# Summary Markov Decision Theory exam

Mariya Karlashchuk

April 25, 2022

## Contents

# 1 Markov decision problems in general

## 1.1 Markov Decision Process basics

Whenever we consider a Markovian problem we want to find the optimal policy and whether it has a particular form. Since certain problems are set up differently (small/large scale) we want to find an algorithm that computes an optimal policy in the most efficient way.

The basic MDP model is always set up as follows:

- Decision epochs; $T = 1, 2, \ldots, N$
- States; $s \in S$, discrete, do not change over time
- Actions; $a \in A_s$, discrete, do not change over time
- Action selection; a probability distribution $q(\cdot)$ with which an action $a$ is selected
- Direct reward; $r_t(s, a)$, $r_N(s)$ is the salvage value
- Transition probabilities; $p_t(\cdot \mid s, a)$ probability of transitioning from state $s$ to state $\cdot$

The Markov Decision Process is then defined as $\{T, S, A_s, r_t(s, a), p_t(\cdot \mid s, a)\}$.

For every MDP we have several decision rules which can occur, there are either deterministic (not random) and randomized (random). These are

- Deterministic Markovian; we decide on picking an action from the action space
- Deterministic history dependent; we decide to pick an action dependent based on the history of our MDP
- Randomized Markovian; same as MD, but the action is chosen with a certain probability based on state
- Randomized history dependent; same as HD, but action is chosen with a probability based on the history

It might occur that we will use a stationary policy, this means that we have a set of decisions which do not change over time.

## 1.2 Induced stochastic process

Assume $X_t, Y_t, Z_t$ to be the state action and history at time $t$ respectively. Then the policy $\pi \in \Pi^{HR}$ and an initial state $s_1$ induce a stochastic process $P^\pi = (X_1, Y_1, X_2, Y_2, \ldots)$ as follows:

$$P^\pi(X_1 = s) = \begin{cases} 1 \text{ if } s = s_1 \\ 0 \text{ otherwise} \end{cases}$$

$$P^\pi(Y_1 = a \mid Z_t = h_t) = q_{d_t(h_t)}(a)$$

$$P^\pi(X_{t+1} = s \mid Z_t = (h_{t-1}, a_{t-1}, s_t), Y_t = a_t) = p_t(s \mid s_t, a_t)$$

## 1.3 Induced discrete time Markov chain

We observe that $P^\pi(X_{t+1} = s \mid Z_t = (h_{t-1}, a_{t-1}, s_t), Y_t = a_t)$ depends on the history through

$$X_t = s_t \text{ and}$$
$$P^\pi(Y_t = a \mid Z_t = h_t) = q_{d_t(h_t)}(a)$$

However, for the Markovian policies we get that

$$P^\pi(Y_t = a \mid Z_t = h_t) = P^\pi(Y_t = a \mid X_t = s_t) = q_{d_t(s_t)}(a)$$

so we get that the process does not become dependent on the history anymore, which makes is a Markov chain.

# 2 Finite horizon Markov decision problems

In order to find the best reward sequence in an MDP we want to compare the expected total rewards for different reward sequences. We can only compare two policy sequences when they are comparable (outcome must be the same for each type of reward function). We express the expected total reward as the utility $\Psi$, which is a real-valued function that represents the decision maker's preference for certain elements (in this case rewards). This is defined as

$$v_N^\pi(s) = E_s^\pi \left[ \sum_{t=1}^{N-1} r_t(X_t, Y_t) + r_N(X_N) \right]$$

We define an optimal policy as $v_N^{\pi^*}(s)$, for which holds that $v_N^{\pi^*}(s) > v_N^\pi(s)$. This means that the value of an optimal policy is larger than the value for any other policy. These optimal policies do not need to exist. We can also consider $\epsilon$-optimal policies, for which holds that $v_N^{\pi_\epsilon^*}(s) + \epsilon > v_N^\pi(s)$.

