Problem Set 2

Group 14 - Ragnhild Klette, Maya Evensen, Halvor Håvardsruud

February 16, 2026

Part 1

Exercise 1 - Markov Processes

1. (*) The model does not depend on history to find the future states, it only relies on the current state. This is the Markov property, and it is satisfied in this model.

2. (*)

   We model the behavior of the Mars Rover as a discrete-time Markov process $(\mathcal{S}, P)$, where $\mathcal{S}$ denotes the set of states and probabilities the transition probability matrix.

   The state space of the Markov process is defined as

   $$\mathcal{S} = \{E, P, S, R, M\},$$

   where:

   - E: Normal exploration mode
   - P: Taking pictures of the sky
   - S: Taking samples from the ground
   - R: Recharging
   - M: Malfunction (absorbing state)

	E	P	S	R	M
E	0.7	0.1	0.1	0.1	0
P	0.6	0	0.4	0	0
S	0.35	0	0.30	0.30	0.05
R	1	0	0	0	0
M	0	0	0	0	1

3. No, all states are not reachable from the normal operative mode. It can not reach malfunction (or shutdown). If the model has reached malfunction, it can not reach any other state except shutdown.

4. If the robot is in normal operative mode, the probability of malfunctioning within the next 10 minutes is 0. Within the next 20 minutes, there is only one path to malfunction, which is through "take samples". The probability of this happening is $0.1 \cdot 0.05 = 0.005$. Within the next 30 minutes, there are multiple paths to malfunction, starting from Explore, E $\rightarrow$:

   - E $\rightarrow$ S $\rightarrow$ M, $0.7 \cdot 0.1 \cdot 0.05 = 0.0035$
   - S $\rightarrow$ S $\rightarrow$ M, $0.1 \cdot 0.3 \cdot 0.05 = 0.002$
   - P $\rightarrow$ S $\rightarrow$ M, $0.1 \cdot 0.4 \cdot 0.05 = 0.0015$
   - Plus the probability of malfunctioning after 20 minutes, 0.005
   - $0.0035 + 0.002 + 0.0015 + 0.005 = 0.012$

   There is a 1.2% chance of malfunction after 30 minutes.

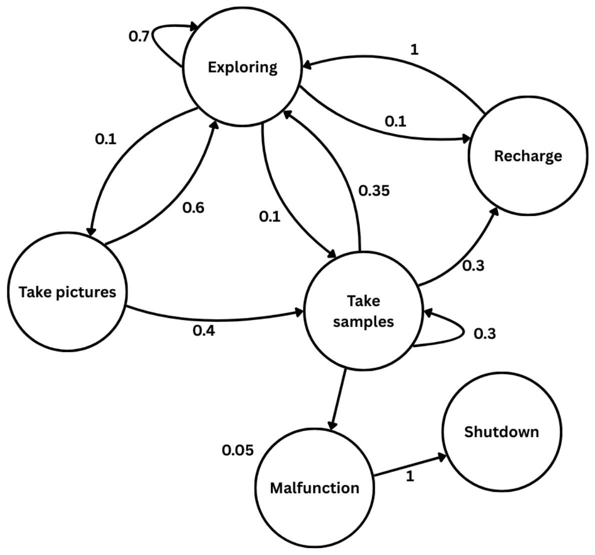

Figure 1: Mars rover Markov process graph representation

5. (*) If the probability of malfunctioning while in the sampling (digging) state depends on the number of times the robot has previously extracted samples, then the process no longer satisfies the Markov property in its current formulation. Since you no longer know the transition probabilities based on the current state. If the malfunction probability changes with the number of past digging actions, then knowing state S in no longer sufficient to determine the next state distribution.

   To recover Markovianity all information needed to determine future transitions need to be present in the current state. Adding the number of times the rover has sampled in the state definition. S_k = "sampling for the k-th time" would fix this issue and the process becomes Markovian again.

6. Yes. Model the board with the configuration of all Xs and Os and whose turn it is, then the state for the game at time t+1 is only dependent on the current state at time t.

