Tim. **Introduction to reinforcement learning**. 2024.
    - Value iteration [URL]
    - Temporal difference reinforcement learning [URL]
    - Reward shaping [URL]

Unit 3: Machine learning for traffic control

Unit 3

Optimizing

Multi-stage



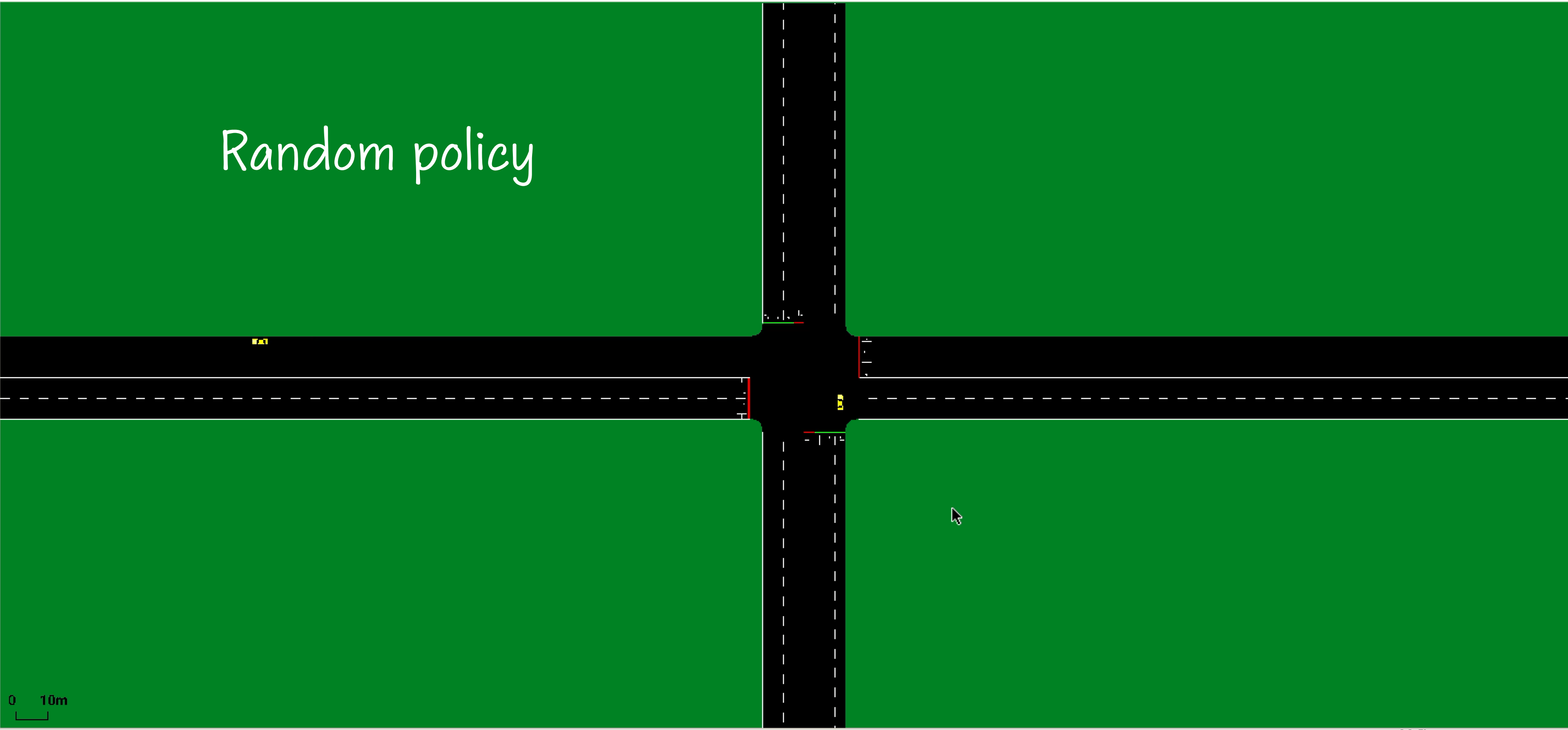
Outline

1. Dynamic programming for traffic control
2. Value iteration algorithm
3. Grid world parking problem
4. Q-value iteration algorithm
5. Q-learning algorithm
6. Reward shaping

