Markov Decision Processes and How to Solve Them

Rui-Yang Zhang

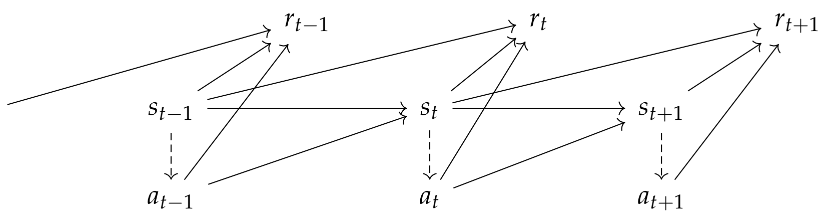
This is a stand-alone notes on the Markov decision processes and its solution methods - policy iteration and value iteration - that was originally used to supplement my talk at the Lancaster AI reading group in January 2025. Two key references used when preparing this notes are the RL short course by Chengchun Shi as well as the introductory RL book of Murphy (2024).

1 Markov Decision Processes

The **Markov decision process** (MDP) is a commonly used model in sequential decision-making (e.g. reinforcement learning). An MDP investigates the interactions between an **agent** and an **environment**, where at time t, the agent does **action** $a_t \in \mathcal{A}$ following some **policy** π and the environment provides feedback of **reward** r_t.

$$\begin{tikzcd}
\text{Agent} \arrow[r, bend left, "\text{action } a_t"] & \text{Environment} \arrow[l, bend left, "\text{reward } r_t"]
\end{tikzcd}$$

The environment has some internal status at time t, captured by state $s_t \in \mathcal{S}$, and its status changes to s_{t+1} after receiving the action a_t from the agent and the change is modelled stochastically via **transition probability** $p_S(s_{t+1}|s_t, a_t)$. Subsequently, the environment provides stochastic feedback to the agent via **reward probability** $p_R(r_t|s_t, a_t, s_{t+1})$. Several assumptions are made here. First, we assume the transition is Markovian, so the probability of the new state s_{t+1} only depends on the current state s_t and action a_t. Second, we assume the reward is instantly fully observed. Third, the probabilities p_S and p_R are time-homogeneous and have no direct time dependency. These assumptions are fundamental to an MDP[1].



Additionally, subsequent discussions in this notes will always assume we know the full world model (i.e. transition p_S and reward p_R). We would also assume the state space $\mathcal{S}$ and action space $\mathcal{A}$ are finite.

The goal of solving an MDP is to find the **optimal policy** π_* that yields the highest reward when interacting with the environment over time (either for a finite time horizon or till infinity). We wish to quantify the quality of a policy, and first, we need to summarise the reward. We define:

$$R(s_t, a_t, s_{t+1}) := \mathbb{E}_{p_R}[r_t] = \sum_r r p_R(r|s_t, a_t, s_{t+1})$$
$$R_t := R(s_t, a_t) := \mathbb{E}_{S_{t+1}\sim p_S}[R(s_t, a_t, S_{t+1})] = \sum_s R(s_t, a_t, s) p_S(s|s_t, a_t)$$

[1] If the rewards are only partially observed, in the sense that some filtering needs to be included to estimate the true reward, then we would have the partially observed MDP (POMDP). Also, if the time-homogeneity is broken, we would have the time-varying MDP.

The quality of a policy π that is initialised at state s can be captured using the **value function**, defined as

$$V_\pi(s) := \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t R_t(s_t, a_t)\ |s_0 = s\right]$$

where $\gamma \in (0,1)$ is the **discount factor**. The inclusion of a discount factor has two immediate reasons: (1) to avoid infinities in the sum, (2) to mimic the general tendency of preferring a reward of fixed value sooner. Closely linked to the value function is the **action-value function**, also known as the **Q-function**, defined as

$$Q_\pi(s,a) := \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t R_t(s_t, a_t)\ |s_0 = s, a_0 = a\right].$$

The value function and the Q-function are directly linked, as we have

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_{a\sim\pi(a|s)}[Q_\pi(s,a)] \\ Q_\pi(s,a) &= \mathbb{E}_{s_{t+1}\sim p_S(s|s_t,a_t)}[R_t(s,a) + \gamma V_\pi(s_{t+1})|s_t = s, a_t = a] \end{aligned}$$

which follow directly from their respective definitions. This conversion will be frequently used in the rest of this notes.