7. There are 3^9 possible configurations of the tic-tac-toe board, since each of the 9 squares can be either empty, contain an X, or contain an O. Many of these configurations are not valid game states, for instance boards with five Xs and one O. Removing such illegal and unreachable configurations significantly reduces the effective size of the state space.

   The number of distinct states can be even more reduced by exploiting symmetries of the board, since multiple configurations are equivalent under rotations and reflections. Even after accounting for these reductions, the number of states remains large and grows exponentially with the size of the board.

   This exponential growth becomes evident when increasing the board size. For a 4×4 board, the total number of configurations is 3^16, which is 2187 times larger than the 3^9 configurations of the 3×3 board. This illustrates the rapid increase of the state space and shows scalability limitation of Markov process models for larger games.

Exercise 2 - Markov Reward Processes

1. (*)

2. E(10 minutes) $= 0.7 \cdot 0 + 0.1 \cdot 5 + 0.1 \cdot 20 + 0.1 \cdot 0 = 2.5$. The probability of getting a negative reward in twenty minutes is the same as the probability of getting a malfunction in twenty minutes, which is 0.5%

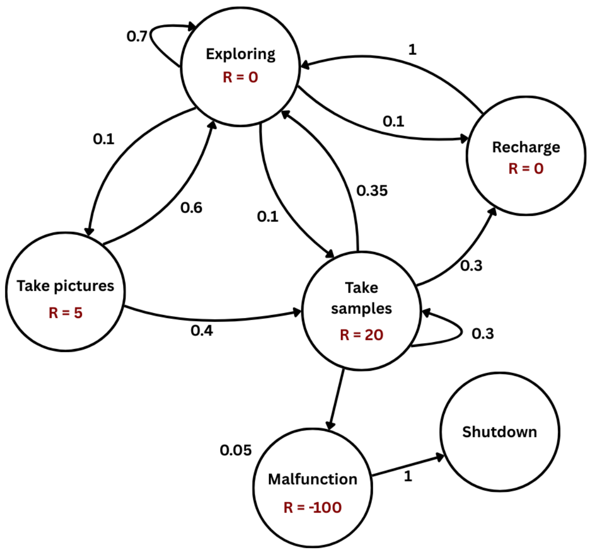

Figure 2: Mars rover MRP (Markov reward process)

Exercise 3 - Markov Decision Processes

1. (*)

States

The state space remains defined as:

$$\mathcal{S} = \{E, P, S, R, M\},$$

where:

- E: Normal exploration mode
- P: Taking pictures of the sky
- S: Taking samples from the ground
- R: Recharging
- M: Malfunction (absorbing state)

Actions

- $A(E)$ = {explore again, take pictures, take samples, recharge}
- $A(P)$ = {explore, take samples}
- $A(S)$ = {explore, take samples again, recharge, malfunction}
- $A(R)$ = {explore}
- $A(M)$ =$\{\phi\}$

Transition probabilities

Moving to state s' after taking action a in state s is:

$$P(s' \mid s, a)$$

Since the exploration state is the only state where the rover chooses the next state, all actions made from E has a transition probability of 1. Transition probabilities from all other states can be read from figure 2.

Policy

A policy π specifies the rover's behavior. E.g. we want a policy that maximizes the expected cumulative reward. So the rover must balance reward vs risk.

2. The optimal policies would most likely not be the same. When the policy is minimizing risk the model could easily avoid "take samples"- state making the risk and the reward = 0. If you want to maximize the reward the model would need to balance reward vs risk.

   If you have a short time horizon both models would still be exploring. The risk minimizing policy might not have encountered a malfunction yet. Therefor the earlier steps might be more similar compared to a longer time horizon where the model start exploiting more than exploring.

3. States: {H, nH}, (H: headache, nH: not headache)

   Actions: A(H) : {A, B, C}

   Rewards: H: 0, nH: positive reward

   Transitions: $P(s' \mid s, a)$

4. RL problems that depend on remembering more that just the current state. Non-stationary environments.

Exercise 4 - Bellman equations

1. (*) Bellman equations describe the recursive relationship of value functions. But a value function requires rewards. Therefor you cannot derive Bellman equations from MPs.