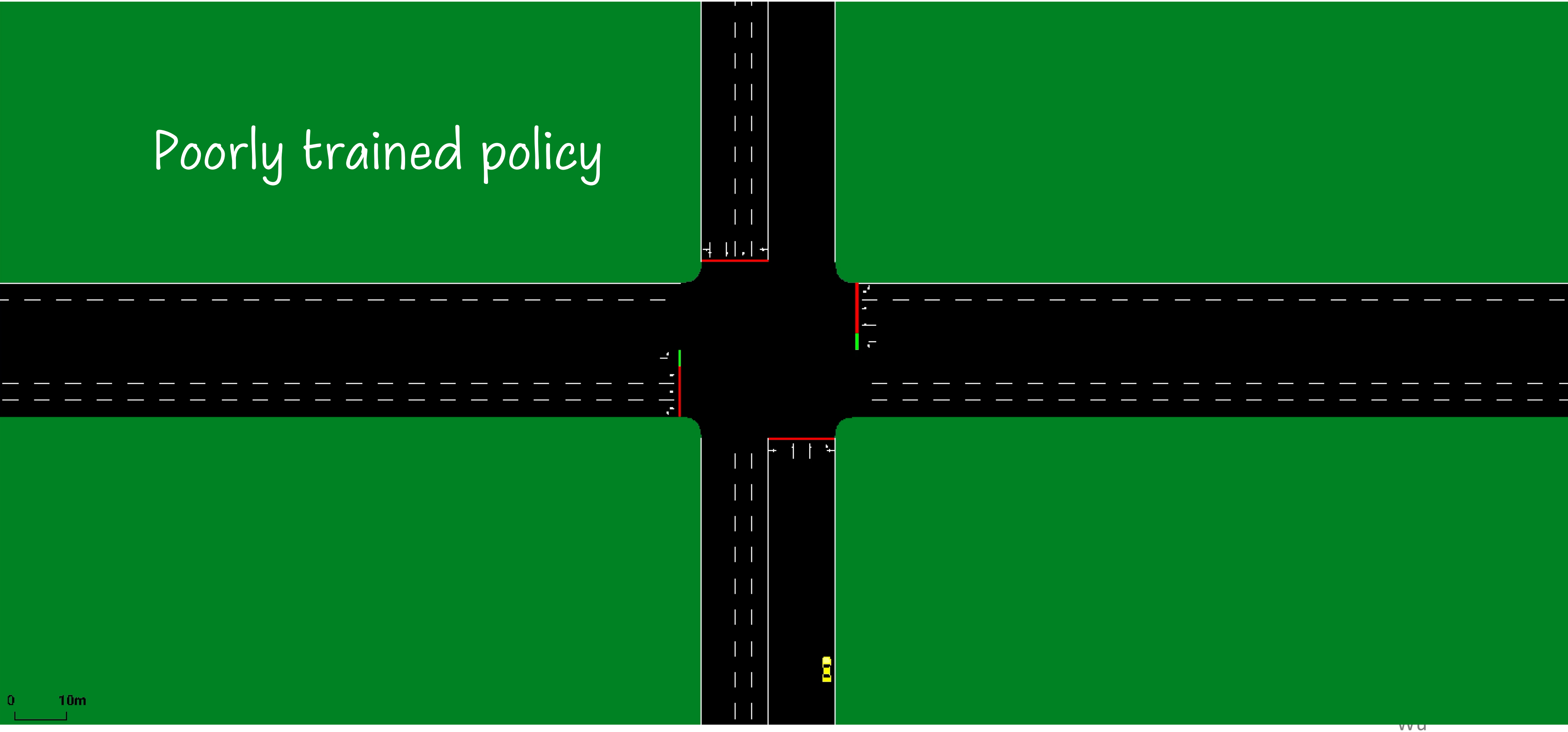
Outline

1. **Dynamic programming for traffic control**
   a. Challenges
   b. A value function for infinite horizon problems
2. Value iteration algorithm
3. Grid world parking problem
4. Q-value iteration algorithm
5. Q-learning algorithm
6. Reward shaping

CL3: Build an AI agent to optimize traffic



CL3: Build an AI agent to optimize traffic



DP for traffic signal control: challenges

(Today)
Updates all states (even impossible/unlikely)

(Lectures 17-18)
Large state space (e.g., $|S| = 2^{80}$)

(Today)
Long horizon (e.g., $T = 5400$)

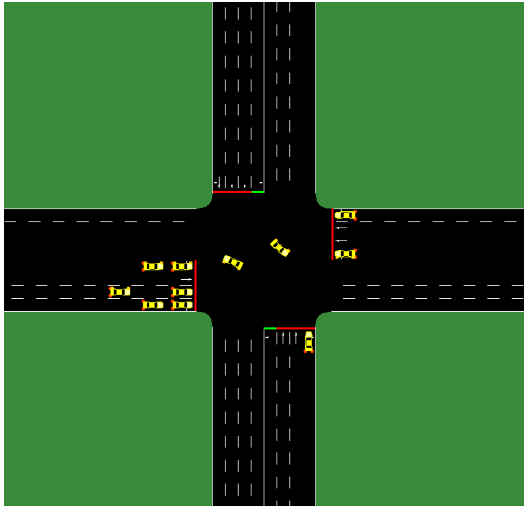
(Today)
Reward sparse (often zero)

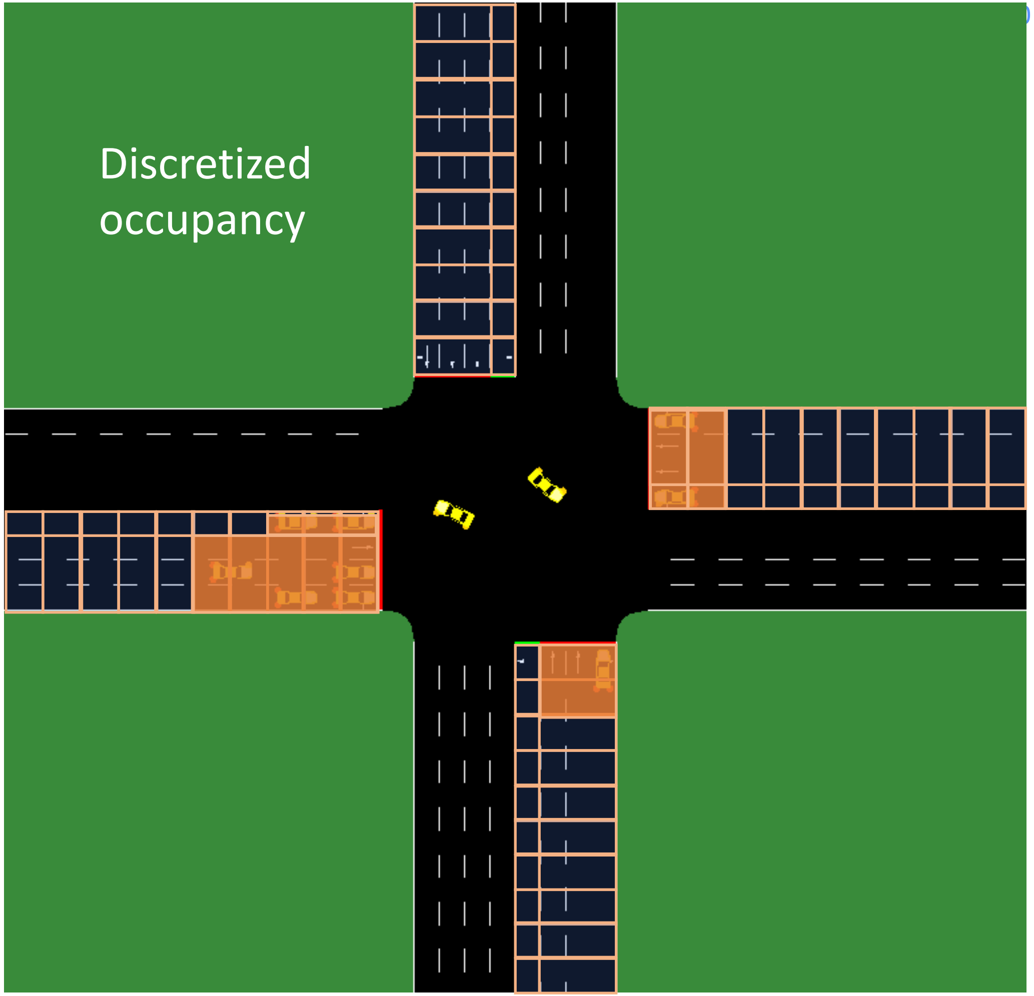
(Today)
Check all next states to select next action