## 2.1 Finite-horizon policy evaluation algorithms for fixed HD and HR

For fixed $\pi \in \Pi^{HD}$ we can set up the following evaluation algorithm:

1. Set $t = N$ and $u_N^\pi(h_N) = r_N(s_N)$ for all $h_N = (h_{N-1}, a_{N-1}, s_N) \in H_N$.
2. If $t = 1$ then stop else go to step 3.
3. Set $t := t - 1$. Compute for each $h_t$

$$u_t^\pi(h_t) = r_t(s_t, d_t(h_t)) + \sum_{j \in S} p_t(j|s_t, d_t(h_t)) u_{t+1}^\pi((h_t, d_t(h_t), j).$$

4. Go to step 2.

For fixed $\pi \in \Pi^{HR}$ we can set up the following evaluation algorithm:

1. Set $t = N$ and $u_N^\pi(h_N) = r_N(s_N)$ for all $h_N = (h_{N-1}, a_{N-1}, s_N) \in H_N$.
2. If $t = 1$ then stop else go to step 3.
3. Set $t := t - 1$. Compute for each $h_t$

$$u_t^\pi(h_t) = \sum_{a \in A_{s_t}} q_{d_t(h_t)(a)} \left\{ r_t(s_t, a) + \sum_{j \in S} p_t(j|s_t, a) u_{t+1}^\pi((h_t, a, j)) \right\}.$$

4. Go to step 2.

## 2.2 Optimality equations

The optimality equations give us the optimal returns and are a useful method to determine whether a policy is optimal or not. They are defined as

$$u_t(h_t) = \sup_{a \in A_{s_t}} \left( r_t(s_t, a) + \sum_{j \in S} p_t(j \mid s_t, a) u_{t+1}((h_t, a, j)) \right)$$

We can find the optimal policy by solving the optimality equations (system of linear equations). This can be done for history dependent and Markovian decision rules. We know that there exists an optimal deterministic Markovian policy if $\forall s_t, t \; \exists a' \in A_{s_t}$, such that

$$a' = \arg \sup_{a \in A_{s_t}} \left( r_t(s_t, a) + \sum_{j \in S} p_t(j \mid s_t, a) u_{t+1}^*((h_t, a, j)) \right)$$

There exists a deterministic Markovian policy that is optimal if

- $A_s$ is finite $\forall s$ or
- $A_s$ is compact, $r_t(s,a)$ continuous for each $s$, $\exists M < \infty$ such that $|r_t(s,a)| < M$ and $p_t(j|s,a)$ continuous

### 2.3 Backward induction algorithm

This is one of the algorithms that is used to determine the optimal value for each time step over the full time horizon.

1. Start at the end of the time horizon, so $t = N$
2. Determine the optimal value for all states $s_N$
3. Set $t := t - 1$, and compute

$$u_t^*(s_t) = \max_{a \in A_{s_t}} \left( r_t(s_t, a) + \sum_{j \in S} p_t(j \mid s_t, a)\, u_{t+1}^*(j) \right)$$

4. Choose the action that provides the best action and use the value with that corresponding action for the next computation
5. Stop when $t = 1$

## 3 Discounted reward Markov decision problems

Discounted MDPs arise when accounting for the time value of rewards. The expected total discounted reward is defined as

$$v_\lambda^\pi(s) = E_s^\pi \left[ \sum_{t=1}^{\infty} \lambda^{t-1} r(X_t, Y_t) \right]$$

We analyse discounted MDPs with Markov policies only, since the induced stochastic process for a HR/HD policy can be rewritten in MR/MD form. To find the optimal policy we try to solve the following set of linear equations:

$$v_\lambda^\pi = r_d + \lambda P_d v_\lambda^{d^{\pi'}}$$

This is written in vector notation. Whenever we have a stationary policy, the value on the left-hand side and on the right-hand side both become $v_\lambda^{d^\infty}$. The discount reward of a stationary policy is the unique solution of a fixed point equation.

Technically, whenever we are dealing with a randomized Markovian policy, we try to find the unique solution of

$$v = r_d + \lambda P_d v \text{ where}$$
$$v_\lambda^{d^\infty}(I - \lambda P_d)^{-1} r_d$$

The optimality equations that one actually needs to solve are defined as

$$v(s) = \sup_{a \in A_s} \left\{ r(s,a) + \sum_{j \in S} \lambda p(j \mid s, a) v(j) \right\}$$

The fact that the value of a discounted MDP satisfies the optimality equations can be proven using the Banach fixed-point theorem.