2.

$$v_\pi(s) = E_\pi[H^t|S^t = s], \qquad G^t = \sum_{i=0}^{T} \gamma^i R^{(t+i)}$$
$$v_\pi(s) = E_\pi[H^0|S^0 = s]$$
$$v_\pi(s) = E_\pi[\sum_{i=0}^{T} \gamma^i R^i|S^0 = s]$$
$$v_\pi(s) = E_\pi[R^0 + \sum_{i=1}^{T} \gamma^i R^i|S^0 = s]$$
$$v_\pi(s) = E_\pi[R^0 + \gamma G^1|S^0 = s]$$
$$v_\pi(s) = E_\pi[R^0 + \gamma v_\pi(s')|S^0 = s]$$

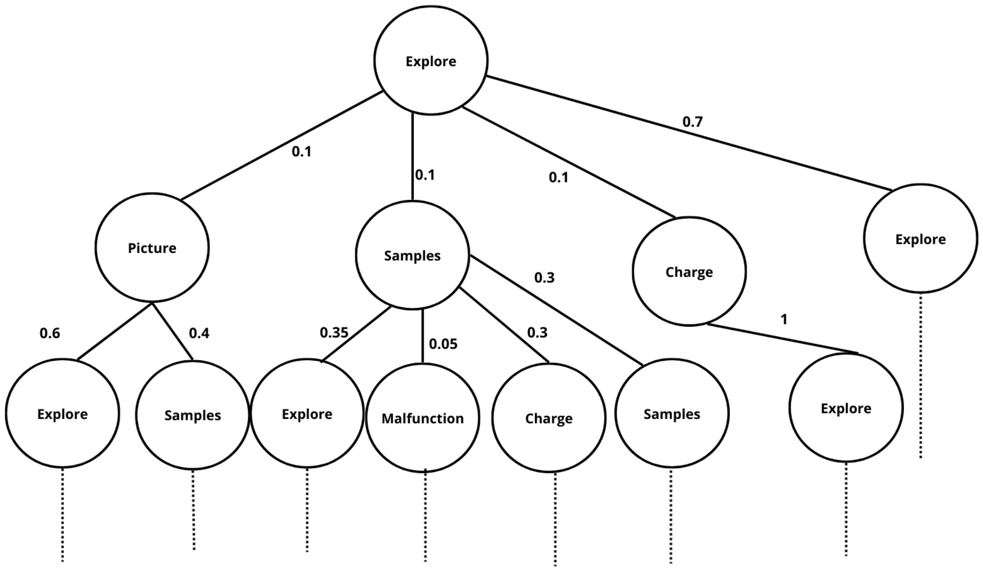

Figure 3: Backup diagram for MRP (4.2)

3. No. To define Bellman optimality equations require you to make an action and MRP - does not have actions therefore no choices and nothing to optimize.

   (a)

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[G_t \mid S_t = s,\ A_t = a] && (1)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s,\ A_t = a] && (2)\\
&= \mathbb{E}_\pi[R_{t+1} \mid S_t = s,\ A_t = a] + \gamma\,\mathbb{E}_\pi[G_{t+1} \mid S_t = s,\ A_t = a] && (3)\\
&= \mathbb{E}_\pi[R_{t+1} \mid s,a] + \gamma \sum_{s',a'} \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s',\ A_{t+1} = a']\ P(s',a' \mid s,a) && (4)\\
&= \mathbb{E}_\pi[R_{t+1} \mid s,a] + \gamma \sum_{s',a'} q_\pi(s',a')\, P(s',a' \mid s,a) && (5)\\
&= \mathbb{E}_\pi[R_{t+1} \mid s,a] + \gamma \sum_{s'} P(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, q_\pi(s',a') && (6)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma\, q_\pi(S_{t+1},A_{t+1}) \mid S_t = s,\ A_t = a]\,. && (7)
\end{aligned}$$

