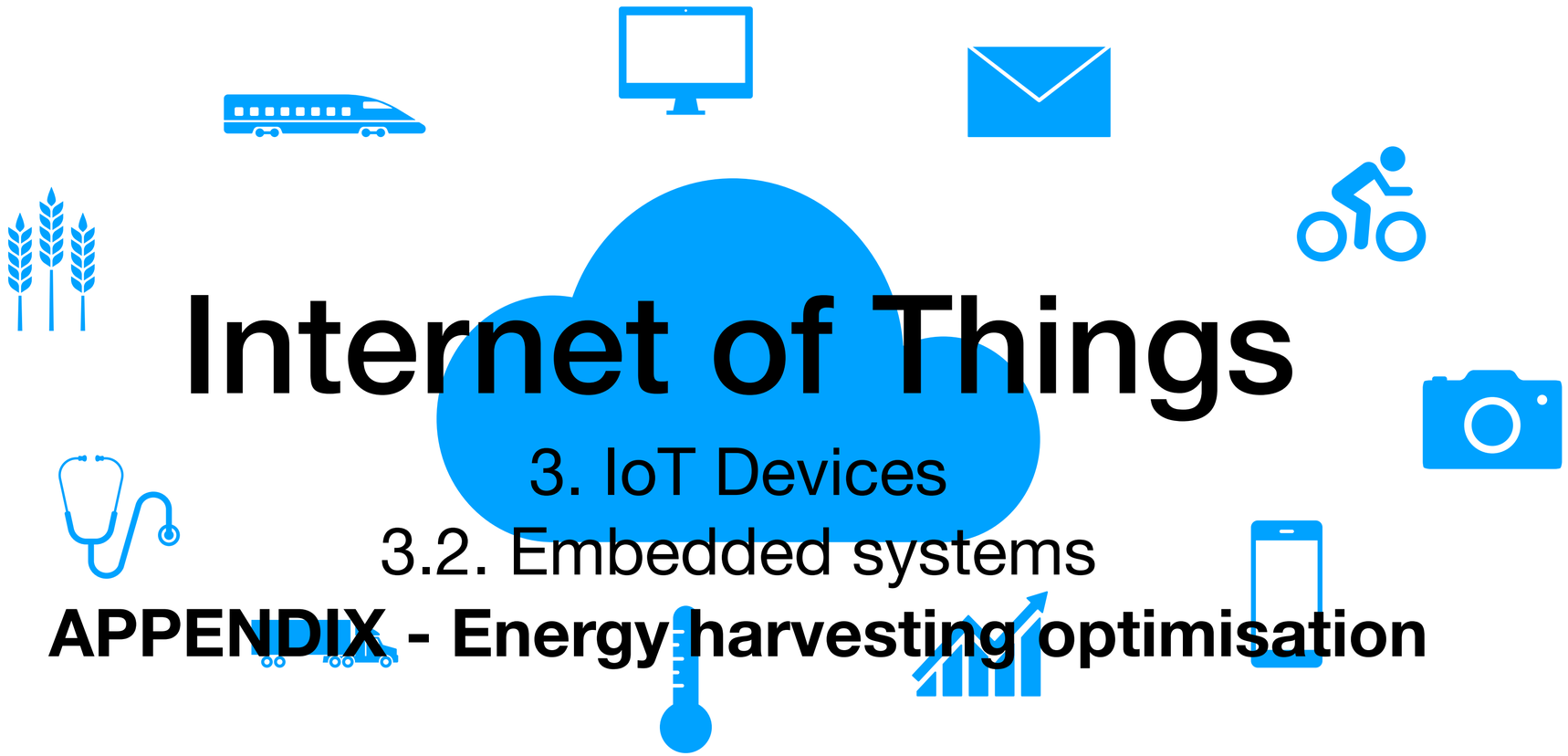


Internet of Things

3. IoT Devices

3.2. Embedded systems

APPENDIX - Energy harvesting optimisation

M.Sc. Computer Science 2024-2025

Viviana Arrigoni

3.3.5. In-depth analysis: Energy harvesting - Markov Decision Processes (MDP)

Markov Chains and Markov Decision Processes

- **On the blackboard (notes released).**
- A lot of online material, e.g.,:
  - Markov Chains: https://www.stat.auckland.ac.nz/~fewster/325/notes/ch8.pdf, https://web.stanford.edu/class/cs265/Lectures/Lecture14/l14.pdf
  - Markov Decision Processes: https://www.davidsilver.uk/wp-content/uploads/2020/03/MDP.pdf

