# Markov Decision Process

*MTH211 - Probability and Statistics*


Abdallah Amr - 120200011

Mostafa Osama - 120200018

Ahmed Abdelkader - 120200028

Ahmad Mongy - 120200033

Mohamed Elsayed - 120200096

Mohanned Ahmed - 120200113

Gana Elsaid - 120200224


Submitted To:

*Prof. Sherif I. Rabia*

# Table of Contents

# I. Introduction:

Markov decision processes (*MDP*) give us a way to formalize sequential decision making. This way is the basis of knowledge for solving the problems with reinforcement learning. In addition, *MDP* give a way to treat an environment rich with randomness. In Markov decision processes, there is an item has to make a decision called agent and interacts with the environment that placed in. These interactions occur sequentially overtime and at each timestep that agent will get some representation of the environment state. Then, that agent selects an action to take according to this representation. The environment is then transitioned into new state and the agent is given a reward because of its previous action. So, the components of an *MDP* are consisting of the environment, the agent, all the possible states of the environment, all the actions that the agent can take in the environment, and all the rewards that the agent can receive from taking actions in the environment. This process of selecting an action from a given state transitioning to a new state and receiving a reward happening sequentially over and over again which creates something called a trajectory (episode) that shows the sequence of states actions and rewards. Throughout the process, it's the agent's goal to maximize the total amount of rewards that it receives from taking actions and given states of the environment. This means that the agent wants to maximize not just the immediate reward but the cumulative rewards that it will receive overtime.

# II. General concepts:

**Environment & Agent:** The agent and environment are complementary to each other. The agent is an item which will make an action in the environment. The environment is a place that an agent locates. The agent will interact with the environment depending on the state of environment locating in.

**State (S):** The state describes the situation of the environment at each time step which the agent will take an action depending on it. $S = \{s_1, s_2, s_3, s_4 \dots s_t\}$ is a set of the environment states. At $t = 1,2,3,4,5 \dots, S_t \in S$ is a possible state to the environment.

**Action (A):** The action describes the decision taken by the agent or the interaction of the agent to the environment depending on its state. $A = \{a_1, a_2, a_3, a_4 \dots a_t\}$ is a set of actions to the agent. At $t = 1,2,3,4,5 \dots, A_t \in A$ is a possible action to the agent.

The state and action spaces may be finite or infinite, for example the set of real numbers. Some processes with countably infinite state and action spaces can be reduced to ones with finite state and action spaces.

**Reward (R):** It is the environment reaction to the reaction of agent which we can figure out the validity of the actions in each state.
$R = \{r_1, r_2, r_3, \dots r_t\}$ is a set of rewards to the environment. At state $S_t$ and action $A_t$ the reward will be $R_{t+1}$.

$$R(S_t, A_t) = R_{t+1}$$

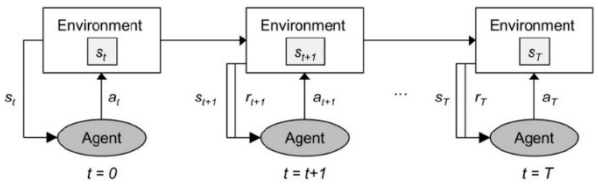


*Figure 1: Markov Decision Process flow between agent and environment, adapted from Duarte, Fernando & Lau, Nuno & Pereira, Artur & Reis, Luís. (2020). A Survey of Planning and Learning in Games. Applied Sciences.*

**Transition probability function (P):** Since all the three sets are finite $S, R, A$. we can calculate their probability. This equation describes the probability of a state and a reward at the same time given the preceding state and action. $[s', s \in S, r \in R, a \in A]$

$$P(s', r \mid s, a) = P(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)$$

**Discount factor(γ):** It is the measure of the importance of future rewards to the current state. Its value ranges between 0 and 1. $\gamma = 0$ represents an immediate reward. $\gamma = 1$ represents a farsighted thinking about the future reward. The intuition behind using a discount is that there is no certainty about the future rewards.