   (b)

$$\begin{aligned}
v_\pi(s) &= \sum_a \pi(a \mid s) \sum_{s',r} p(s',r \mid s,a)\ \big[r + \gamma v_\pi(s')\big] && (8)\\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s',r \mid s,a)\, r + \gamma \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s,a)\, v_\pi(s') && (9)\\
&= r_\pi(s) + \gamma \sum_{s'} P_\pi(s,s')\, v_\pi(s'). && (10)
\end{aligned}$$

$$\begin{aligned}
r_\pi(s) &= \sum_a \pi(a \mid s) \sum_{s',r} p(s',r \mid s,a)\, r, && (11)\\
P_\pi(s,s') &= \sum_a \pi(a \mid s)\, p(s' \mid s,a). && (12)
\end{aligned}$$

$$\begin{aligned}
v_\pi &= r_\pi + \gamma P_\pi v_\pi, && (13)\\
(I - \gamma P_\pi)\, v_\pi &= r_\pi, && (14)\\
v_\pi &= (I - \gamma P_\pi)^{-1} r_\pi. && (15)
\end{aligned}$$

   (c) If we derive the matrix form of the Bellman equations, we can solve the system exactly using linear solvers. This also gives theoretical insight into properties like existence, uniqueness, and convergence.

Exercise 5 - Optimality

1. No, $v\pi(s)$ is not an absolute or objective value. It depends on the given policy or model. By definition $v\pi(s) = E[G_t|S_t = s]$ the expectation value is given under policy π. Which means that $\pi_1 \neq \pi 2 \implies v\pi 1 \neq v\pi 2$. The value also depends on other factors such as transition probabilities, the reward function and the discount rate. Changing any of these will also change the expected return.

2. An optimal policy may not always need to be deterministic. There might be multiple different actions that are optimal given a state. If $q(s, a1) = 1(s, a2)$ then we have a stochastic optimal policy. and any mixture of these solution will be optimal.

3. A deterministic policy selects $\pi(s) = a$. This is pure exploitation. There is therefore no exploration. During learning, og we have a purely deterministic policy, it can prevent sufficient exploration, may giving us a suboptimal policy.

4. No. The optimal policy does not guarantee the highest return every turn. The optimal policy only guarantees $v\pi >= v'_\pi$ for all π'. The expectation is for the expected return over many turns.

Part II - Markov Report

1. $M = (S, A, T, R, \gamma)$

   States: The grid made up by 31x100 pixels where each state is S = (i, j). Terminal states are all states in the rightmost column S = (i, 99)

   Actions: The actions at state (i, j) is: $A((i, j)) = \{(i + 1, j), (i + 1, j - 1), (i + 1, j + 1)\}$. Straight to the right, diagonally to the top right and diagonally to the bottom right respectfully.

   Transition probability:

   - If inside bounds $P(s'|s, a) = 1$
   - Boundary rule: If robot tries to move above row 0 or below row 30 $\rightarrow$ it moves straight instead

   Reward function: Since we want to traverse the mountain in the fastest way possible - a smoother terrain gives a higher reward and a rougher terrain gives lower reward. $R(s') = roughness(s')$

   Discount factor: $\gamma \in [0, 1]$ Since we are finding the shortest path in a finite space $\gamma = 1$ is natural.

2. The value of going straight = expected immediate reward + $\gamma\cdot$ expected future value $v_{\pi_{str}}(s) = E_\pi[R^0 + v_{\pi_{str}}(s')|S^0 = s]$

3. (*) We assume that the task is asking to perform one iteration of the policy evaluation. When running policy evaluation with different sweep orders, the intermediate results differ significantly. After one sweep, we obtained:

   - Right-to-left (RTL): $V(0, 0) \approx -67.43$
   - Left-to-right (LTR): $V(0, 0) \approx -0.78$

   The difference occurs because updates are done in-place. Since transitions always move to the right, a right-to-left sweep updates "future" states first, allowing value information to propagate backward more quickly. A left-to-right sweep initially uses mostly un-updated values, so it reflects less of the long-term return after one pass.

4. (*) We learned that state ordering affects convergence speed but not the final value function (assuming full convergence). Right-to-left converges faster in this problem because it aligns with the direction in which the value information must propagate.