1.1 GridWorld

The recurring example that we will use in this notes is the GridWorld problem. The environment where our agent is acting is a square grid of size $s \times s$. In each of the $s \times s$ cells, the agent has four possible actions: move up, down, left, right by one unit. When a move is not possible, for example the agent is at the top-right corner of the grid and it cannot move further top/right, it has a smaller set of possible actions to choose from. Finally, upon visiting a cell, the agent will obtain a cell-specific reward that could be positive or negative.

To summarise, the state space is

$$\mathcal{S} = \{1, 2, \ldots, s\} \times \{1, 2, \ldots, s\}$$

and the action space is

$$\mathcal{A} = \{\text{'up'}, \text{'down'}, \text{'left'}, \text{'right'}\}.$$

The reward is deterministic and cell-specific. See Figure 1 for an example of a 3×3 GridWorld. We will use a discount factor $\gamma = 0.5$ for the computations we do - an arbitrary choice.

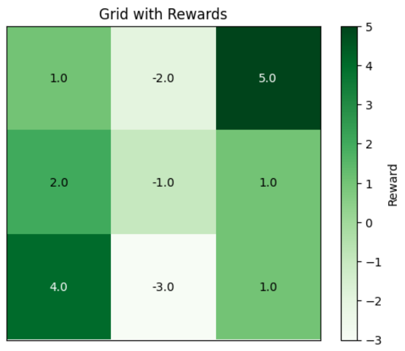


Figure 1: 3×3 GridWorld with Rewards

What we have used above is an extremely simple version of GridWorld. We can easily include roadblocks (cells that are unreachable) and terminating locations to turn the problem into a route-finding/maze-solving problem by having a -1 reward on all but terminating cells and a positive reward for the terminating cell. Many other possibilities exist.

The GridWorld problem can be thought of as a simplification of many real-world reinforcement learning tasks. For example, the agent could be a drone and the environment is some landscape we wish to survey.

2 Bellman Equation and Optimality

Bellman equations are fundamental equations related to an MDP. First, we will derive the **Bellman optimality equations** for the optimal policy π_*. Then we will use that to prove the existence of an optimal policy π_*. Finally, we will establish the Bellman equations for any policy π.

2.1 Bellman Optimality Equation

For an optimal policy π_* of an MDP, we have the following Bellman optimality equations that recursively define the value and Q-functions of π_*:

$$\begin{aligned} V_*(s) &= \max_a R(s,a) + \gamma \mathbb{E}_{p_S(s'|s,a)}[V_*(s')] \\ Q_*(s,a) &= R(s,a) + \gamma \mathbb{E}_{p_S(s'|s,a)}[\max_{a'} Q_*(s',a')] \end{aligned}$$

for all $s \in \mathcal{S}, a \in \mathcal{A}$. These allow us to solve for V_* and Q_* as systems of linear equations subject to knowing all the R and p_S, of course. Since we could obtain Q_* from V_*, we will only look at the value functions here.

By definition, the optimal policy π_* is the policy that yields the largest value function, i.e.

$$V_*(s) = \max_{\{a_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$$

where $\{s_t\}_{t=0}^{\infty}$ with $s_0 = s$ are generated using p_S. Using this equation, we have

$$\begin{aligned} V_*(s) &= \max_{\{a_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \\ &= \max_a R(s,a) + \max_{\{a_t\}_{t=1}^{\infty}} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \\ &= \max_a R(s,a) + \gamma \max_{\{a_t\}_{t=1}^{\infty}} \sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, a_t) \\ &= \max_a R(s,a) + \gamma \mathbb{E}_{s_1 \sim p_S(s'|s,a_0)}[V_*(s')] \end{aligned}$$

as required for the Bellman optimality equation for the value function.

A similar derivation can be used to obtain the Bellman optimality equation for the Q-function.