A policy $\pi^*$ is optimal if and only if $v_\lambda^{\pi^*}$ solves the optimality equations. We can also set up several conditions for an optimal deterministic policy to exist. These are (if we assume that $S$ is discrete)

- $A_s$ is finite for each $s \in S$, or
- $A_s$ is compact, $r(s,a)$ continuous for each $s$ and $p(j|s,a)$ continuous, or
- $A_s$ is compact, $r(s,a)$ upper semicontinuous for each $s$ and $p(j|s,a)$ lower semicontinuous

Whenever optimal policies do not exist, we opt to consider $\epsilon$-optimal policies $\pi^*_\epsilon$ for which must hold $v_\lambda^{\pi^*_\epsilon} \geq v^*_\lambda - \epsilon e$.

# 4 Algorithms for discounted reward Markov decision problems

There are three algorithms that are used to find optimal policies for finite discounted MDPs, namely value iteration, policy iteration and linear programming. Value iteration only returns $\epsilon$-optimal policies, whereas policy iteation and linear programming return optimal policies.

## 4.1 Value iteration

The idea of this algorithm is that we are constantly solving optimality equations for each state. We find action that provides us the largest value and use this in our further calculations. This way we construct an $\epsilon$-optimal policy.

1. Initialization.
   Select $v^0 \in V$, specify $\varepsilon > 0$ (desired accuracy), set $n = 0$.
2. Iteration.
   For each $s \in S$ compute
   $$v^{n+1}(s) = \max_{a \in A_s} \left( r(s,a) + \lambda \sum_{j \in S} p(j|s,a) v^n(j) \right).$$
3. Convergence test.
   If $\|v^{n+1} - v^n\| < \varepsilon(1-\lambda)/(2\lambda)$, go to step 4.
   Otherwise increment $n$ by 1 and return to step 2.
4. Final result.
   For each $s \in S$ choose
   $$d_\varepsilon(s) \in \arg\max_{a \in A_s} \left( r(s,a) + \lambda \sum_{j \in S} p(j|s,a) v^{n+1}(j) \right).$$

## 4.2 Policy iteration

For the policy iteration we start with a certain initial policy, where compute a certain value. After, we fix the value of the chosen policy and use it to determine a new policy. If the policy does not need to change, we are done. Otherwise we fix the new policy and continue the process.

1. Initialization.
   Set $n = 0$ and select $d_0 \in D^{\mathrm{MD}}$ arbitrary.
2. Policy evaluation.
   Obtain $v^n$ by solving $(I - \lambda P_{d_n})v = r_{d_n}$.
3. Policy improvement.
   Choose $d_{n+1} \in \arg\max_{d \in D^{\mathrm{MD}}} (r_d + \lambda P_d v^n)$ componentwise, setting $d_{n+1} = d_n$ if possible.
4. Stop criterion.
   If $d_{n+1} = d_n$ then stop and set $d^* = d_n$. Otherwise increment $n$ by 1 and return to step 2.

### 4.3 Linear programming

In linear programming we set up a linear program for the MDP that we are dealing with. Ofttimes it is difficult to compute the primal problem, hence it needs to be rewritten in a dual problem. Usually the solution is provided for the discounted MDP.

- Let $\alpha(j) > 0$, $j \in S$ such that $\sum_{j \in S} \alpha(j) = 1$.
- Primal linear program:

  $\min_v \sum_{j \in S} \alpha(j) v(j)$

  subject to $v(s) - \sum_{j \in S} \lambda p(j|s,a) v(j) \geq r(s,a)$

  for $a \in A_s$, $s \in S$, and $v(s)$ unconstrained.

- Dual linear program:

  $\max_x \sum_{s \in S} \sum_{a \in A_s} r(s,a) x(s,a)$

  subject to $\sum_{a \in A_j} x(j,a) - \sum_{s \in S} \sum_{a \in A_s} \lambda p(j|s,a) x(s,a) = \alpha(j)$

  and $x(s,a) \geq 0$ for $a \in A_s$, $s \in S$.