**Policy (π):** It is a plan with actions that agent will do to maximize its reward (the expected value). It can be stochastic policy (in a given state, the agent can choose different actions) $\pi(s, a) = P(a_t = a | s_t = s)$ or deterministic policy (in each state the agent choses a unique action). $\pi(s) = a$

A policy consists of a decision function for each decision variable. A **decision** function for a decision variable is a function that specifies a value for the decision variable for each assignment of values of its parents. Thus, a policy specifies what the agent will do for each possible value that it could sense.
A **stationary policy** is a function $\pi: S \rightarrow A$. That is, it assigns an action to each state. Given a reward criterion, a policy has an expected value for every state. Let $V^{\pi}(s)$ be the expected value of following π in state $S$. This specifies how much value the agent expects to receive from following the policy in that state. Policy $\pi$ is an **optimal policy** if there is no policy $\pi'$ and no state $s$ such that $V^{\pi`}(s) > V^{\pi}(s)$. That is, it is a policy that has a greater or equal expected value at every state than any other policy.

# III. Sequential Decisions

Generally, agents do not make decisions in the dark without observing something about the world, nor do they make just a single decision. A more typical scenario is that the agent makes an observation, decides on an action, carries out that action, makes observations in the resulting world, then makes another decision conditioned on the observations, and so on. Subsequent actions can depend on what is observed, and what is observed can depend on previous actions. In this scenario, it is often the case that the sole reason for carrying out an action is to provide information for future actions.
A **sequential decision problem** is a sequence of decisions, where for each decision you should consider

- what actions are available to the agent.
- what information is, or will be, available to the agent when it has to act.
- the effects of the actions
- the desirability of these effects.

## a. Markov Decision Process:

### 1. Markov Property:

$$P[S_{t+1} | S_t] = P[S_{t+1} | S_1, \dots S_t]$$

According to the Markov Property, the current state of the agent depends only on its immediate previous state (or the previous timestep). It doesn't depend on the history of it states. Similarly, its next state depends only on its current state. Formally, for a state s to be Markov, the probability of the next state $s$` should only be dependent on the current state $s$, and not on the rest of the past states $s_1, s_2$ …

2. Markov Chain (process) and Markov Decision Process (MDP):

$$P_{ss'} = P[S_{t+1} = s' \mid S_t = s]$$

A Markov Process is defined by $(S, P)$ where $S$ are the states, and $P$ is the state-transition probability. It consists of a sequence of random states $S_1$, $S_2$, … where all the states obey the Markov Property.

The state transition probability or $Pss'$ is the probability of jumping to a state s` from the current state $s$.

$$P^a_{ss'} = P[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$R^a_s = E[R_{t+1} = s' | S_t = s, A_t = a]$$

For MDP, it is defined by $(S, A, P, R, \gamma)$, where $A$ is the set of actions, $R_s$ is the reward, and $\gamma$ is the discount factor. It is essentially a Markov chain with actions leading to certain reward. The state reward $R_s$ is the expected reward over all the possible states that one can transition to from state $s$. This reward is received for being at the state $S_t$. By convention, it is said to be received after the agent leaves the state and hence, regarded as $R_{t+1}$.

Actions give more control over the Markov process. Previously, state transition probability and state rewards were more or less stochastic (random). However, now the rewards and the next state also depend on what action the agent picks. Basically, the agent can now control its own fate.

# IV. Decision Processes

Often an agent must reason about an ongoing process or it does not know how many actions it will be required to do. These are called **infinite horizon** problems when the process may go on forever or **indefinite horizon** problems when the agent will eventually stop, but where it does not know when it will stop. To model these situations, we augment the Markov chain with actions. At each stage, the agent decides which action to perform; the resulting state depends on both the previous state and the action performed. For ongoing processes, you do not want to consider only the utility at the end, because the agent may never get to the end. Instead, an agent can receive a sequence of **rewards**. These rewards incorporate the action costs in addition to any prizes or penalties that may be awarded. Negative rewards are called **punishments.** Indefinite horizon problems can be modeled using a stopping state. A **stopping state** or **absorbing state** is a state in which all actions have no effect; that is, when the agent is in that state, all actions immediately return to that state with a zero reward. Goal achievement can be modeled by having a reward for entering such a stopping state.