$V_T(s_T) = r_T(s_T)$
for $t = T - 1, \dots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**
    $V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} r_t(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P(\cdot | s_t, a_t)}[V_{t+1}(s_{t+1})]$
end for

- Dynamic programming: O(|S|²|A|T) = $2^{80 \times 2} \times 4 \times 5400$
  - Not so efficient ☹
- Parts that are (surprisingly) OK
  - DP recursion
  - Action space usually small

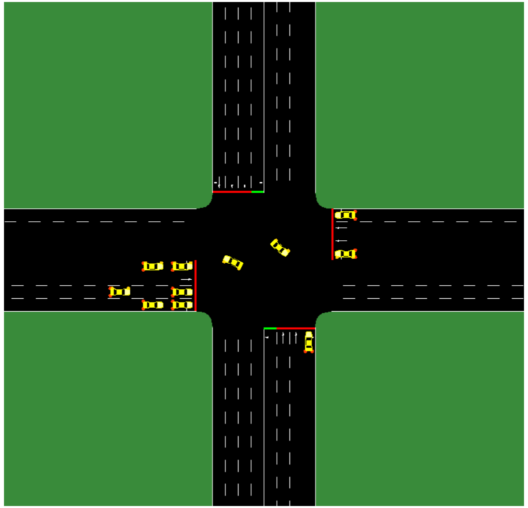
State representation





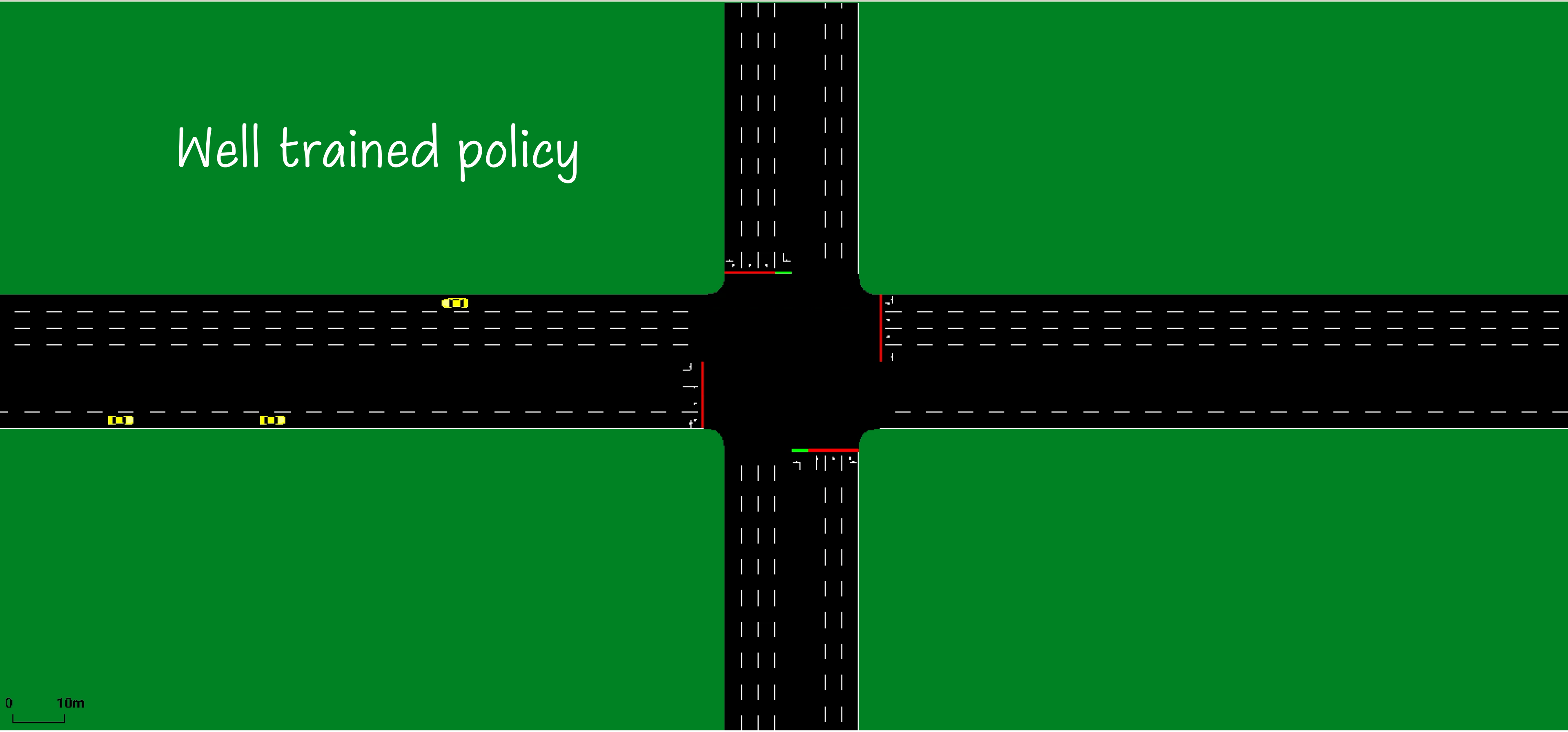
Wu

Reward function



Total wait time among all vehicles
(over 90 minutes)

CL3: Build an AI agent to optimize traffic



DP for traffic signal control: challenges

(Today)
Updates all states (even impossible/unlikely)

(Lectures 17-18)
Large state space (e.g., $|S| = 2^{80}$)

(Today)
Long horizon (e.g., $T = 5400$)

(Today)
Reward sparse (often zero)

(Today)
Check all next states to select next action

$V_T(s_T) = r_T(s_T)$
for $t = T - 1, \dots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**
    $V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} r_t(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)}[V_{t+1}(s_{t+1})]$
end for

- Dynamic programming: O(|S|²|A|T) = $2^{80\times 2} \times 4 \times 5400$
  - Not so efficient ☹
- Parts that are (surprisingly) OK
  - DP recursion
  - Action space usually small

Recall (finite horizon): The value function

Given a policy π (deterministic to simplify notation)

- Finite time horizon T: deadline at time T, the agent focuses on the sum of the rewards up to T.

$$V^{\pi}(t,s) = \mathbb{E}\left[\sum_{\tau=t}^{T-1} r(s_\tau, \pi(s_\tau)) + R(s_T) | s_t = s; \pi\right]$$

where R is a value function for the final state.