One could obtain an optimal (greedy) policy from V_* and Q_*. Given V_* and/or Q_*, an greedy optimal policy π_* is given as

$$\pi_*(s) = \operatorname*{argmax}_a Q_*(s,a) = \operatorname*{argmax}_a \Big(R(s,a) + \gamma \mathbb{E}_{p_S(s'|s,a)}[V_*(s)] \Big).$$

Note that the optimal policy may not be unique. Multiple optimal policies would exist and they will lead to different actions, however, they would all have the same value function V_*.

2.2 Existence of Optimal Policy

For an MDP with discrete state space $\mathcal{S}$ and action space $\mathcal{A}$ as well as bounded reward, there exists an optimal stationary[2] policy π_*.

The above result can be proved using the Banach fixed point theorem and the Bellman optimal equation of the value function. We define the **Bellman backup** operator $\mathcal{B}$ such that

$$\mathcal{B} \circ V(s) := \max_a R(s,a) + \gamma \mathbb{E}_{p_S(s'|s,a)}[V_*(s')] = V(s) \tag{1}$$

[2]same policy is applied for all t

where the second equality is the Bellman optimality equation. If we can show a fixed point exists for the operator $\mathcal{B}$, we could show that there exists at least an optimal policy. The Banach fixed point theorem provides a tool to establish such fixed point existence, as long as we can show that $\mathcal{B}$ is a contraction.

Proposition 2.1. *The Bellman backup operator $\mathcal{B}$ is a contraction for $\gamma < 1$.*

Proof. Consider the infimum norm $\|V\|_\infty := \sup_s |V(s)|$. For any V_1, V_2, we have

$$\begin{aligned}\|\mathcal{B} \circ V_1 - \mathcal{B} \circ V_2\|_\infty &= \sup_s |\mathcal{B} \circ V_1(s) - \mathcal{B} \circ V_2(s)| \\ &= \sup_s | \max_a \gamma \mathbb{E}_{p_S(s'|s,a)}[V_1(s')] - \max_a \gamma \mathbb{E}_{p_S(s'|s,a)}[V_2(s'))] \\ &\leq \gamma \max_a \sup_s |\mathbb{E}_{p_S(s'|s,a)}[V_1(s')] - \mathbb{E}_{p_S(s'|s,a)}[V_2(s')] \\ &= \gamma \max_a \sum_{s'} p_S(s'|s,a) \sup_s |V_1(s') - V_2(s')| \\ &\leq \gamma \|V_1 - V_2\|_\infty\end{aligned}$$

which is a contraction when $\gamma < 1$, as desired. □

2.3 Bellman Equation

A Bellman equation exists for any policy π, given by

$$V_\pi(s) = \mathbb{E}_\pi[R(s,a) + \gamma V_\pi(s_1)|s_0 = s]$$

where the action a is chosen following the policy π.

The above equation can be derived simply. We have

$$\begin{aligned}V_\pi(s) &= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| S_0 = s \right] \\ &= \mathbb{E}_\pi[R(s,a)] + \gamma \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, a_t) \middle| S_0 = s \right] \\ &= \mathbb{E}_\pi[R(s,a)] + \gamma \mathbb{E}_\pi \left[\mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, a_t) | S_1, S_0 \right] \middle| S_0 = s \right] \\ &= \mathbb{E}_\pi[R(s,a)] + \gamma \mathbb{E}_\pi \left[\mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, a_t) | S_1 \right] \middle| S_0 = s \right] \\ &= \mathbb{E}_\pi[R(s,a)] + \gamma \mathbb{E}_\pi \left[V_\pi(S_1) | S_0 = s \right] \\ &= \mathbb{E}_\pi \left[R(s,a) + \gamma V_\pi(S_1) | S_0 = s \right]\end{aligned}$$

as desired.

We can also express the Bellman equation V_π in matrix form

$$V_\pi = \mathcal{R} + \gamma \mathcal{P} V_\pi$$

where it is fully written out as

$$\begin{bmatrix} V_\pi(1) \\ V_\pi(2) \\ \vdots \\ V_\pi(n) \end{bmatrix} = \begin{bmatrix} \sum_a \pi(a|1) R(1,a) \\ \sum_a \pi(a|2) R(2,a) \\ \vdots \\ \sum_a \pi(a|n) R(n,a) \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \mathcal{P}_{21} & \cdots & \mathcal{P}_{2n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} V_\pi(1) \\ V_\pi(2) \\ \vdots \\ V_\pi(n) \end{bmatrix}$$

where $n = |\mathcal{S}|$, $\mathcal{P}_{ij} = \sum_a \pi(a|i)\mathcal{P}^a_{ij}$, and $\mathcal{P}^a_{ij} = \mathbb{P}[s_{t+1} = s'|s_t = s, a_t = a]$.