## 5 Average reward Markov decision problems

We use average reward MDPs whenever we need to make frequent decisions or whenever the discount value is very close to 1. Therefore, there are a bit less assumptions we need to make for the average reward MDPs. Observe that we are making computations with the gain and the bias. We define the average expected reward aka the gain of a history dependent randomized policy to be

$$g^\pi(s) = \lim_{N \to \infty} \frac{1}{N} E_s^\pi \left\{ \sum_{t=1}^{N} r\left(X_t, Y_t\right) \right\} = \lim_{N \to \infty} \frac{1}{N} v_{N+1}^\pi(s)$$

If this limit does not exist, we consider the lim inf average reward or the lim sup average reward. If there are equal, the gain exists. In order to see this, we must determine the direct reward sequence, from this we compute the total reward sequence, and eventually the average reward sequence. We will see to which value both limits converge, which allows us to draw conclusions regarding the gain for a certain policy. In conclusion, a policy hence can be either average optimal, lim sup average optimal or lim inf average optimal. The gain for HR policies equal the gain for MR policies.

### 5.1 Gain and bias for Markov Reward Process

Define the Cesaro limit to be $P^* = lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N-1} P^n$. Then the following properties hold:

- $P^* = eq^T$ with $q$ being the stationary distribution of $P$
- $PP^* = P^*P = P^*P^* = P^*$

Using the Cesaro limit, we can say the following about the gain for a stationary policy:

$$g^{d^\infty}(s) = \lim_{N \to \infty} \frac{1}{N} v_{N+1}^{d^\infty}(s) = P_d^* r_d(s)$$

An MDP with a stationary policy generates an MRP with a transition matrix $P_d$ and reward $r_d$. The gain is a constant function if our Markov chain is irreducible or has a single recurrent class and possibly some transient states.

We define the bias of an MRP to be $h = H_p r$, where $H_p = (I - P + P^*)^{-1}(I - P^*)$. When computing the bias for each state, we can set one bias to 0.

The gain $g$ and the bias $h$ in an MRP can be computed as follows:

$$(I - P)g = 0$$
$$g + (I - P)h = r$$

### 5.2 Optimality equations

Often we consider unichain models, where we have a constant optimal gain. A transition matrix corresponding to every deterministic stationary policy unichain. So we have a single recurrent class, plus a possibly empty set of transient states.

We can compute the average reward uniquely by solving $ge + (I - P)h = r$. This in case for $P$ unichain or irreducible. The optimality equations for the MDP can be solved by the gain and bias of the MDP. These are set up as

$$0 = \max_{a \in A_s} \left\{ r(s,a) - g + \sum_{j \in S} p(j \mid s,a)h(j) - h(s) \right\}$$

or $0 = \max_{a \in A_s}\{r_d - ge + (P_d - I)h\}$ in matrix notation. Also, define $B(g,h) = \max_{a \in A_s}\{r_d - ge + (P_d - I)h\}$. Whenever $B(g^*, h^*) = 0$ we have an optimal stationary policy that is average optimal if $d^*$ is $h^*$-improving. The latter means that the decision rule is the largest for $r_d + P_d h$.

## 6 Algorithms for average reward Markov decision problems

Again, we have here that the value iteration returns $\epsilon$-optimal policies, whereas the other algorithms return optimal policies. Each algorithm functions in the same fasion as for the discounted MDP, except that the stop criterion and type of computation is different.

### 6.1 Value iteration

We need to consider here the fact that $sp(v) = \max_s v(s) - \min_s v(s)$.

1. Initialization.
   Select $v^0 \in V$, specify $\varepsilon > 0$, set $n = 0$.
2. Iteration.
   For each $s \in S$ compute
   $$v^{n+1}(s) = \max_{a \in A_s} \left\{ r(s,a) + \sum_{j \in S} p(j|s,a) v^n(j) \right\}.$$
3. Stop criterion.
   If $sp(v^{n+1} - v^n) < \varepsilon$ then go to step 4.
   Otherwise increment $n$ by 1 and return to step 2.
4. Final result.
   For each $s \in S$ choose
   $$d_\varepsilon(s) \in \arg\max_{a \in A_s} \left\{ r(s,a) + \sum_{j \in S} p(j|s,a) v^n(j) \right\}$$
   and stop.

The value iteration algorithm provides us several bounds on the gain, namely

$$g' = \frac{1}{2}\left[ \max_{s \in S} \left( v^{n+1}(s) - v^n(s) \right) + \min_{s \in S} \left( v^{n+1}(s) - v^n(s) \right) \right] \text{ will result in}$$
$$|g' - g^*| < \epsilon/2 \text{ and } |g' - g^{(d_\epsilon)^\infty}| < \epsilon/2$$

## 6.2 Policy iteration

As mentioned before, we can set the bias for one of the states to 0, then choose $h_n$ to satisfy $P^*_{d_n} h_n = 0$ or $-h_n + (P_{d_n} - I)w = 0$ for some value $w$.