- Used when: there is an intrinsic deadline to meet.
- Shorthand: $V_t^{\pi}(s)$ or simply V_t^{π} (think: vector of size $|S|$)

The infinite horizon value function

- Given a policy $\pi = (d_1, d_2, \dots)$ (deterministic to simplify notation)
  - Infinite time horizon with discount: the problem never terminates but rewards which are closer in time receive a higher importance.

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, \pi_{t(h_t)}\right) | s_0 = s; \pi\right]$$

  with discount factor $0 \leq \gamma < 1$:

    - Small = short-term rewards, big = long-term rewards
    - For any $\gamma \in [0, 1)$ the series always converges (for bounded rewards)

  - Used when: there is uncertainty about the deadline, to model an intrinsic definition of discount, or to model a long deadline.

Optimization Problem

- Same as before, but optimizing the infinite horizon value function
- Our goal: achieve the best value
  - Max value-to-go (min cost-to-go)

Definition (Optimal policy and optimal value function)

The solution to an MDP is an optimal policy π^* satisfying

$$\pi^* \in \arg\max_{\pi \in \Pi} V_0^{\pi}$$

where Π is some policy set of interest.

The corresponding value function is the optimal value function

$$V^* = V_0^{\pi^*}$$

Outline

1. Dynamic programming for traffic control
2. **Value iteration algorithm**
   a. Bellman operator
3. Grid world parking problem
4. Q-value iteration algorithm
5. Q-learning algorithm
6. Reward shaping

Value iteration algorithm

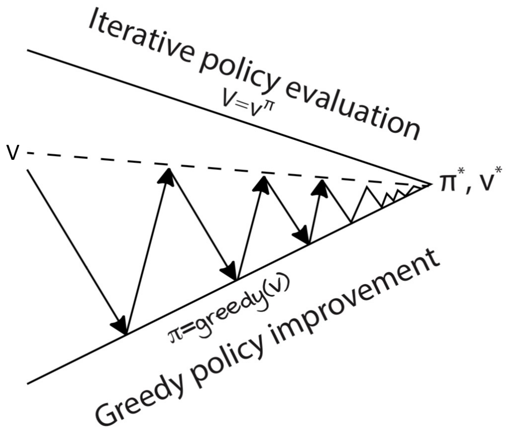
1. Let $V_0(s)$ be any function $V_0: S \to \mathbb{R}$. [Note: not stage 0, but iteration 0.]
2. Apply the principle of optimality so that given V_i at iteration i, we compute
$$V_{i+1}(s) = \mathcal{T}V_i(s) \coloneqq \max_{a \in A} r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} [V_i(s')] \quad \text{for all } s$$
3. Terminate when V_i stops improving, e.g. when $\max_s |V_{i+1}(s) - V_i(s)|$ is small.
4. Return the greedy policy: $\pi_K(s) = \arg\max_{a \in A} r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V_K(s')$

Definition (Optimal Bellman operator)

For any $W \in \mathbb{R}^{|S|}$, the optimal Bellman operator is defined as
$$\mathcal{T}W(s) \coloneqq \max_{a \in A} r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} W(s') \quad \text{for all } s$$

☞ Then we can write the algorithm step 2 concisely:
$$V_{i+1}(s) = \mathcal{T}V_i(s) \quad \text{for all } s$$

☞ A key result: $V_i \to V^*$, as $i \to \infty$.



Adapted from Morales, Grokking Deep Reinforcement Learning, 2020.

The Optimal Bellman Equation

Bellman's Principle of Optimality (Bellman (1957)):

"An optimal policy has the property that, whatever the initial state and the initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

The Optimal Bellman Equation

Theorem (Optimal Bellman Equation)

The optimal value function V^* (i.e. $V^* = \max_\pi V^\pi$) is the solution to the optimal Bellman equation:

$$V^*(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum_{s'} p\,(s'|s,a)\, V^*(s') \right]$$

And any optimal policy is such that:

$$\pi^*(a|s) \geq 0 \Leftrightarrow a \in \arg\max_{a' \in A} \left[r(s,a') + \gamma \sum_{s'} p\,(s'|s,a)\, V^*(s') \right]$$

Or, for short: $V^* = \mathcal{T} V^*$

☞ There is always an optimal deterministic policy (see: Puterman, 2005, Ch. 7)

Value Iteration: the Complexity

Time complexity

- Each iteration takes on the order of $S^2 A$ operations.

$$V_{k+1}(s) = \mathcal{T} V_k(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_k(s') \right]$$

- The computation of the greedy policy takes on the order of $S^2 A$ operations.

$$\pi_K(s) \in \arg\max_{a \in A} \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_K(s') \right]$$

- Total time complexity on the order of $K S^2 A$.

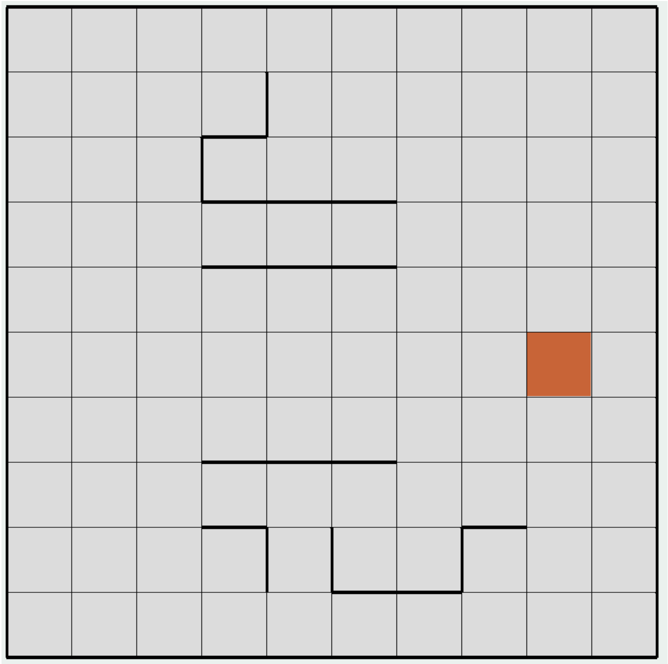
Space complexity

- Storing the MDP: dynamics on the order of $S^2 A$ and reward on the order of SA.
- Storing the value function and the optimal policy on the order of S.