To decide what to do, the agent compares different sequences of rewards. The most common way to do this is to convert a sequence of rewards into a number called the **value** or the **cumulative reward**. To do this, the agent combines an immediate reward with other rewards in the future.

There are three common ways to combine rewards into a value $V$:

**Total Reward**: $V = \sum_{i=1}^{\infty} r_i$. In this case, the value is the sum of all of the rewards. This works when you can guarantee that the sum is finite; but if the sum is infinite, it does not give any opportunity to compare which sequence of rewards is preferable.

**Average Reward:** $V = lim_{n\to\infty}(r1 + \cdots + rn)/n.$ In this case, the agent's value is the average of its rewards, averaged over for each time period. As long as the rewards are finite, this value will also be finite. However, whenever the total reward is finite, the average reward is zero, and so the average reward will fail to allow the agent to choose among different actions that each have a zero average reward.
**Discounted Reward**: $V = r1 + \gamma r2 + \gamma 2\, r3 + \cdots + \gamma i - 1\, ri + \cdots,$ where $\gamma$, the **discount factor**, is a number in the range $0 \leq \gamma < 1$. Under this criterion, future rewards are worth less than the current reward. If γ was 1, this would be the same as the total reward. When $\gamma = 0$, the agent ignores all future rewards. Having $0 \leq \gamma < 1$ guarantees that, whenever the rewards are finite, the total value will also be finite.

# V. Policies

A **policy** specifies what the agent should do under all contingencies. An agent wants to find an optimal policy - one that maximizes its expected utility.
A policy consists of a decision function for each decision variable. A **decision function** for a decision variable is a function that specifies a value for the decision variable for each assignment of values of its parents. Thus, a policy specifies what the agent will do for each possible value that it could sense.

A **stationary policy** is a function $\pi: S \to A$. That is, it assigns an action to each state. Given a reward criterion, a policy has an expected value for every state. Let $V^{\pi}(s)$ be the expected value of following $\pi$ in state $s$. This specifies how much value the agent expects to receive from following the policy in that state. Policy $\pi$ is an **optimal policy** if there is no policy $\pi'$ and no state $s$ such that $V^{\pi`}(s) > V^{\pi}(s)$. That is, it is a policy that has a greater or equal expected value at every state than any other policy.

# VI. Value of a Policy

Consider how to compute the expected value, using the discounted reward of a policy, given a discount factor of $\gamma$. The value is defined in terms of two interrelated functions:
$V^{\pi}(s)$ is the expected value of following policy $\pi$ in state $s$.
$Q^{\pi}(s, a)$, is the expected value, starting in state $s$ of doing action $a$, then following policy $\pi$. This is called the ***Q*-value** of policy $\pi$.

$Q^{\pi}$ and $V^{\pi}$ are defined recursively in terms of each other. If the agent is in state $\boldsymbol{s}$, performs action a, and arrives in state $s`$, it gets the immediate reward of $R(s, a, s`)$ plus the discounted future reward, $\gamma V^{\pi}(s`)$ . When the agent is planning it does not know the actual resulting state, so it uses the expected value, averaged over the possible resulting states:

$$Q^{\pi}(s, a) = \sum_{s`} P(s`|s, a)\big(R(s, a, s`) + \gamma V^{\pi}(s`)\big)$$

$$= R(s,a) + \gamma \sum_{s`} P(s`|s,a)\, V^{\pi}(s`)$$

*Equation 1*

**Where** $R(s,a) = \sum_{s`} P(s`|s,a)\, R(s,a,s`)$.

$V^{\pi}(s)$ is obtained by doing the action specified by $\pi$ and then following $\pi$:

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s))$$

# VII. Value of an Optimal Policy

Considering actions as the parameter for state transition. So, it is necessary to evaluate actions along with states. For this, we define action value functions that essentially give us the expected Return over actions. It specifies how good it is for an agent to perform a particular action in a state with a policy $\pi$.

Formulating Bellman equation for state and action value functions, we get:

$$V^{\pi}(s) = P^{\pi}[\, R_{t+1} + \gamma . v_{\pi}(S_{t+1}) | S_t = s]$$

$$Q^{\pi}(s) = P^{\pi}[\, R_{t+1} + \gamma . Q^{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

Note that we represent the states using circles and actions using dots; both the diagrams (Figure 2) are a different level view of the same MDP, up being the 'state-centric' view and down being the 'action-centric' view.