5. Figure 4

6. Figure 4

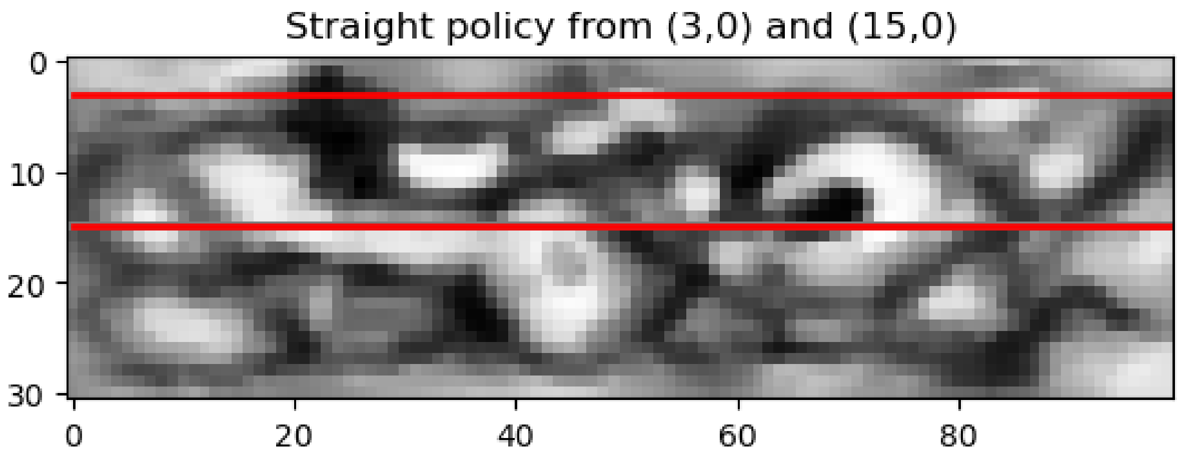

Figure 4: Straight policy trajectory from (3,0) and (15,0)

7. By following the straight policy trajectory, the optimal starting point would be at row index 7.

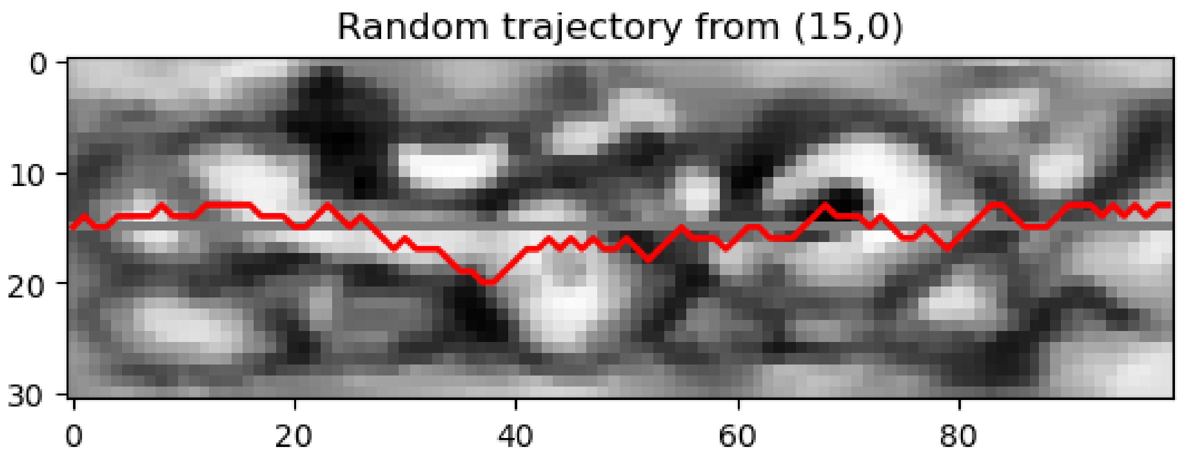

Figure 5: Random policy trajectory from (15,0)

8. Done in code
9. Done in code
10. (*) Yes, comparing the two value functions clearly shows that the performance of a policy depends on how it interacts with the terrain.