Outline

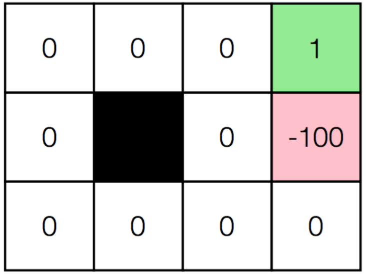
1. Dynamic programming for traffic control
2. Value iteration algorithm
3. **Grid world parking problem**
   a. Value iteration demo
4. Q-value iteration algorithm
5. Q-learning algorithm
6. Reward shaping

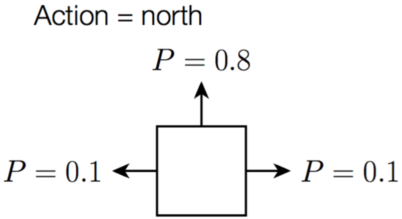
The Grid-World Problem



Example: Winter parking (with ice and potholes)

- Simple grid world with a *goal state* (green, desired parking spot) with reward (+1), a *"bad state"* (red, pothole) with reward (-100), and all other states neural (+0).
- *Omnidirectional vehicle (agent)* can head in any direction. Actions move in the desired direction with probably 0.8, in one of the perpendicular directions with.
- Taking an action that would bump into a wall leaves agent where it is.





[Source: adapted from Kolter, 2016]

Example: value iteration

Running value iteration with $\gamma = 0.9$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

Recall value iteration algorithm:

$$V_{i+1}(s) = \max_{a \in A} r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V_i(s') \quad \text{for all } s$$

Let's arbitrarily initialize V_0 as the reward function, since it can be any function.

Example update (red state):

$$V_1(\text{red}) = -100 + \gamma \max\{ \begin{array}{ll} 0.8V_0(\text{green}) + 0.1V_0(\text{red}) + 0, & \text{[up]} \\ 0 + 0.1V_0(\text{red}) + 0, & \text{[down]} \\ 0 + 0.1V_0(\text{green}) + 0, & \text{[left]} \\ 0.8V_0(\text{red}) + 0.1V_0(\text{green}) + 1 \ \} & \text{[right]} \end{array}$$

$= -100 + 0.9(0.1 * 1) = -99.91$ [best: go left]

Example: value iteration

Running value iteration with $\gamma = 0.9$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

Recall value iteration algorithm:

$$\mathrm{V}_{\mathrm{i+1}}(s) = \max_{a \in A} r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V_i(s') \quad \text{for all } s$$

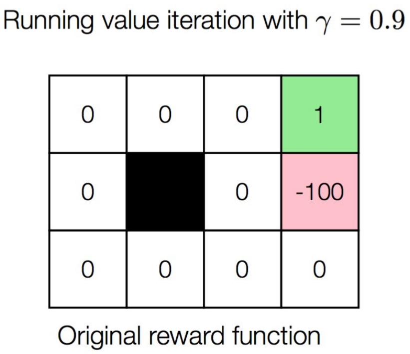
Let's arbitrarily initialize V_0 as the reward function, since it can be any function.

Example update (green state):

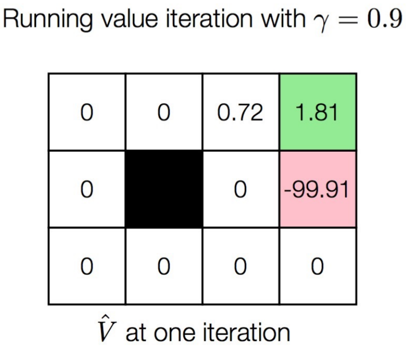
$$\begin{aligned} \mathrm{V}_1(\text{green}) = 1 \quad + \gamma \max\{ \; & 0.8V_0(\text{green}) + 0.1V_0(\text{green}), && \text{[up]} \\ & 0.8V_0(\text{red}) + 0.1V_0(\text{green}), && \text{[down]} \\ & 0 + 0.1V_0(\text{green}) + 0.1V_0(\text{red}), && \text{[left]} \\ & 0.8V_0(\text{red}) + 0.1V_0(\text{green}) + 0 \; \} && \text{[right]} \end{aligned}$$

$= 1 + 0.9(0.9 * 1) = 1.81$ [best: go up]

Example: value iteration



(a)



(b)

Recall value iteration algorithm:

$$V_{i+1}(s) = \max_{a \in A} r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V_i(s') \quad \text{for all } s$$

Let's arbitrarily initialize V_0 as the reward function, since it can be any function.

Need to also do this for all the "unnamed" states, too.