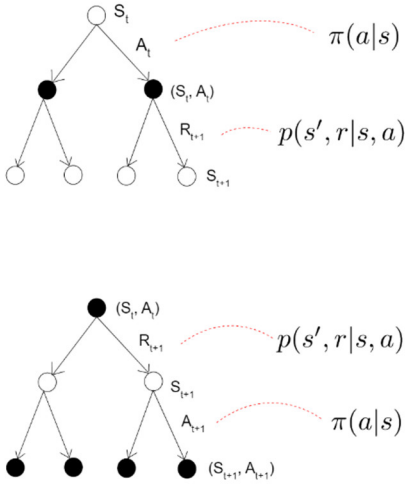


*Figure 2: Intuition on Bellman Equation*

Circle to dot: The agent is in a state $s$; it picks an action $a$ according to the policy. So, in this part of the transition, the agent picks an action. This part is completely controllable by the agent as it gets to pick the action.

Dot to Circle: The environment acts on the agent and sends it to a state based on the transition probability. After both, we call it a complete state transition. This part is not controllable by the agent as it cannot control how the environment acts.

we treat these as two individual mini transitions, since we have a state to action transition, we take the expected action value over all the actions and same for the action value function.

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s)\, Q^{\pi}(s,a)$$

$$Q^{\pi}(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \; V^{\pi}(s')$$

Substituting both equations:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s)\, (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \; V^{\pi}(s'))$$

$$Q^{\pi}(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \, (\sum_{a' \in A} \pi(a'|s')\, Q^{\pi}(s',a'))$$

Gives us the solution for the bellman equation for state and action value function to define an MDP and use it to implement our search algorithm for the optimum policy such as Value iteration, Q-learning and SARSA.

# VIII. Solutions Algorithms to Markov Decision Process:

## a. Policy iteration:

**Policy iteration** starts with a policy and iteratively improves it. It starts with an arbitrary policy $\pi$ (an approximation to the optimal policy works best) and carries out the following steps starting from $i = 0$:

1. Policy evaluation: determine $V^{\pi_i}(s)$. The definition of $V^{\pi}$ is a set of $|S|$ linear equations in $|S|$ unknowns. The unknowns are the values of $V^{\pi_i}(s)$. There is an equation for each state. These equations can be solved by a linear equation solution method (such as Gaussian elimination) or they can be solved iteratively.
2. Policy improvement: choose $\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$, where the Q-value can be obtained from V using Equation 1. To detect when the algorithm has converged, it should only change the policy if the new action for some state improves the expected value; that is, it should set $\pi_{i+1}(s)$ to be $\pi_i(s)$ if $\pi_i(s)$ is one of the actions that maximizes $Q^{\pi_i}(s,a)$.
3. Stop if there is no change in the policy, if $\pi_{i+1} = \pi_i$ , otherwise increment i and repeat.

```
1: procedure Policy_iteration(S, A, P, R)
2:     Inputs
3:         S is the set of all states
4:         A is the set of all actions
5:         P is state transition function specifying P(s' | s, a)
6:         R is a reward function R(s, a)
7:     Output
8:         optimal policy π
9:     Local
10:        action array π[S]
11:        Boolean variable noChange
12:        real array V[S]
13:    set π arbitrarily
14:    repeat
15:        noChange := true
16:        Solve V[s] = R(s, a) + γ * Σ_{s'∈S} P(s' | s, π[s]) * V[s']
17:        for each s ∈ S do
18:            QBest := V[s]
19:            for each a ∈ A do
20:                Qsa := R(s, a) + γ * Σ_{s'∈S} P(s' | s, a) * V[s']
21:                if Qsa > QBest then
22:                    π[s] := a
23:                    QBest := Qsa
24:                    noChange := false
25:    until noChange
26:    return π
```

*Figure 3: Pseudocode Algorithm for optimal policy derivation (Policy iteration for MDPs), adapted from artint.info, ARTIFICIAL INTELLIGENCE 2E, FOUNDATIONS OF COMPUTATIONAL AGENTS*

## b. Value iteration:

Value iteration is a method of computing an optimal policy for an MDP and its value.