1. Initialization.
   Set $n = 0$ and select $d_0 \in D^{\text{MD}}$ arbitrary.
2. Policy evaluation.
   Obtain scalar $g_n$ and $h_n \in V$ by solving
   $$0 = r_{d_n} - ge + (P_{d_n} - I)h_n.$$
3. Policy improvement.
   Choose $d_{n+1}$ to satisfy
   $$d_{n+1} \in \arg\max_{d \in D^{\text{MD}}} \{r_d + P_d h_n\},$$
   setting $d_{n+1} = d_n$ if possible.
4. Stop criterion.
   If $d_{n+1} = d_n$ then stop and set $d^* = d_n$. Otherwise increment $n$ by 1 and return to step 2.

Interesting to mention is that at successive iterates we obtain a larger gain, otherwise we obtain a larger bias.

## 6.3 Linear programming

For linear programming we set up a primal, but preferrably we solve the dual program. For the latter we can conclude what the optimal decision rules should be.

- Primal linear program:

$$\begin{aligned} &\min\ g \\ &\text{subject to } g + h(s) - \sum_{j \in S} p(j|s,a)h(j) \geq r(s,a), \quad a \in A_s,\ s \in S \\ &\qquad g, h \text{ unconstrained} \end{aligned}$$

- Dual linear program:

$$\begin{aligned} &\max_x \sum_{s \in S} \sum_{a \in A_s} r(s,a)x(s,a) \\ &\text{s.t.} \sum_{a \in A_j} x(j,a) - \sum_{s \in S} \sum_{a \in A_s} p(j|s,a)x(s,a) = 0, \quad j \in S, \\ &\qquad \sum_{s \in S} \sum_{a \in A_s} x(s,a) = 1, \\ &\qquad x(s,a) \geq 0, \quad a \in A_s,\ s \in S. \end{aligned}$$

- Note: one of constraints for $j \in S$ is redundant.

When we find a basic feasible solution $x$ for the dual LP, then $d_x(s)$ is deterministic. It either equals the action if $x(s,a) > 0$ and the state exists, otherwise it can be chosen arbitrary. So the feasible solution is also the policy.

# 7 Introduction to large scale discounted reward MDPs

In large-scale MDPs we assume that we have a very large state space, which can be infinitely large. This might lead to unbounded rewards, which is why a different approach is needed here. We can

claim for large scale discounted MDPs that under mild conditions, the optimal value and optimal stationary policy exist. The idea is that we derive these conditions.

## 7.1 Assumptions existence optimal value and optimal stationary policy

We need the following assumptions: Here, we must choose the function $w(s)$ accordingly, usually

1. $\exists\, \mu < \infty$:

$$\sup_{a \in A_s} |r(s,a)| \leq \mu w(s) \qquad \forall\, s \qquad \Leftrightarrow \|r_d\|_w \leq \mu$$

   - Suitable choice for $w$: $w(s) = \max\{\sup_{a \in A_s} |r(s,a)|, 1\}$

2. $\exists\, \kappa, 0 \leq \kappa < \infty$:

$$\sum_{j \in S} p(j|s,a) w(j) \leq \kappa w(s) \quad \forall\, a, \forall\, s \qquad \Leftrightarrow \|P_d\|_w \leq \kappa$$

3. $\forall\, \lambda, 0 \leq \lambda < 1, \exists\, \alpha, 0 \leq \alpha < 1$ and $\exists\, J \in \mathbb{N}$:

$$\lambda^J \sum_{j \in S} P_\pi^J(j|s) w(j) \leq \alpha w(s) \quad \forall\, \pi \in \Pi^{MD} \Leftrightarrow \|\lambda^J P_\pi^J\|_w \leq \alpha$$

- More easily verifiable (Prop. 6.10.5): (2) and (3) hold if there exists a constant $L > 0$ for which $\sum_{j \in S} p(j|s,a) w(j) \leq w(s) + L \quad \forall\, a, \forall\, s$.

this is somehow based on the context. Or it is given. If these conditions above hold, our value function is bounded from above by $\frac{\mu}{1-\alpha}[1+\lambda\kappa+\lambda^2\kappa^2+\ldots+(\lambda\kappa)^{J-1}]w(s)$. Also, if these conditions hold, then $L$ and $\mathcal{L}$ are $J$-stage contractions on $V_w$ (something with Banach spaces).