A Learning Theoretic Approach to Energy Harvesting Communication System Optimization

Pol Blasco, Deniz Gündüz, and Mischa Dohler

IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, VOL. 12, NO. 4, APRIL 2013

System Model (1)

- Wireless transmitter equipped with an Energy Harvesting (EH) device and a rechargeable battery with limited storage capacity.
- Communication system operates in a time-slotted fashion over time steps (TSs) of equal duration.
- Channel state remains constant during each TS and can change from one TS to another.
- Data and energy arrive in packets in each TS.

System Model (2)

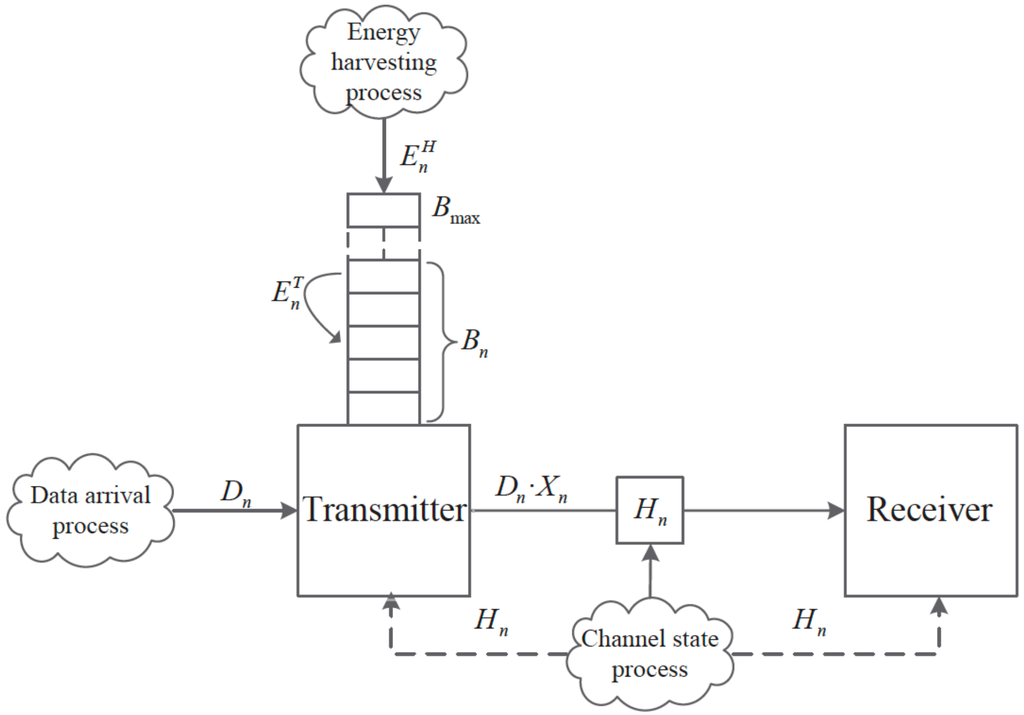
- The **size of the data and energy packets arriving at the beginning of each TS are modelled as a discrete Markov chain**.
- $D_n \in \mathscr{D} = \{d_1, \ldots, d_{N_{\mathscr{D}}}\}$ is the size of the data packet arriving at TS n.
- $p_d(d_j, d_k)$: transition probability, i.e., probability that the next data packet has size d_k given that the current packet has size d_j.
- $E_n^H \in \mathscr{E} = \{e_1, \ldots, e_{N_{\mathscr{E}}}\}$ is the amount of energy harvested during TS n.
- $p_e(e_j, e_k)$: state transition probability function.
- E_n^H: energy stored in the battery that can be used for data transmission from TS $n+1$.
- The battery is limited and has size of B_{max} energy units. The energy harvested when the battery is full goes lost. The amount of energy stored in the battery in TS n is B_n, $0 \leq B_n \leq B_{max}$.