Value iteration starts at the "end" and then works backward, refining an estimate of either $Q^*$ or $V^*$. There is really no end, so it uses an arbitrary end point. Let $V_k$ be the value function assuming there are k stages to go and let $Q_k$ be the Q-function assuming there are k stages to go. These can be defined recursively. Value iteration starts with an arbitrary function $V_0$.For subsequent stages, it uses the following equations to get the functions for $K + 1$ stages to go from the functions for k stages to go

```
1: procedure Value_iteration(S, A, P, R)
2:     Inputs
3:         S is the set of all states
4:         A is the set of all actions
5:         P is state transition function specifying P(s' | s, a)
6:         R is a reward function R(s, a)
7:     Output
8:         π[S] approximately optimal policy
9:         V[S] value function
10:    Local
11:        real array V_k[S] is a sequence of value functions
12:        action array π[S]
13:    assign V_0[S] arbitrarily
14:    k := 0
15:    repeat
16:        k := k + 1
17:        for each state s do
18:            V_k[s] = max_a R(s, a) + γ * Σ_{s'} P(s' | s, a) * V_{k-1}[s']
19:    until termination
20:    for each state s do
21:        π[s] = arg max_a R(s, a) + γ * Σ_{s'} P(s' | s, a) * V_k[s']
22:    return π, V_k
```

*Figure 4 Pseudocode Algorithm for optimal policy derivation (Value iteration for MDPs), storing V. Adapted from artint.info, ARTIFICIAL INTELLIGENCE 2E, FOUNDATIONS OF COMPUTATIONAL AGENTS*

$$Q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s`} P(s`|s,a)\, V_k(s`)$$

$$V_k = max_a\, Q_k(s,a)$$

It can either save the $V[S]$ array or the $Q[S,A]$ array. Saving the V array results in less storage, but it is more difficult to determine an optimal action, and one more iteration is needed to determine which action results in the greatest value. Figure 2 shows the value iteration algorithm when the $V$ array is stored. This procedure converges no matter what the initial value function $V_0$ is. An initial value function that approximates $V^*$ converges quicker than one that does not. The basis for many abstraction techniques for MDPs is to use some heuristic method approximate $V^*$and to use this as an initial seed for value iteration. The value iteration algorithm of Figure 2 has an array for each stage, but it really only needs to store the current and the previous arrays. It can update one array based on values from the other. A common refinement of this algorithm is asynchronous value iteration. Rather than sweeping through the states to create a new value function, asynchronous value iteration updates the states one at a time, in any order, and stores the values in a single array. Asynchronous value iteration can store either store the V[S] array or the Q[S, A] array.

## c. Value Iteration in practice:

For illustration of a value iteration method, we are going to observe a game called Gridworld, adapted from:
https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html
Gridworld is a toy environment that is often used as a toy model in the Reinforcement Learning literature.

QR code for the URL

In this particular case:

- State space: GridWorld has 10x10 = 100 distinct states. The start state is the top left cell. The gray cells are walls and cannot be moved to.
- Actions: The agent can choose from up to 4 actions to move around. In this example
- Environment Dynamics: GridWorld is deterministic, leading to the same new state given each state and action
- Rewards: The agent receives +1 reward when it is in the center square (the one that shows R 1.0), and -1 reward in a few states (R -1.0 is shown for these). The state with +1.0 reward is the goal state and resets the agent back to start. The rewards can be updated as desired. This is a deterministic, finite Markov Decision Process (MDP) and as always, the goal is