## 7.2 Finite-state approximations

What we want to do is to truncate the state space and approximate the value in order to reduce a large computational time for algorithms (downscale the problem). We know that when we use finite state approximation may result in converging monotonically to $v_\lambda^*$. Define $S_N = \{1, \ldots, N\}$, fix a value $u \in V_w$ and define

$$v^{N,u}(s) = \begin{cases} v(s), & s \leq N \\ u(s), & s > N \end{cases}$$

and the now operator

$$L_d^{N,u} v(s) = \begin{cases} r_d(s) + \lambda \sum_{j \leq N} p_d(j \mid s) v(j) + \lambda \sum_{j > N} p_d(j \mid s) u(j) \\ s \leq N \\ u(s), \quad s > N \end{cases}$$

We can deduce that for a fixed $N$ and policy for a MD policy that the operator on $v(s)$ is an $N$-stage contraction on $V_w$. This means that it has a unique fixed point in $V_w$, which results in $v_d^{N,u}(s)$ being an $N$-state approximation to the value with a stationary policy.

The conditions that were mentioned before could lead to the conclusion that this value converges monotonically from above to the optimal value as $N$ grows large.

# 8 Large scale average reward MDPs

To solve large scale average reward MDPs we can use the 'differential discounted reward' approach. Meaning, we will approximate the value as we do for discounted MDPs. Define

$$g = \lim_{\lambda \uparrow 1} (1-\lambda) V_\lambda^*(0)$$
$$h(s) = \lim_{\lambda \uparrow 1} (v_\lambda^*(s) - V_\lambda^*(0))$$

which results in the average reward optimality equation

$$h(s) = \max_{a \in A_s} \left\{ r(s,a) - g + \sum_{j \in S} p(j \mid s,a) h(j) \right\}$$

We can again set up a set of assumptions which will lead us to the existence of an optimal policy. These are Whenever these conditions hold the optimality equations mentioned before have a solu-

1. $\forall\, s \in S$:
$$-\infty < r(s,a) \leq R < \infty$$
2. $\forall\, s \in S$ and $0 \leq \lambda < 1$:
$$v_\lambda^*(s) > -\infty$$
3. $\exists\, K < \infty$ such that $\forall\, s \in S$ and $0 \leq \lambda < 1$:
$$h_\lambda(s) \equiv v_\lambda^*(s) - v_\lambda^*(0) \leq K$$
4. There exists a non-negative function $M(s)$ such that:
   - $M(s) < \infty$;
   - $\forall\, s \in S$ and $0 \leq \lambda < 1$: $v_\lambda^*(s) - v_\lambda^*(0) \geq -M(s)$; and
   - $\exists\, a_0 \in A_0$: $\sum_{j \in S} p(j|0, a_0) M(j) < \infty \Rightarrow$ there exists a lim inf average optimal stationary policy;

     alternatively (4'), $\forall (s,a)$: $\sum_{j \in S} p(j|s,a) M(j) < \infty \Rightarrow$ the optimality equation has a solution.

tion. Ofttimes condition (1) is easy to verify, but this is not the case for the others. In that case, we can check the following statements for a stationary policy such that:

- the derived Markov chain is positive recurrent (expected recurrence time is finite)
- $g^{d^\infty} > -\infty$
- The set of states for which the direct reward is larger that the gain mentioned above for some action is nonempty and finite

The points above are usually easier to verify.

# 9 Introduction to Approximate Dynamic Programming

First of all, when dealing with very large MDPs, we encounter the curses of dimensionality. These include dimensionality of the state space, the outcome space and the action space. Approximate dynamic programming deals with these three types of curses of dimensionality. The idea of ADP is that we step forward through time and try to approximate the value function, instead of analytically computing it.