System Model (3)

- $H_n \in \mathscr{H} = \{h_1, \ldots, h_{N_{\mathscr{H}}}\}$ is the state of the channel during TS n.
- The channel state follows a Markov model, $p_h(h_j, h_k)$ is the channel state transition probability.
  - The channel state is a measure of the current condition of a wireless communication link, and affects signal strength, interference, quality of signal propagation. It affects the quantity of energy necessary to propagate the signal.
- For each channel state H_n and packet size D_n, the transmitter knows the amount of minimum energy $E_n^T = f_e(D_n, H_N) : \mathscr{D} \times \mathscr{H} \rightarrow \mathscr{E}_u$ to transmit the data packet through the channel.

System Model (4)

- DECISIONS: at the beginning of each TS, the transmitter makes a **binary** decision: to transmit or to drop the packet.
  - Transmission can still fail with probability γ.
- The transmitter must guarantee that the energy spent in TS n is not greater than the energy available in the battery B_n.



Expected total transmitted data maximisation problem (ETD-problem)

- The transmitter aims at maximising the **total transmitted data** over the long term.
- Decision variable: $X_n \in \{0,1\}, X_n = \begin{cases} 1 & \text{incoming packet is transmitted} \\ 0 & \text{otherwise} \end{cases}$

$$\max_{\{X_i\}_{i=0}^{\infty}} \lim_{N\to\infty} \mathbb{E}\left[\sum_{n=0}^{N} \gamma^n X_n D_n\right] \qquad \text{ETDP}$$

subject to: $X_n E_n^T \leq B_n$

$$B_{n+1} = \min\{B_n - X_n E_n^T + E_n^H, B_{max}\}$$

$0 \leq 1-\gamma \leq 1$ iid probabilities that the transmission terminates its operations in each TS

MDP for ETD

- State at TS n: $S_n = (E_n^H, D_n, H_n, B_n)$, since all the components of the state have a finite number of values, $\mathcal{S} = \{s_1, \ldots, s_{N_\mathcal{S}}\}$ is finite.
- Actions: $\mathcal{A} = \{0,1\}$. 0 = packet is dropped, 1 = packet is transmitted.
- Transition probabilities: $p_{X_n}(s_j, s_k) = \mathbb{P}[S_{n+1} = s_k \,|\, S_n = s_j, A_n = X_n]$.
- Immediate reward yielded by action $X_n \in \mathcal{A}$ when the state change from S_n to S_{n+1} is $R_{X_n}(S_n, S_{n+1}) = X_n D_n$.
- Objective: find the policy $\pi : \mathcal{S} \to \mathcal{A}$, $\pi(s) = a$, that maximises the expected discounted sum rewards (i.e., the expected total transmitted data), that is equivalent to take the best decision at each TS, X_n.
  - In fact, the objective of ETD, $\max\limits_{\{X_i\}_{i=0}^{\infty}} \lim\limits_{N\to\infty} \mathbb{E}\left[\sum\limits_{n=0}^{N} \gamma^n X_n D_n\right]$, is the maximisation of the discounted gain/return in a MDP.

Solving ETDP

- Three approaches implemented in the paper, **depending on the assumptions on the knowledge of the model**:

1. **Online optimisation**:
   - If the transmitter has prior information on the transition probabilities $p_{X_n}(s_j, s_k)$ and rewards $R_{X_n}(S_n, S_{n+1})$.
   - solve with **Dynamic programming.**

2. **Exploration & Exploitation**:
   - If the transmitter has no prior information on transition probabilities and rewards.
   - solve with **Reinforcement Learning.**

3. **Offline optimisation**:
   - Assume that all future EH states, packet sized and channel states are known over a finite horizon.
   - The problem is still NP-hard, solved with Branch-and-Bound algorithm.