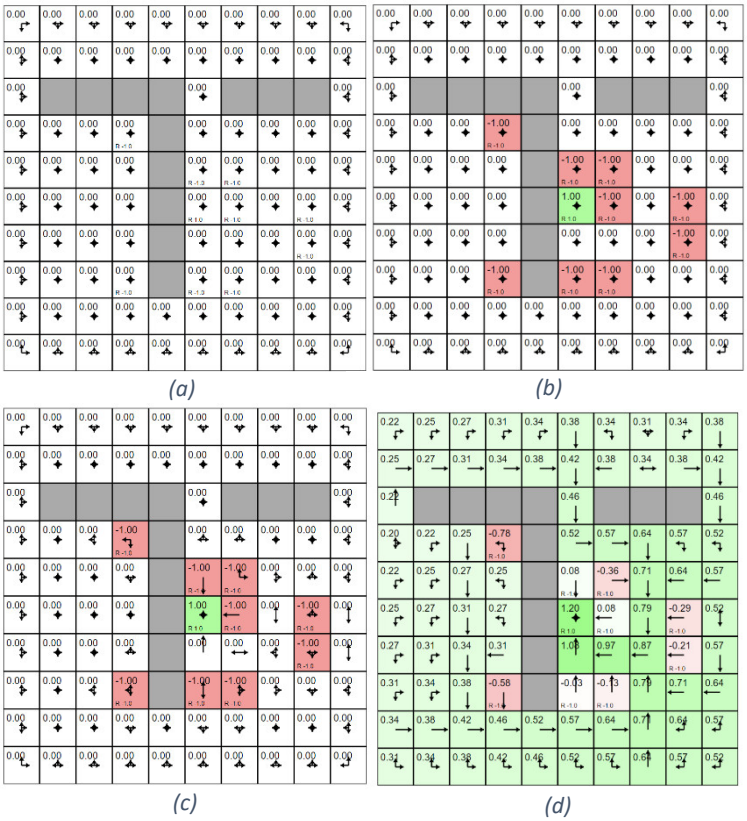


*Figure 5 (a) Initial state of the environment, (b) After policy evaluation (one sweep), (c) After Policy update based on last evaluation, (d) After running value iteration*

to find an agent policy (shown here by arrows) that maximizes the future discounted reward.

**As shown in figure 4 (a)Initial state of Interface**. The color of the cells (initially all white) shows the current estimate of the Value (discounted reward) of that state, with the current policy.

## d. Q-Learning:

Another solution algorithm is Q-learning to find the optimum policy for an agent to follow and take specific actions given its current state. Q-learning is an off-policy algorithm meaning that it learns from actions outside the current policy i.e., taking random actions to explore the environment, meaning that it does not need a policy to follow, it just finds the policy that maximizes its reward.

For a state $s$ we can say that the optimal policy $\pi*$ to follow is the action with the most expected return, from this we can implement that:

$$V^*(s) = max\ Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s' \in S} P^a_{ss'}\,(R^a_s + \gamma V^*(s'))$$

From the value iteration concept, we can also iterate for the Q-value:

$$Q^*(s,a) = \sum_{s' \in S} P^a_{ss'}\,(R^a_s + \gamma . \max Q^*(s',a'))$$

For some problems, we may have no access for the transition probability function, or we are not interested in it, so we can approximate this function to make it even simple to implement. Now we can have an equation for the Q-value:

$$Q_{new}(s,a) = Q_{old}(s,a) + a(R^a_s + \gamma . \max Q(s',a') - Q_{old}(s,a))$$

Where $\alpha$ [0,1] is the learning rate, determines to how much of new information overrides old information. $\alpha$ = 0 make the agent learn nothing new, optimal for stochastic problems (not zero but near to it). $\alpha$ = 1 make the agent considers the acquired information, ignoring the old ones, suitable for fully deterministic problems.

## e. Sample problem to solve and explain algorithm:

Frozen Lake problem: The agent controls the movement of a character in a grid world. Some tiles of the grid are walkable (F), and others lead to the agent falling into the water (H). Additionally, the movement direction of the agent is uncertain and only partially depends on the chosen direction (slippery). The agent is rewarded for finding a walkable path to a goal tile (G).

Q-table(table1): a zero matrix with each row represents a state and each column represents an action, for each value in the table $Q[s,a]$ identify the expected return from doing this action given you are in this state.

| | A0 | A1 | … | An |
|---|---|---|---|---|
| S0 | Q(S0, A0) | … | | |
| S1 | Q(S1, A0) | | | |
| … | … | | | |
| Sn | | | | Q(S0, A0) |

*Table 1: Sample Q-Table*

For Frozen Lake, we have a Q-table with no. of rows = 64 (states), no. of columns = 4 (actions).