## 9.1 Basic ADP algorithm

The general idea is that we perform several number of iterations. During each of these iterations we determine the best policy by solving the optimality equations. This is done for every time epoch of the MDP. When we arrive at the last epoch, we continue to the next iteration and the process repeats itself. The algorithm can be summarized as follows:

1. Initialization. Initialize $\bar{V}_t^0(S_t)\ \forall\ S_t$, choose an initial state $S_0^1$, set $n = 1$.
2. Sampling. Choose a sample path $\omega^n$.
3. Iteration. For $t = 0, 1, 2, \ldots, T$ do:
   3a. Solve

$$\hat{v}_t^n = \max_{a_t \in A_t^n} \left( C_t(S_t^n, a_t) + \gamma \sum_{s' \in S} \mathbb{P}(s'|S_t^n, a_t) \bar{V}_{t+1}^{n-1}(s') \right),$$

   and let $a_t^n$ be the value of $a_t$ that solves the maximization problem.
   3b. Update $\bar{V}_t^{n-1}(S_t)$ using

$$\bar{V}_t^n(S_t) = \begin{cases} \hat{v}_t^n, & S_t = S_t^n, \\ \bar{V}_t^{n-1}(S_t), & \text{otherwise.} \end{cases}$$

   3c. Compute $S_{t+1}^n = S^M(S_t^n, a_t^n, W_{t+1}(\omega^n))$.
4. Increment or stop. Let $n = n + 1$. If $n < N$, go to step 2.

## 9.2 Q-learning

Q-learning is one of the ADP algorithms that is extended. It is a strategy for problems with small state and action spaces, where there is no mathematical model of the transition function. We consider $Q(s, a)$ to be the value of being in state $s$ after taking action $a$ and $\bar{Q}^n(s, a)$ the statistical estimate of $Q(s, a)$ after $n$ iterations.

In this case we choose an action $a^n$ by determining the maximum of $\bar{Q}^{n-1}(S^n, a)$. The updated value of being in state $S^n$ and taking action $a^n$ is

$$\hat{q}^n = \hat{C}\left(S^n, a^n\right) + \gamma \max_{a' \in A} \bar{Q}^{n-1}\left(S^{n+1}, a'\right)$$

We update the $Q$-factors, namely the statistical estimates, as

$$\bar{Q}^n\left(S^n, a^n\right) = \left(1 - \alpha_{n-1}\right) \bar{Q}^{n-1}\left(S^n, a^n\right) + \alpha_{n-1} \hat{q}^n$$

The value of that iteration in state $s$ equals the maximum of the $Q$-factors.

One of the issues that arise in $Q$-learning is that $Q$-factors might underestimate the value of a state-action pair. Therefore the algorithm might not choose actions that take us to a certain state, errors are not corrected or we ignore actions that might be attractive. Therefore, we need to make a trade off between exploration or exploitation. In exploration we consider choosing an action at random, in exploitation we choose the action that currently seems optimal. We consider the sampling policy, which learns which action to take and the learning policy, which determines the action that appears best. Both lead to on-policy learning (sampling = learning) and off-policy learning (sampling $\neq$ learning).

## 9.3 Approximate value iteration

Approximate value iteration can be used to estimate the values without computing the one-step transition matrix. In this algorithm we generate a sample for the exogenous information and

determine the probability of that outcome. Afterwards, we approximate the expectation of the transition state with

$$\mathbb{E}\bar{V}^{n-1}\left(S^M\left(S^n,a,W\right)\right) \approx \sum_{\omega\in\hat{\Omega}^n} p^n(\omega)\bar{V}^{n-1}\left(S^M\left(S^n,a,W(\omega)\right)\right)$$

and use this to estimate the value of being in state $S^n$ with

$$\hat{v}^n = \max_{a\in A}\left(C\left(S^n,a\right)+\gamma\sum_{\omega\in\hat{\Omega}^n} p^n(\omega)\bar{V}^{n-1}\left(S^M\left(S^n,a,W(\omega)\right)\right)\right)$$

Lastly we update our estimate for the state as $\bar{V}^n\left(S^n\right) = \left(1-\alpha_{n-1}\right)\bar{V}^{n-1}\left(S^n\right)+\alpha_{n-1}\hat{v}^n$.