Why Dynamic Programming?

- A MDP is not a “problem” that we have to “solve”. Given a MDP, the goal is to find an optimal policy. To do that, we need to find the policy that maximises the Bellman equation:

$$V^*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in S} P_{s,s'}^a V^*(s')$$

$$Q^*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} P_{s,s'}^a \max_{a'} Q^*(s',a')$$

- These equations have two properties:
  - **optimal substructure** (an optimal solution can be built from optimal solutions of the sub-problems)
  - **overlapping sub-problems** (the problem can be divided into smaller sub-problems that can be revisited over and over again)
- which are the criteria that a problem must have to be solved with dynamic programming.
- Two main DP algorithms for MDPs: **Value Iteration** and **Policy Iteration**.

Value Iteration

IDEA: iteratively compute the state value of each state, and output the policy that maximises such values.

Algorithm 1: Value Iteration

Result: Find $V^{\pi^*}(s)$ and $\pi^*(s)$, $\forall s$

1 $V(s) \leftarrow 0, \ \forall s \in S;$ *Initialise as 0 the state value of each state*

2 $\Delta \leftarrow \infty;$

3 **while** $\Delta \geq \Delta_0$ **do** Δ_0 *is the a small scalar representing tolerance*

4 $\quad \Delta \leftarrow 0;$

5 $\quad$ **for** *each* $s \in S$ **do**

6 $\quad\quad temp \leftarrow V(s);$

7 $\quad\quad V(s) \leftarrow \max_a \sum_{s'} P^a_{s,s'} [R(s,a,s') + \gamma V(s')];$

8 $\quad\quad \Delta \leftarrow \max_a(\Delta, |temp - V(s)|);$

9 $\quad$ **end**

10 **end**

11 $\pi^*(s) \leftarrow argmax_a \sum_{s'} P^a_{s,s'} [R(s,a,s') + \gamma V^*(s'), \forall s \in S;$

12 **return** $\pi^*(s), \forall s \in S$

For each state, determine the state value function, assuming that the agent takes the best possible action in that state under the current estimate of the value function.

Loop until the improvement is small

Define the optimal policy by selecting the action that maximise the state value of each state

Policy Iteration

- DP algorithm that converges to the optimal policy when the MDP has finite action and state spaces and bounded and stationary immediate reward function.
- Starts with initialisation:

1

for each $s_j \in \mathcal{S}$ **do**
  initialize $V(s_j)$ and $\pi(s_j)$ arbitrarily
end for

then divided into two phases:

- Policy evaluation
- Policy improvement

Policy Iteration - Policy evaluation

2

repeat
  $\Delta \leftarrow 0$
  for each $s_j \in \mathcal{S}$ **do**
    $v \leftarrow V(s_j)$
    $V(s_j) \leftarrow \sum_{s_k} p_{\pi(s_j)}(s_j, s_k) \left[R_{\pi(s_j)}(s_j, s_k) + \gamma V(s_k)\right]$
    $\Delta \leftarrow \max(\Delta, \|v - V(s_j)\|)$
  end for
until $\Delta < \epsilon$

- The state-value function for each state is updated iteratively until the difference between two consecutive values is small enough.

  - This technique converges to $V^{\pi}(s_j), \forall s_j \in S$ for $\gamma \in [0,1)$.