Simple linear algebra manipulation will give us

$$V_\pi = (1 - \gamma\mathcal{P})^{-1}\mathcal{R}$$

assuming the inconvertibility of $1 - \gamma\mathcal{P}$. Since $\mathcal{P}$ is an $|\mathcal{S}| \times |\mathcal{S}|$ matrix, the above computation is of time complexity $O(|\mathcal{S}|^3)$.

Alternatively, one could also find V_* iteratively from any starting point V_0 and repetitively applying

$$V_{k+1} \leftarrow \mathcal{R} + \gamma\mathcal{P}V_k$$

for $k = 0, 1, 2, \ldots$ - the Newton iteration method. This approach has time complexity $O(k|\mathcal{S}|^2)$ where k is the number of iterations needed till convergence (up to a sufficient degree). For large $|\mathcal{S}|$, the iterative approach is more desirable, and it can leverage parallel computing speed up as it only has matrix multiplications and additions - as opposed to the matrix inverse of the direct approach.

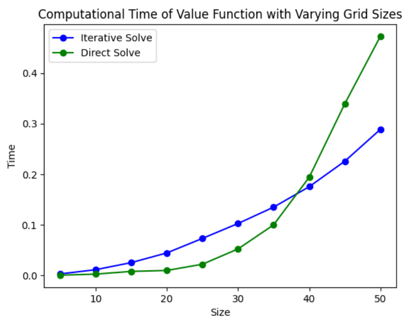


Figure 2: Computational Time Comparison of GridWorld's Value Function with varying sizes s using Direct Solver and Iterative Solver.

From Figure 2 where a computational time comparison of calculating the value functions of the GridWorld example with varying sizes using the two solution methods is shown, one can notice the two methods' respective scaling as well as the advantage of iterative solver for larger sizes.

3 Policy Iteration and Value Iteration

Two common approaches to solving an MDP are **policy iteration** and **value iteration**, which we will describe below.

3.1 Policy Iteration

The policy iteration method consists of iteratively applying (1) policy evaluation and (2) policy improvement. Starting from an arbitrary policy π, we first evaluate its value V_π using methods described in Section 2.3, then we use V_π to give us an improved policy π'. This two-step procedure is repeated until we reach convergence, i.e. the new policy π' agrees with π for all state $s \in \mathcal{S}$.

A policy π can be improved using its value function V_π, and in particular the Q-function Q_π which we can obtain from V_π using the relationship

$$Q_\pi(s, a) = \mathbb{E}_\pi[R(s, a) + \gamma V_\pi(s_{t+1})|s_t = s, a_t = a].$$

The new policy π' is defined by $\pi'(s) := \text{argmax}_a\, Q_\pi(s, a)$.

Theorem 3.1 (Policy Improvement Theorem)**.** *The improved policy π' has no smaller value function than π, i.e. $V_{\pi'}(s) \geq V_\pi\ \forall s \in \mathcal{S}$.*

Proof. This can be established using induction. We let $\pi_0 := \pi$, π_1 be the policy that makes the first move using $\operatorname{argmax}_a Q_\pi(s,a)$ and the rest using π, and π_k be the policy that makes the first k moves using Q_π. We would also let $\pi_\infty := \pi'$.

First, we can show that $\pi_1(s) \geq \pi_0(s)$ for $s \in \mathcal{S}$. Notice that,

$$Q_\pi(s, \pi'(s)) = \max_a Q_\pi(s,a) \geq \sum_a \pi(a|s) Q_\pi(s,a) = V_\pi(s)$$

and π_1 is no worse than π_0.

Subsequently, we have

$$V_{\pi_{k+1}(s)} - V_{\pi_k}(s) = \gamma^k \mathbb{E}_{\pi'}[Q_\pi(s_k, \pi'(s_k))|s_0 = s] - \gamma^k \mathbb{E}_{\pi'}[V_\pi(s_k)|s_0 = s] \geq 0$$

using a similar argument as above.