The new Q-value for state $s_0$, taken action $a_0$, is equal to the old Q-value for same state and action, adding the learning rated term of the reward and the maximum reward that can be obtained from state $s_1$ (new state after action $a_0$).

### f. Exploitation vs Exploration (ε -greedy):

An agent interacts with the environment in 2 ways. First exploiting, using the q-table as a reference and view all possible actions for a given state. The agent then selects the action based on the max value of those actions. Second way is to act randomly (exploring). Instead of selecting actions based on the max future reward we select an action at random. Acting randomly allows the agent to explore and discover new states that may have higher reward.

You can balance exploration/exploitation using epsilon (ε) and setting the value of it by making it move towards exploitation as you proceed in training. In other words, agent will act randomly at first, then starts to use the q-table.

By implementing all these steps on the frozen lake, we made this code that solve the frozen lake problem with Q-learning, depending on Markov decision process. Implementation for solution algorithm to solve Frozen Lake using Q-Learning: https://colab.research.google.com/drive/1r2NKey5XcEqY5tMuLLM3FbBPWNoYVwVp#scrollTo=Z94vVIY_c-xM

## IX. Basic Elements of an MDP model:

MDP can be considered to be a way of modeling problems by automating the decision-making process in uncertain environments by following a simple algorithm that tries to maximize the reward of the system (environment) to help find the optimal policy (the best action to be taken while interacting with the environment and changing the current state to a new state).

MDP has four main components: state space, action space, the transition probability matrices for all actions (which resembles the effect of actions), and optimal policy formulation (which is found by the immediate value of the actions).

For any MDP model, there are possible states from which to start and go based on the actions taken from a particular state; every state has some policies that can be taken, generating random paths with different utilities (the reward of taking that policy from a particular path). The main problem of the paper had been formulated to be an MDP model to obtain the optimal policy for the SU (secondary user) while selecting a random channel.

### a. State-space:

The proposed problem had two states: the number of energy units in the energy storage (e) and the number of data buffered in the data queue (q), each state is a two-parameters variable; S stands for the state.

### b. Action-space:

For every action taken in an MDP model, it has a specific value (the expected utility after taking that action) for it, so the optimal policy will depend on getting the values of taking random actions with getting the action with the maximum value to be the optimal policy for the MDP model.

The SU has three possible actions to take based on the state of the selected random channel (idle or busy); the possible actions for the SU to take are performing active transmission if the channel was found to be idle or performing one of energy harvesting or backscattering data if the channel was found to be busy.

The action space can be divided into two subspaces; the subspaces will be two different actions to be taken while finding the channel either busy or idle after sensing it; the first subspace of actions will be performing backscattering data when the sensed channel is busy and active transmission when the channel is idle; the second subspace of actions will be performing harvesting energy when the sensed channel is busy and active transmission when the channel is idle.

## c. Transition probability matrices:

Transition probability matrices take the agent of the MDP model from one state to the other; it has a vital role in calculating the value of any action as it is used in the equation of calculating the utility of any action, using the value iteration algorithm.

This problem had two subspaces of actions, so a transition probability matrix was driven for each one of them.

A transition probability matrix was driven and divided into three main parts according to the transitions of the queue state.

The matrix tells that when q is less than some specific value R, the active transmission cannot occur as there will be not enough data in the data queue; then, a new matrix was defined to describe the energy state transition, which is undoubtedly independent of the queue state transitions.

The SU will not perform the active transition mode as long as $q < R$, so the energy will never decrease, but it can be increased by one unit due to the energy harvesting mode, which can be successful sometimes and unsuccessful for other times due to the missed detection; if e = E; then, this denotes that the energy storage is complete, and so the energy state will remain the same.

The second random action will be taken when the $R < q < Q$ (size of the data queue) – R; then, three different possible scenarios can happen: q can increase by a number from 1 to R, decreases by some number from R – i to 0 or remains the same; for the first case, the energy harvesting can be successful in the case of the channel is busy with no missed detection, and the energy state increases by one unit; for the second case, the active transmission happens successfully, and this happens when the channel is idle with no false alarm, and there are sufficient energy units in the energy storage, so when the active transmission happens, the energy state will decrease by some W units; for the third case, no batch arrival occurs with success backscatter transmission, and the data queue decreases by one packet.