    Under the straight policy, the robot follows a fixed horizontal path, so the total return depends entirely on the roughness of that specific row. This makes the outcome predictable, but it also means the robot cannot avoid rough areas.

    Under the random policy, the robot moves up, down or straight with equal probability. While this introduces variability, it allows the robot to sometimes more into smoother terrain. In our results, the random policy actually achieved a higher value at (15, 0) than the straight policy, meaning it resulted in a lower expected travel time from that starting position.

    This shows that a simple deterministic strategy is not necessarily better - in this terrain, some flexibility can lead to improved expected performance.

11. (*) We compare the three values:

$$v_{\pi_{\text{str}}}(15,0) = -52.80$$
$$v_{\pi_{\text{str}}}(3,0) = -57.81$$
$$v_{\pi_{\text{rnd}}}(15,0) = -51.31$$

    Since higher values (i.e., less negative values) correspond to lower expected total travel time, the best combination is:

$$\pi_{\text{rnd}} \text{ starting from } (15,0)$$

    This combination has the highest value among the three and therefore yields the lowest expected travel time.

12. No, exploration does not affect the final result. The value functions were computed using dynamic programming and a known model of the environment, not from sampled experience. Therefore every state is evaluated directly through the Bellman expectation equation regardless of whether it would be visited under the policy. Exploration is only necessary in learning algorithms such as Monte Carlo or temporal-difference methods, where the agent must visit states to estimate their values. Here, both the straight and random policies converge to their true value functions independently of exploration.