The approximate value iteration algorithm works as follows

1. Initialization. Initialize $\bar{V}_t^0(S_t)\,\forall\, S_t$, choose an initial state $S_0^1$, set $n=1$.
2. Sampling. Choose a random sample of outcomes $\hat{\Omega}^n \subset \Omega$.
3. Iteration. For $t=0,1,2,\ldots,T$ do:
   3a. Solve
   $$\hat{v}_t^n = \max_{a_t\in A_t^n}\left(C_t(S_t^n,a_t)+\gamma\sum_{\hat{\omega}\in\hat{\Omega}^n} p^n(\hat{\omega})\bar{V}_{t+1}^{n-1}\left(S^M(S_t^n,a_t,W_{t+1}(\hat{\omega}))\right)\right),$$
   and let $a_t^n$ be the value of $a_t$ that solves the maximization problem.
   3b. Update $\bar{V}_t^{n-1}(S_t)$ using
   $$\bar{V}_t^n(S_t) = \begin{cases}(1-\alpha_{n-1})\bar{V}_t^{n-1}(S_t^n)+\alpha_{n-1}\hat{v}_t^n, & S_t = S_t^n,\\ \bar{V}_t^{n-1}(S_t), & \text{otherwise.}\end{cases}$$
   3c. Compute $S_{t+1}^n = S^M(S_t^n,a_t^n,W_{t+1}(\omega^n))$.
4. Increment or stop. Let $n=n+1$. If $n<N$, go to step 2.

## 9.4 Post-decision state variable

This is the state of a system after we made a decision, but before any new information arrives. Whenever computing the estimate of the value of the transition function $S_{t+1}$ is easy we use the pre-decision state variable, otherwise we opt for the post-decision state variable. Consider the original transition function $S_{t+1} = S^M(S_t,a_t,W_{t+1})$. We can break it into two steps, which gives us the post-decision state variable:

$$\begin{aligned}S_t^a &= S^{M,a}(S_t,a_t)\\ S_{t+1} &= S^{M,W}(S_t^a,W_{t+1})\end{aligned}$$

We can use these post-decision state variable in order to rewrite the optimality equations. Consider $V_t(S_t)$ to be the value of being in state $S_t$ just before the decision and $V_t^a(S_t^a)$ the value of being in state $S_t^a$ immediately after the decision. We can set up the following equations for the value estimates:

$$\begin{aligned}V_{t-1}^a\left(S_{t-1}^a\right) &= \mathbb{E}\left[V_t\left(S_t\right)\mid S_{t-1}^a\right],\\ V_t\left(S_t\right) &= \max_{a_t\in A_t}\left(C_t\left(S_t,a_t\right)+\gamma V_t^a\left(S_t^a\right)\right),\\ V_t^a\left(S_t^a\right) &= \mathbb{E}\left[V_{t+1}\left(S_{t+1}\right)\mid S_t^a\right].\end{aligned}$$

Substituting the third equation into the second gives us the Bellman equation where we define the value at state $S_t$. Substituting the second equation into the first gives us the optimality equations

around the post decision variable. The benefit of doing this is that we compute the conditional expectation over the whole max-function instead of it appearing in the max-function itself. This eases computation.

We can also set up an ADP algorithm with a post-decision state variable, where the value estimate is updated using smoothing only. We get

1. Initialization. Initialize $\bar{V}_t^0, t \in T$, choose an initial state $S_0^1$, set $n = 1$.
2. Sampling. Choose a sample path $\omega^n$.
3. Iteration. For $t = 0, 1, 2, \ldots, T$ do:
   3a. Solve
   $$\hat{v}_t^n = \max_{a_t \in \mathcal{A}_t^n} \left( C_t(S_t^n, a_t) + \bar{V}_t^{n-1}(S^{M,a}(S_t^n, a_t)) \right),$$
   and let $a_t^n$ be the value of $a_t$ that solves the maximization problem.
   3b. If $t > 0$, update $\bar{V}_{t-1}^{n-1}$ using
   $$\bar{V}_{t-1}^n(S_{t-1}^{a,n}) = (1 - \alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{a,n}) + \alpha_{n-1}\hat{v}_t^n$$
   3c. Find the post-decision state $S_t^{a,n} = S^{M,a}(S_t^n, a_t^n)$
   and the next pre-decision state $S_{t+1}^n = S^M(S_t^n, a_t^n, W_{t+1}(\omega^n))$.
4. Increment or stop. Let $n = n + 1$. If $n \leq N$, go to step 2.
5. Result. Return the value functions $(\bar{V}_t^N)_{t=0}^T$.

# 10 Approximate Dynamic Programming

## 10.1 Value function approximations

## 10.2 Temporal-difference learning