Policy Iteration - Policy improvement

3

policy-stable $\leftarrow$ true
for each $s_j \in \mathcal{S}$ **do**
  $b \leftarrow \pi(s_j)$
  $\pi(s_j) \leftarrow \text{argmax}_{x_i \in \mathcal{A}} \sum_{s_k} p_{x_i}(s_j, s_k) \left[R_{x_i}(s_j, s_k) + \gamma V(s_k) \right]$
  if $b \neq \pi(s_j)$ **then**
    policy-stable $\leftarrow$ false
  end if
end for

- In this phase, the algorithm looks for a policy π' that is better than the previously evaluated policy π.
  - For each state s_j, it updates the policy by searching the action X that maximises $V^{\pi}(s_j)$

Policy Iteration - iteration criterion

4

if policy-stable **then**
  stop
else
  go to 2 (Policy evaluation)
end if

Worst case complexity:

$$O\left(\frac{2^{N_S}}{N_S}\right)$$

- PI works iteratively by first evaluating $V^{\pi}(s_j)$, then finding a better policy π', evaluating $V^{\pi'}(s_j)$, finding a better policy π'', and so forth.
- When the same policy is found in two consecutive time steps, the algorithm has converged.

Evaluation

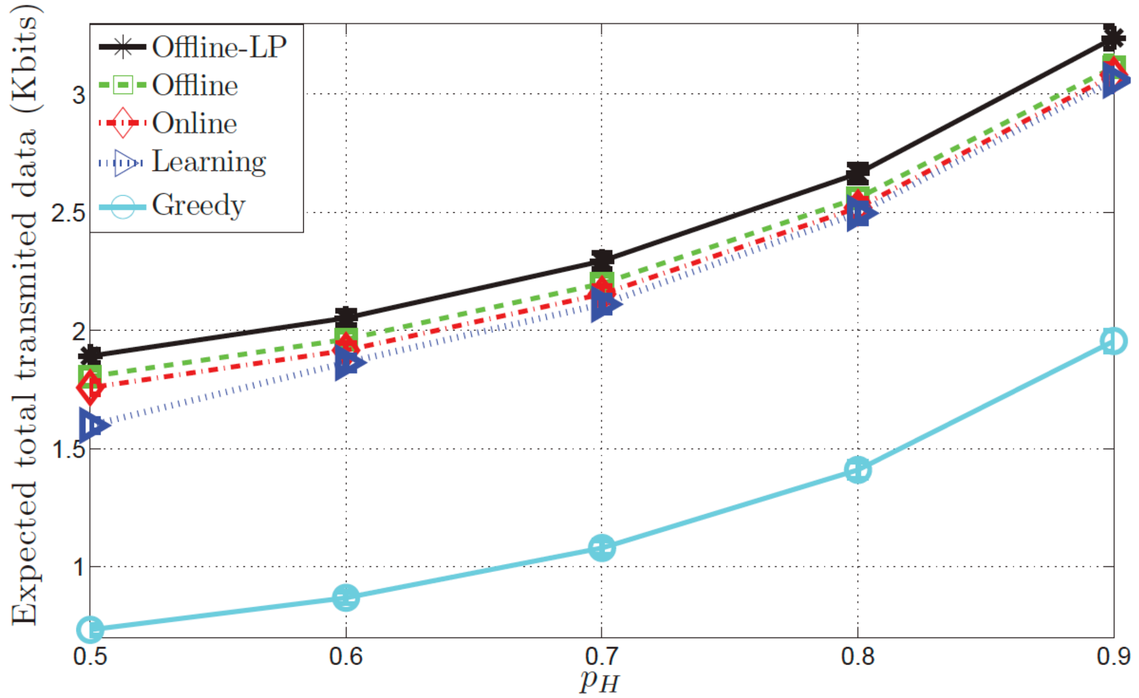


Fig. 3. Expected total transmitted data for $p_H = \{0.5, \ldots, 0.9\}$ and $B_{max} = 5$.