Example: value iteration

Running value iteration with $\gamma = 0.9$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

Running value iteration with $\gamma = 0.9$

0	0	0.72	1.81
0		0	-99.91
0	0	0	0

$\hat{V}$ at one iteration

(b)

Running value iteration with $\gamma = 0.9$

0.809	1.598	2.475	3.745
0.268		0.302	-99.59
0	0.034	0.122	0.004

$\hat{V}$ at five iterations

(c)

Running value iteration with $\gamma = 0.9$

2.686	3.527	4.402	5.812
2.021		1.095	-98.82
1.390	0.903	0.738	0.123

$\hat{V}$ at 10 iterations

(d)

Running value iteration with $\gamma = 0.9$

5.470	6.313	7.190	8.669
4.802		3.347	-96.67
4.161	3.654	3.222	1.526

$\hat{V}$ at 1000 iterations

(e)

Running value iteration with $\gamma = 0.9$

→	→	→	↑
↑		←	←
↑	←	←	↓

Resulting policy after 1000 iterations

(f)

Value iteration demo

Outline

1. Dynamic programming for traffic control
2. Value iteration algorithm
3. Grid world parking problem
4. **Q-value iteration algorithm**
    a. State-action values (“Q values”)
5. Q-learning algorithm
6. Reward shaping

DP for traffic signal control: challenges

(Today)
Updates all states (even impossible/unlikely)

(Lectures 17-18)
Large state space (e.g., $|S| = 2^{80}$)

(Today)
~~Long horizon (e.g., $T = 5400$)~~

(Today)
Reward sparse (often zero)

(Today)
Check all next states to select next action

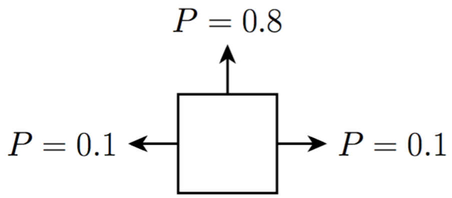
$V_T(s_T) = r_T(s_T)$
for $t = T-1, \dots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**
    $V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} r_t(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)}[V_{t+1}(s_{t+1})]$
end for

- Dynamic programming: $O(|S|^2|A|T) = 2^{80\times 2} \times 4 \times 5400$
  - Not so efficient ☹
- Parts that are (surprisingly) OK
  - DP recursion
  - Action space usually small

State-Action Value Function ("Q table")

- Example: Winter parking (with ice and potholes)

0	0	0	1
0		0	-100
0	0	0	0



Running value iteration with $\gamma = 0.9$

It is convenient to keep track of not only the long term value of a state, but also the state, jointly with the next action.

$V(s)$

5.470	6.313	7.190	8.669
4.802		3.347	-96.67
4.161	3.654	3.222	1.526

$\hat{V}$ at 1000 iterations

$Q(s,a)$

s

↑	↓	←	→
2.5	1.4	3.2	5.4
1.0	3.2	5.1	6.3
5.2	4.2	5.5	7.2
8.7	3.4	2.0	8.0
4.8	2.5	3.5	4.2
1.0	3.0	3.3	1.2
-180	-172	-99.7	-150
4.2	2.1	3.2	3.7
2.1	2.0	3.7	3.1
3.0	1.2	3.2	2.7
0.1	1.5	0.1	1.0

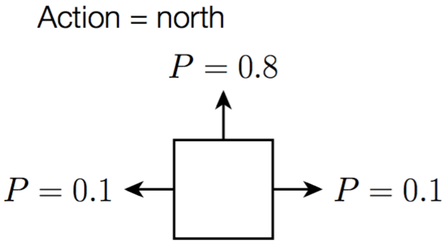
a

Wu

Convenient for selecting next action!

- Winter parking (with ice and potholes)

0	0	0	1
0		0	-100
0	0	0	0



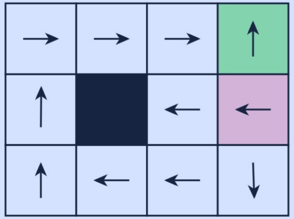
Before

Running value iteration with $\gamma = 0.9$

5.470	6.313	7.190	8.669
4.802		3.347	-96.67
4.161	3.654	3.222	1.526

$\hat{V}$ at 1000 iterations

Running value iteration with $\gamma = 0.9$



Resulting policy after 1000 iterations

$$\pi_K(s) = \arg\max_{a \in A} r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V_K(s')$$

$Q(s,a)$

s \ a	↑	↓	←	→
	2.5	1.4	3.2	5.4
	1.0	3.2	5.1	6.3
	5.2	4.2	5.5	7.2
	8.7	3.4	2.0	8.0
	4.8	2.5	3.5	4.2
	1.0	3.0	3.3	1.2
	-180	-172	-99.7	-150
	4.2	2.1	3.2	3.7
	2.1	2.0	3.7	3.1
	3.0	1.2	3.2	2.7
	0.1	1.5	0.1	1.0

State-Action Value Function

Definition (State-action Value Function)

In discounted infinite horizon problems, for any policy π, the state-action value function (or Q-function) $Q^\pi : S \times A \mapsto \mathbb{R}$ is

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a, a_t = \pi(s_t), \forall t \geq 1\right]$$

The optimal Q-function is

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

and the optimal policy can be obtained as

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- → Q-value iteration (just like value iteration, but with Q instead of V).
- Benefit: computing the greedy policy from the Q-function does not require the MDP

$$\pi_K(s) \in \arg\max_{a \in A} Q_K(s,a)$$

- Compare:

$$\pi_K(s) = \arg\max_{a \in A} r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V_K(s')$$

Q-value Iteration

Q-iteration:

1. Let Q_0 be any Q-function

2. At each iteration $k = 1, 2, \dots, K$
   - Compute $Q_{k+1} = \mathcal{T} Q_k$

3. Return the greedy policy

$$\pi_K(s) \in \arg\max_{a \in A} Q_K(s, a)$$

Remark

- Still requires model to compute $Q_{k+1} = \mathcal{T} Q_k$

$Q_0(s, a)$

↑	↓	←	→
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

a

Wu

Outline

1. Dynamic programming for traffic control
2. Value iteration algorithm
3. Grid world parking problem
4. Q-value iteration algorithm
5. **Q-learning algorithm**
   a. Exploration vs exploitation
6. Reward shaping

DP for traffic signal control: challenges

(Today)
Updates all states (even impossible/unlikely)

(Lectures 17-18)
Large state space (e.g., $|S| = 2^{80}$)

(Today)
~~Long horizon (e.g., $T = 5400$)~~

(Today)
Reward sparse (often zero)

(Today)
~~Check all next states to select next action~~

$V_T(s_T) = r_T(s_T)$
for $t = T - 1, \dots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**
    $V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} r_t(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)}[V_{t+1}(s_{t+1})]$
end for

- Dynamic programming: O(|S|²|A|T) = $2^{80\times 2} \times 4 \times 5400$
  - Not so efficient ☹
- Parts that are (surprisingly) OK
  - DP recursion
  - Action space usually small