13. Done in code

14. Figure 6

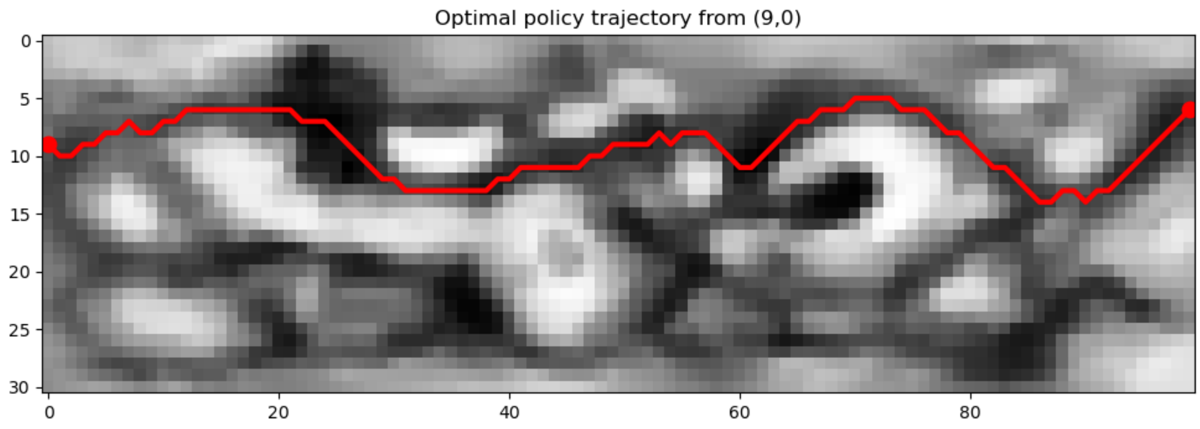

Figure 6: Standard policy iteration

15. Done in code

16. Figure 7

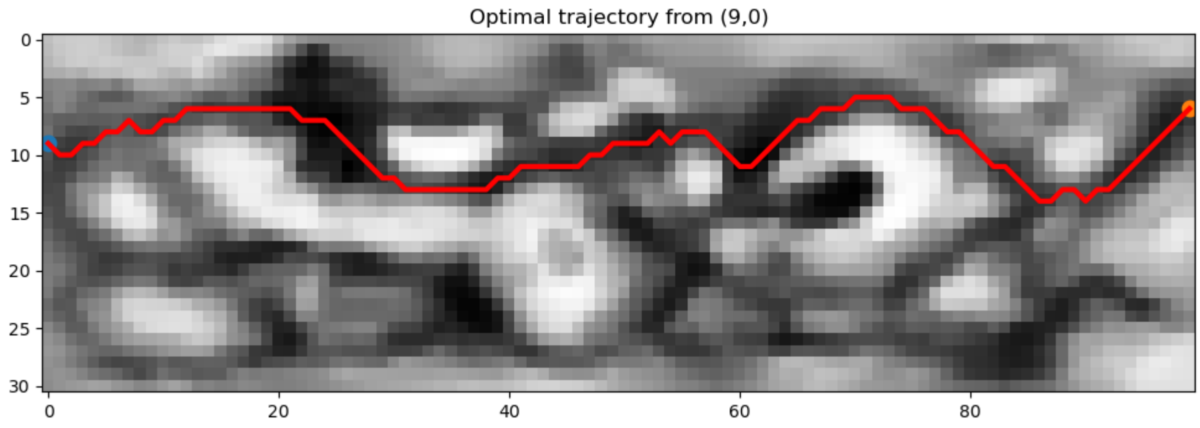

Figure 7: Truncated policy iteration

17. In k = 1 truncated policy iteration we alternate between one sweep of policy evaluation and greedy policy improvement. In direct value iteration we update the value function using the Bellman optimality equation by taking the maximum over actions at each step. Both approaches converge to the same optimal policy, but they differ in whether they explicitly maintain a policy during learning.

18. Figure 8

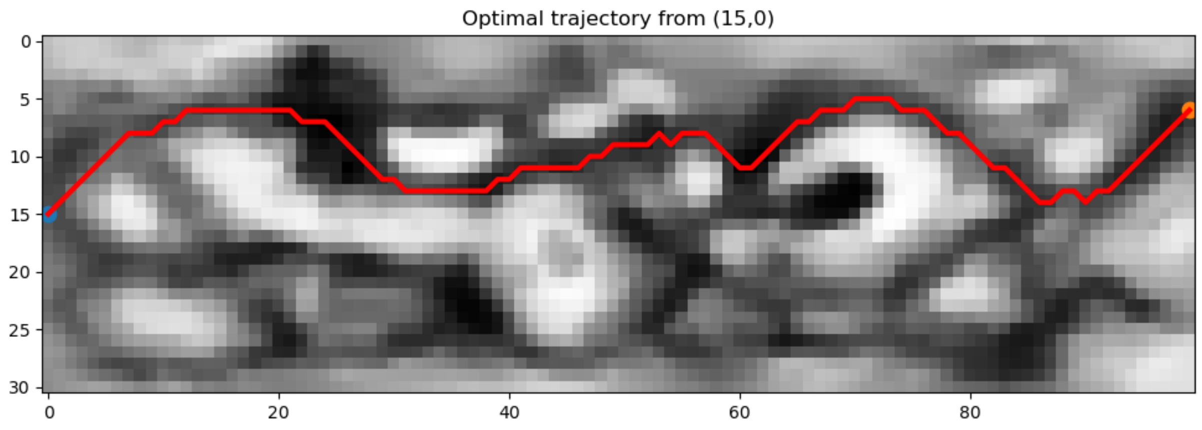

Figure 8: Value iteration

19. (*) Yes, the optimal policies obtained from policy iteration, truncated policy iteration, and value iteration are essentially identical. All algorithms converge to a solution of the Bellman optimality equations for the same MDP. Minor differences may appear due to numerical tolerance or tie-breaking when multiple actions have equal value, but the achieved returns and trajectories are effectively the same.

20. (*) Among the methods tested, direct value iteration was the fastest. The measured runtimes were approximately:

    - Value Iteration: 0.057 s
    - Policy Iteration: 0.266 s
    - Truncated Policy Iteration (k = 3): 0.412 s

    The reason value iteration performs best in this problem is that the environment is effectively acyclic: the robot always moves to the right. By sweeping states from right to left, value information propagates very efficiently toward the start states. As a result, only a small number of sweeps are needed.

    Policy iteration and truncated policy iteration include additional overhead from repeatedly maintaining and updating an explicit policy. In this gridworld, computing highly accurate intermediate value functions is not necessary to determine the optimal action, so the extra work makes these methods slower.