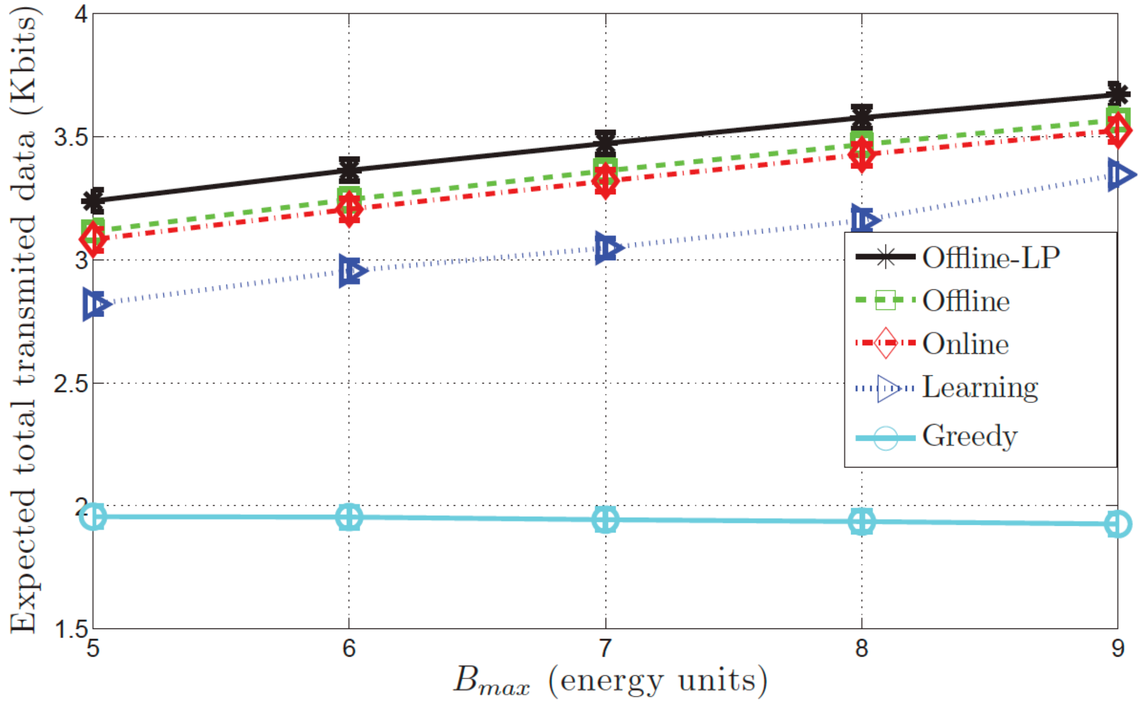


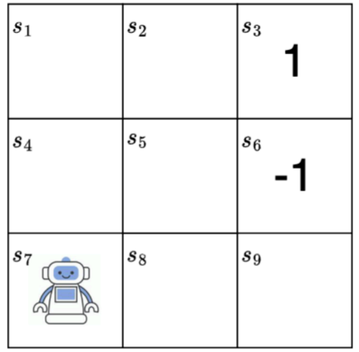
Fig. 4. Expected total transmitted data for $B_{max} = \{5, \ldots, 9\}$ and $p_H = 0.9$.

What are MDPs for?

- They model a huge variety of problems, which can have:
  - Finite state space

~10^{45} states

9 states

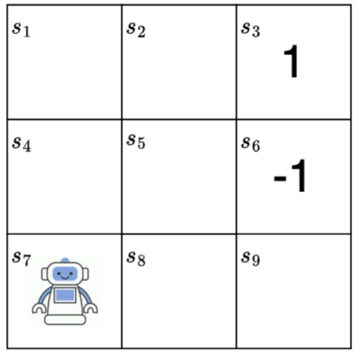


Do grid world and chess have similar state spaces?

  - Continuous state space

What are MDPs for?

- They model a huge variety of problems, which can have:
  - Finite horizon (final states exist)
  - Infinite horizon (no final state)



Q-learning

- If an agent does not know the environment, it has to **explore** it first. By exploring it, it learns the rewards and the state space.
- Q-learning is an algorithm for finding the optimal policy of a MDP (has convergence guarantees for both deterministic and non deterministic MDPs).
- Q-learning is one of the simplest **Reinforcement Learning algorithms**.
  - RL algorithms are **unsupervised** Machine Learning algorithms, i.e., they do not need to see labeled samples to learn tasks. They only need a feedback from the environment (state and reward).