Q-learning

- Key idea: **Update one state at a time** (a little bit)
- Q-value iteration update

$$Q_{i+1}(s,a) = \mathcal{T}Q_i(s) \coloneqq \max_{a' \in A} r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[Q_i(s',a')] \quad \forall s,a$$

- Q-learning update

$$Q_{k+1}(s,a) = (1-\eta_k)Q_k(s,a) + \eta_k \left(r + \gamma \max_{a' \in A} Q_k(s',a')\right)$$

  - with learning rates $\eta_k \ll 1$

- Q-learning is called model-free because it does not require access to P and r functions. Only requires samples (data) from P and r.
  - Compare: value iteration, Q-value iteration are model-based.
- Dynamic programming is model-based
- Reinforcement learning is model-free

Q Learning Algorithm

1. Let Q_0 be any Q-function, s be an initial state,
2. At each iteration $k = 0, 1, 2, \dots, K$
   - Use Q_k to select an action a
   - Observe next state s' and reward r
   - Update Q function: $Q_{k+1}(s,a) = (1-\eta_k)Q_k(s,a) + \eta_k\left(r + \gamma \max_{a'} Q_k(s',a')\right)$
   - $s \leftarrow s'$
3. Return the greedy policy

learning rates $\eta_k \in [0,1]$.

$$= Q_k(s,a) + \eta_k\left(\underbrace{r + \gamma \max_{a'} Q_k(s',a')}_{\text{TD / bootstrap target}} - \underbrace{Q_k(s,a)}_{\text{Current guess of value}}\right)$$

Temporal difference (TD) error δ_k

$$\pi_K(s) = \arg\max_{a \in A} Q_K(s,a)$$

Additional assumption needed for convergence:

- **Coverage**: All the state-action pairs are visited infinitely often.
- **Learning rate**: If for any $n, \eta_n \geq 0$ and are such that:

$$\sum_{n\geq 0} \eta_n = \infty, \qquad \sum_{n\geq 0} \eta_n^2 < \infty$$

Exploration vs exploitation

- How to ensure that learning agent visits potentially good states?
- From the Q-learning algorithm: Use Q_k to select an action a
- Complete exploitation:

$$\arg\max_{a \in A} Q_k(s, a)$$

- Complex exploration:

$$\pi(a|s) = \frac{1}{|\mathcal{A}|}$$

- ϵ-greedy: a simple strategy to balance the two
  - With probability ϵ, explore. Else, exploit

Outline

1. Dynamic programming for traffic control
2. Value iteration algorithm
3. Grid world parking problem
4. Q-value iteration algorithm
5. Q-learning algorithm
6. **Reward shaping**
    a. Potential-based reward shaping
    b. Reward hacking
    c. Reward tuning
    d. Reward shaping demo

DP for traffic signal control: challenges

(Today)
~~Updates all states (even impossible/unlikely)~~

(Today)
~~Long horizon (e.g., $T = 5400$)~~

(Lectures 17-18)
Large state space (e.g., $|S| = 2^{80}$)

(Today)
Reward sparse (often zero)

(Today)
~~Check all next states to select next action~~

$V_T(s_T) = r_T(s_T)$
for $t = T - 1, \dots, 0$ **do**
  for $s_t \in \mathcal{S}_t$ **do**
    $V_t(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} r_t(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)}[V_{t+1}(s_{t+1})]$
end for

- Dynamic programming: O(|S|²|A|T) = $2^{80 \times 2} \times 4 \times 5400$
  - Not so efficient ☹
- Parts that are (surprisingly) OK
  - DP recursion
  - Action space usually small

Challenge: Reward sparsity

- Rewards are sparse when few state/action pairs have non-zero rewards.

Reward shaping

- **Reward shaping** is the use of small intermediate 'fake' rewards given to the learning agent that help it converge more quickly.
- Can we speed up learning and/or improve our final solution by nudging our reinforcement learning agent towards behavior we think is good?
- Yes!
- By incorporating **domain knowledge** - stuff about the domain that the human modeller knows about while constructing the model to be solved.

Shaped reward

- In TD learning methods, we update a Q-function when a reward is received. E.g, for 1-step Q-learning:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

- The approach to reward shaping is not to modify the reward function or the received reward r, but to just give some additional reward for some actions:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + F(s,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Additional reward

- Shaped reward: $r + F(s,s')$
- Reward tuning: even more generally, **tuned reward**: $r + G(s,a,s')$

Additional reward

Potential-based Reward Shaping

Potential-based reward shaping is a particular type of reward shaping with nice theoretical guarantees. In potential-based reward shaping, F is of the form:

$$F(s, s') = \gamma\Phi(s') - \Phi(s)$$

We call Φ the **potential function** and $\Phi(s)$ is the **potential** of state s.

So, instead of defining $F : S \times S \to \mathbb{R}$, we define $\Phi : S \to \mathbb{R}$, which is some heuristic measure of the value of each state $s \in S$.

Theoretical guarantee: this will still converge to the optimal policy under the assumption that all state-action pairs are sampled infinitely often.

Potential-based Reward Shaping

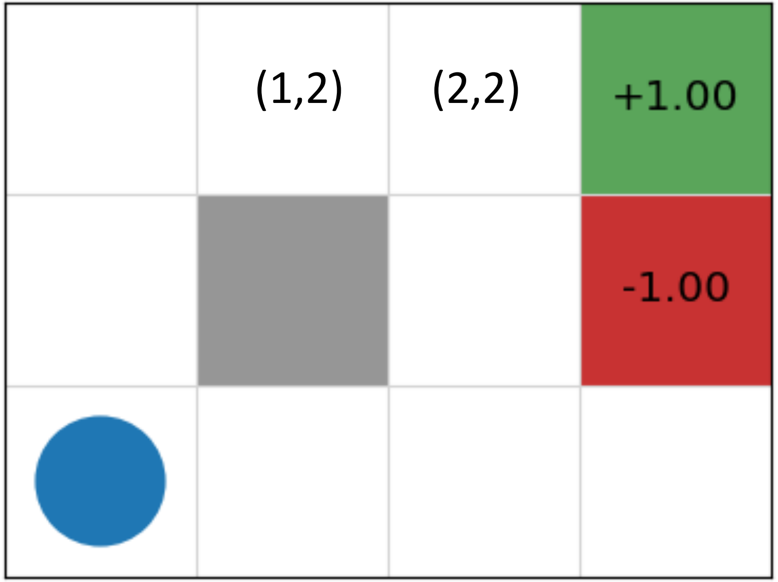
This is quite straightforward to show as follows. Consider an episode with shaped reward G^Φ :