These were almost all the actions that can be taken based on the current state of the SU; with such an analysis of the system, it can be modeled as an MDP problem to help obtain the optimal policy for the SU of when to harvest energy and perform active transmission or backscattering data from the PU's signal without interfering and performing active transmission, getting the maximum reward of the proposed hybrid model.

## d. Optimal policy:

After describing all the main components of the MDP system, an optimal policy can be found by calculating the values of taking any action from any state; then, the action with the maximum value (expected utility) will be the optimal policy to solve the problem.

In this problem, it is needed to find the optimal time to use any of the two subspaces of actions; in fact, the problem is formulated as an MDP system to find when to use which one of the channel-mode pair of actions to maximize the reward (choosing the proper action in the suitable time) of the SU over the long run.

There are different ways to obtain the optimal policy: The value iteration algorithm, the policy iteration, or solving LP (linear programming) problem.

The value iteration algorithm was used to obtain the optimal policy for solving this problem; in this algorithm, the system starts with assuming that all the values of the states are zero and then using the equation of calculating the utility in the value iteration algorithm, updating the value of each state for each iteration till it converges to some fixed value; this can be deduced by comparing each new value with the previous value by checking if

$|V_{t+1} - V_t| < \varepsilon$ for ε close to zero, so it indicates that the value of each state is converging, so an optimal policy can be found by choosing the action with the maximum value.

## e. Conclusion:

As proven in the previous problem of the paper, any problem can be formulated as an MDP model to obtain the optimal policy for solving the problem maximizing the reward; the four main components of the MDP model should be identified first, as done in the previous paper; the set of states should be defined; then, the set of actions that can be taken in any different state should be identified; after that, to move from one state to the other, it is a random process, so a transition probability matrix should be derived, and with the utility of the value iteration algorithm the utilities of each action taken from a particular state can be calculated to identify the optimal policy for the problem.

# X. References:


1- David L. Poole and Alan K. Mackworth. 2017. Artificial Intelligence: Foundations of Computational Agents (2nd. ed.). Cambridge University Press, USA. http://www.cambridge.org/9781107195394

2- Sutton, R.S. & Barto, A.G., 2018. Reinforcement learning: An introduction, MIT press. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

3- Prof. John Tsitsiklis, Probabilistic Systems Analysis and Applied Probability, video lectures 16-18 Markov Chains I,II,III. MIT OpenCourseWare. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-041-probabilistic-systems-analysis-and-applied-probability-fall-2010

4- Andrej Karpathy, GridWorld: Dynamic Programming Demo. https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

5- Dimitri P. Bertsekas Dynamic Programming and Optimal Control 3rd Edition, Volume II

6- Paul A. Gagniuc, Markov Chains, From Theory to Implementation and Experimentation. John Wiley & Sons (2017)

7-Jagtap, R. Understanding Markov Decision Process (MDP) - Towards Data Science. Medium. https://towardsdatascience.com/understanding-the-markov-decision-process-mdp-8f838510f150

8-Lambert, N. Fundamental Iterative Methods of Reinforcement Learning. Medium. https://towardsdatascience.com/fundamental-iterative-methods-of-reinforcement-learning-df8ff078652a

9-Salloum, Z. Basics of Reinforcement Learning, the Easy Way - Ziad SALLOUM. Medium. https://zsalloum.medium.com/basics-of-reinforcement-learning-the-easy-way-fb3a0a44f30e#:%7E:text=Discount%20factor%20is%20a%20value,importance%20to%20the%20current%20state

10-Stanford Online [stanfordonline]. Lecture 8: Markov Decision Processes - Reinforcement Learning | Stanford CS221: AI (Autumn 2019) [Video]. YouTube. https://www.youtube.com/watch?v=HpaHTfY52RQ&ab_channel=stanfordonline

11-David Silver [DeepMind]. RL Course by David Silver - Lecture 2: Markov Decision Process [Video]. YouTube. https://www.youtube.com/watch?v=lfHX2hHRMVQ&list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ&index=4&ab_channel=DeepMind