Q-learning - finite horizon environments (1)

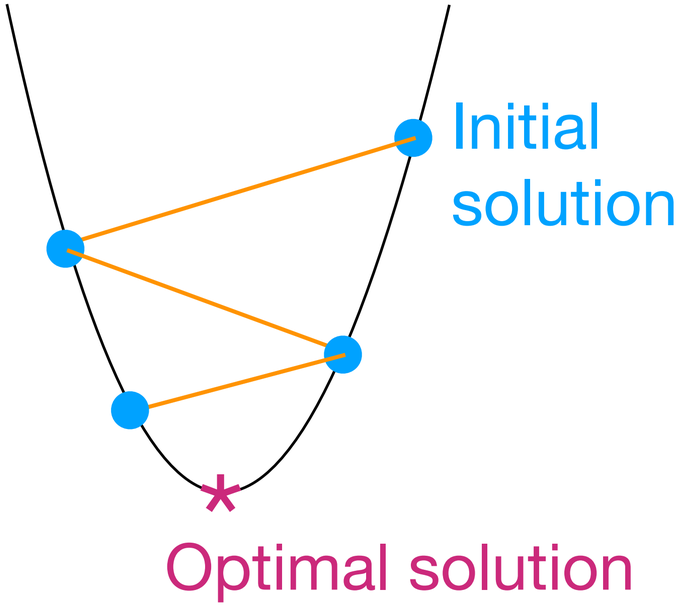
- Parameters: learning rate $\alpha \in (0,1]$, small $\epsilon > 0$
- Initialise $Q(s,a)$ randomly $\forall s \in S, a \in A$. Set $Q(s_{final}, \cdot) = 0 \ \forall a \in S_F$ (final states)
- For each episode:
  - Initialise initial state s
  - While $s \notin S_F$
    - Choose a possible action a from A_s derived from Q
    - Take action a, observe reward r and state s'
    - Update: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max Q(s',a) - Q(s,a)]$
    - $s \leftarrow s'$
  - Decrease ϵ

For example, ϵ-greedy approach:
- Compute $a^* = \arg\max Q(s,a)$
- With probability 1-ϵ
  - $a \leftarrow a^*$
- With probability ϵ
  - Choose a random a

Q-learning - parameters

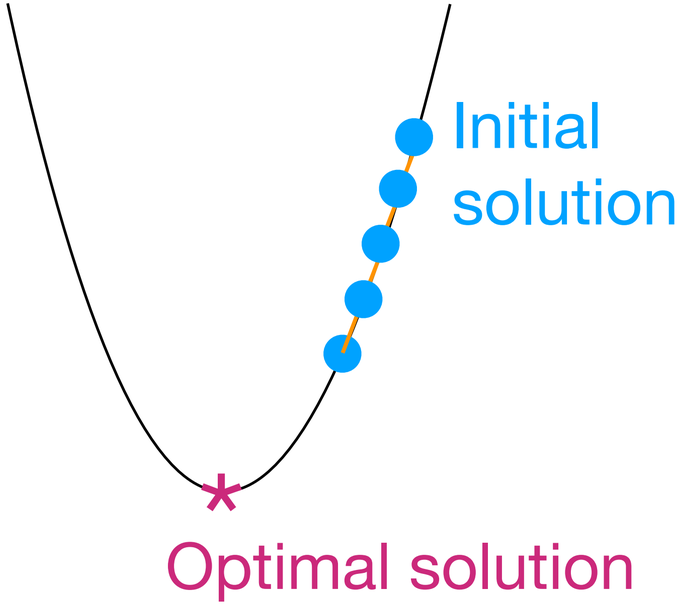
- In ϵ-greedy:
  - Large values of ϵ allow more **exploration** of the environment.
  - As we discover the environment, decreasing ϵ improves **exploitation** of the achieved knowledge
- α is the learning rate and represents how big is the step that we take to move towards the optimal solution.