Thus, we can finish the proof using induction. □

The method of policy iteration is slow. For each iteration, we need to evaluate the policy, which is $O(k_1|\mathcal{S}|^2)$ when we do it iteratively, and improve the policy, which is $O(|\mathcal{A}| \times |\mathcal{S}|^2)$. So, if k is used to denote the number of policy iterations, the overall method has time complexity of $O(k(k_1|\mathcal{S}|^2 + |\mathcal{A}| \times |\mathcal{S}|^2))$

3.2 Value Iteration

Value iteration is simple - we find V_*, then define an optimal policy π_* using it.

Recall that $V_* = \mathcal{B} \circ V_*$ where $\mathcal{B}$ is the Bellman backup operator we defined earlier as (1). We can start from some V_0 and apply iteratively $\mathcal{B}$ to it until convergence to obtain V_*. This can be justified as $\mathcal{B}$ is a contraction, as proved in Proposition 2.1. An example of this convergence is also shown in Figure 3.

Finally, one could obtain an optimal (greedy) policy from V_* and Q_*. Given V_* and/or Q_*, an greedy optimal policy π_* is given as

$$\pi_*(s) = \operatorname*{argmax}_a Q_*(s,a) = \operatorname*{argmax}_a \left(R(s,a) + \gamma \mathbb{E}_{p_S(s'|s,a)}[V_*(s)] \right).$$

Thus, the time complexity of value iteration is $O(k \cdot |\mathcal{S}|^2 \times |\mathcal{A}|)$ since the one-off policy finding is negligible compared to the main value iteration.

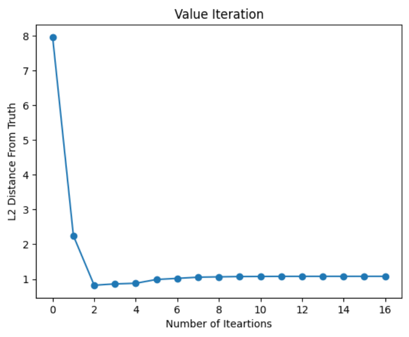


Figure 3: Value Iteration of GridWorld. L2 distance to optimal value function (obtained from direct solve) over the number of iterations.

3.3 Solving GridWorld

Returning to the GridWorld example that we outlined in Section 1.1, we will use the methods we have outlined to find an optimal policy of it.

Considering a simple 3×3 example with randomly generated rewards, shown in Figure 1, we apply three methods of solving MDP: direct solution via Bellman optimality equation (Section 2.1), value iteration (Section 3.2), and policy iteration (Section 3.1. Their respective optimal value functions obtained are presented below directly from the code output. The minor discrepancies between the approximate solutions and the direct solution is mostly due to numerical rounding-offs and the choice of tolerance level for iterative solvers.

```
Optimal Value Function (Direct Solution):
[[5.6010101  7.65656566 4.78282828]
 [6.74747475 5.55555556 7.47474747]
 [5.32828283 6.56565657 4.51010101]]

Optimal Value Function (Value Iteration):
[[5.33333302 7.3333329  4.66666639]
 [6.66666627 5.33333302 7.3333329 ]
 [5.33333302 6.66666627 4.66666639]]

Optimal Value Function (Policy Iteration):
[[5.33333325 7.33333329 4.66666658]
 [6.66666651 5.33333325 7.33333329]
 [5.33333325 6.66666651 4.66666658]]
```

Subsequently, one could use an optimal value function to find an optimal policy using the Q-function. This extra step is not needed for policy iteration as that method directly looks for the policy. The policies obtained are presented below. Notice that they are identical.

```
Optimal Policy (Direct Solution):
[['down' 'right' 'down']
 ['down' 'left' 'up']
 ['up' 'left' 'up']]

Optimal Policy (Value Iteration):
[['down' 'right' 'down']
 ['down' 'left' 'up']
 ['up' 'left' 'up']]

Optimal Policy (Policy Iteration):
[['down' 'right' 'down']
 ['down' 'left' 'up']
 ['up' 'left' 'up']]
```

References

Murphy, K. (2024). Reinforcement learning: An overview, *arXiv preprint arXiv:2412.05265* .