$$
\begin{aligned}
G^\Phi &= \sum_{i=0}^{\infty} \gamma^i (r_i + F(s_i, s_{i+1})) \\
&= \sum_{i=0}^{\infty} \gamma^i (r_i + \gamma\Phi(s_{i+1}) - \Phi(s_i)) \\
&= \sum_{i=0}^{\infty} \gamma^i r_i + \sum_{i=0}^{\infty} \gamma^{i+1}\Phi(s_{i+1}) - \sum_{i=0}^{\infty} \gamma^i \Phi(s_i) \\
&= G + \sum_{i=0}^{\infty} \gamma^i \Phi(s_i) - \Phi(s_0) - \sum_{i=0}^{\infty} \gamma^i \Phi(s_i) \\
&= G - \Phi(s_0)
\end{aligned}
$$

where G refers to the shaped reward for the episode, and s_0 is the starting state of the episode. What this says is that the shaped reward G^Φ is just the unshaped reward G minus the potential of the initial state s_0. However, because F does not depend on the actions and G^Φ does not depend on shaped rewards beyond the initial state, the **shaped Q function**, which we refer to as Q^Φ, can be defined as just $Q^\Phi(s, a) = Q(s, a) + \Phi(s)$. Given this, any optimal policy extracted from Q^Φ will be equivalent to any optimal policy extracted from Q.

However! While it provides guarantees about the end result, potential-based reward shaping may either increase or decrease the time taken to learn. A well-designed potential function decrease the time to convergence.

Example -- Potential Reward Shaping for GridWorld



Example -- Potential Reward Shaping for GridWorld

For Grid World, we use the Manhattan distance to define the potential function, normalised by the size of the grid:

$$\Phi(s) = 1 - \frac{|x(g) - x(s)| + |y(g) - y(s)|}{width + height - 2}$$

in which $x(s)$ and $y(s)$ return the x and y coordinates of the agent respectively, g is the goal state. and $width$ and $height$ are the width and height of the grid respectively. Note that the coordinates are indexed from 0, so we subtract 2 from the denominator.

Even on the very first iteration, a greedy policy such as ϵ-greedy, will feedback those states closer to the +1 reward. From state (1,2) with $\gamma = 0.9$ if we go Right, we get:

$$\begin{aligned} F((1,2),(2,2)) &= \gamma\Phi(2,2) - \Phi(1,2) \\ &= 0.9 \cdot (1 - \tfrac{1}{5}) - (1 - \tfrac{2}{5}) \\ &= 0.12 \end{aligned}$$

Example -- Potential Reward Shaping for GridWorld

We can compare the Q-values for these states for the four different possible moves that could have been taken from (1,2), using and $\alpha = 0.1$ and $\gamma = 0.9$:

Action	r	$F(s, s')$	$\gamma \max_{a'} Q(s', a')$	New $Q(s, a)$
Up	0	$0.9(1 - \frac{2}{5}) - (1 - \frac{2}{5}) = -0.06$	0	-0.006
Down	0	$0.9(1 - \frac{2}{5}) - (1 - \frac{2}{5}) = -0.06$	0	-0.006
Right	0	$0.9(1 - \frac{1}{5}) - (1 - \frac{2}{5}) = 0.12$	0	0.012
Left	0	$0.9(1 - \frac{3}{5}) - (1 - \frac{2}{5}) = -0.24$	0	-0.024

Thus, we can see that our potential reward function rewards actions that go towards the goal and penalises actions that go away from the goal. Recall that state (1,2) is in the top row, so action Up just leaves us in state (1,2) and Down similarly because we cannot go through the walls.

But! It will not always work. Compare states (0,0) and (0,1). Our potential function will reward (0,1) because it is closer to the goal, but we know from from our value iteration example that (0,0) is a higher value state than (0,1). This is because our reward function does not consider the negative reward.

In practice, it is non-trivial to derive a perfect reward function -- it is the same problem as deriving the perfect search heuristic. If we could do this, we would not need to even use reinforcement learning -- we could just do a greedy search over the reward function.

Reward hacking

- **Reward hacking** is the phenomenon where optimizing an imperfect proxy reward function leads to poor performance according to the true reward function.

J. Skalse, N. H. R. Howe, D. Krasheninnikov, and D. Krueger, “Defining and Characterizing Reward Hacking.” arXiv, Sep. 26, 2022. doi: 10.48550/arXiv.2209.13085.

Reward shaping demo [URL]

Summary & Takeaways

- **Traffic signal control** is a harder sequential decision problem than those considered thus far, due to its long horizon, large state space, and sparse rewards.
- Fortunately, the ideas from **dynamic programming** extend to the **discounted infinite horizon setting** (value iteration), **state-action value functions** (Q-value iteration), and **learning directly from samples** (Q-learning).
  - These three algorithms are all guaranteed to converge to the optimal solution asymptotically (i.e., if run for long enough)
- **Reward shaping** takes in some domain knowledge that "nudges" the learning algorithm towards more positive actions.
  - A weakness of model-free methods is that they spend a lot of time exploring at the start of the learning. It is not until they find some rewards that the learning begins. This is particularly problematic when rewards are **sparse**.
  - **Potential-based reward shaping** guarantees that the policy will converge to the same policy without reward shaping.

References

1. Miller, Tim. **Introduction to reinforcement learning**. 2024.
   - Value iteration [URL]
   - Temporal difference reinforcement learning [URL]
   - Reward shaping [URL]
2. Morales, Miguel. **Grokking deep reinforcement learning**. 2020. Chapter 3: Balancing immediate and long-term goals.
3. Dimitri P. Bertsekas. Dynamic Programming and Optimal Control. Volume 2. 4th Edition. (2012). Chapters 1-2: Discounted Problems.
4. R. E. Bellman. Dynamic Programming. Princeton University Press, Princeton, N.J., 1957.
5. Many slides adapted from Alessandro Lazaric and Matteo Pirotta.