High learning rate

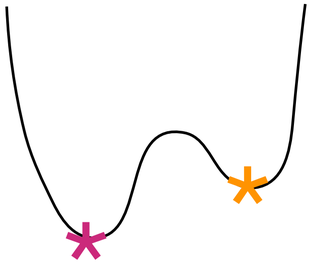


Solutions can overshoot or oscillate around the optimal solution

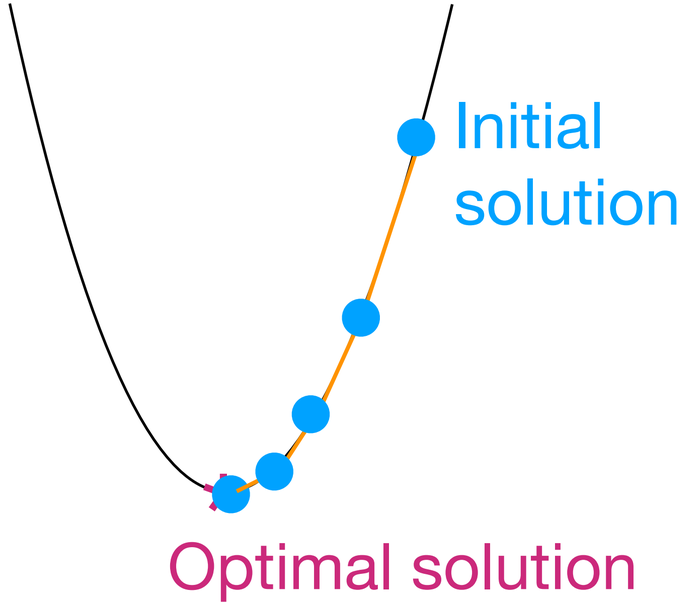
Low learning rate



Slow convergence rate
Solutions can get stuck to suboptimal solutions



Decaying learning rate



The learning rate decays and the solution quickly converges to the optimal one

Q-learning - finite horizon environments (2)

- Parameters: learning rate $\alpha \in (0,1]$, small $\epsilon > 0$
- Initialise $Q(s,a)$ randomly $\forall s \in S, a \in A$. Set $Q(s_{final}, \cdot) = 0 \ \forall a \in S_F$
- For each episode:
  - Initialise initial state s
  - While $s \notin S_F$
    - Choose a possible action a from A_s derived from Q
    - Take action a, observe reward r and state s'
    - Update: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max Q(s',a) - Q(s,a)]$
    - $s \leftarrow s'$
  - Decrease ϵ and α

Q learning - convergence

- The Q-learning algorithm finds the optimal Q^*, and hence the optimal policy π^* if
  - All state-action pairs are visited infinitely often (i.e., the procedure **explores** the environment enough, can be achieved with ϵ-greedy).
  - The rewards are bounded, $\exists R_{max} < \infty : |R_s| \leq R_{max} \forall s$.
  - The learning rate decreases, but not too slowly:
    - $\sum_{t=1}^{\infty} \alpha_t = \infty, \sum_{t=1}^{\infty} \alpha_t^2 < \infty$, for instance: $\alpha_t = \frac{1}{t}$.

Q-learning - finite horizon environments

- Parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$
- Initialise $Q(s,a)$ randomly $\forall s \in S, a \in A$ (no final states exist in this case)
- Initialise initial state s
  - While !stopping condition
    - Choose a possible action a from A_s derived from Q (e.g., ϵ-greedy)
    - Take action a, observe reward r and state s'
    - Update: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max Q(s',a) - Q(s,a)]$
    - $s \leftarrow s'$
  - Decrease ϵ and α

Bibliography

“A Learning Theoretic Approach to Energy Harvesting Communication System Optimization”, Pol Blasco, Deniz Gündüz, and Mischa Dohler

IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, VOL. 12, NO. 4, APRIL 2013.

Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8 (1992): 279-292.