PSG COLLEGE OF TECHNOLOGY

(AUTONOMOUS INSTITUTION)

COIMBATORE – 641004

ELECTRONICS AND COMMUNICATION ENGINEERING

19L702 – WIRELESS COMMUNICATION

ASSIGNMENT PRESENTATION

(BATCH – 14)

TOPIC : 5G HANDOFF USING REINFORCEMENT LEARNING

SUBMITTED BY,

21L414 - B SAKTHITHASAN

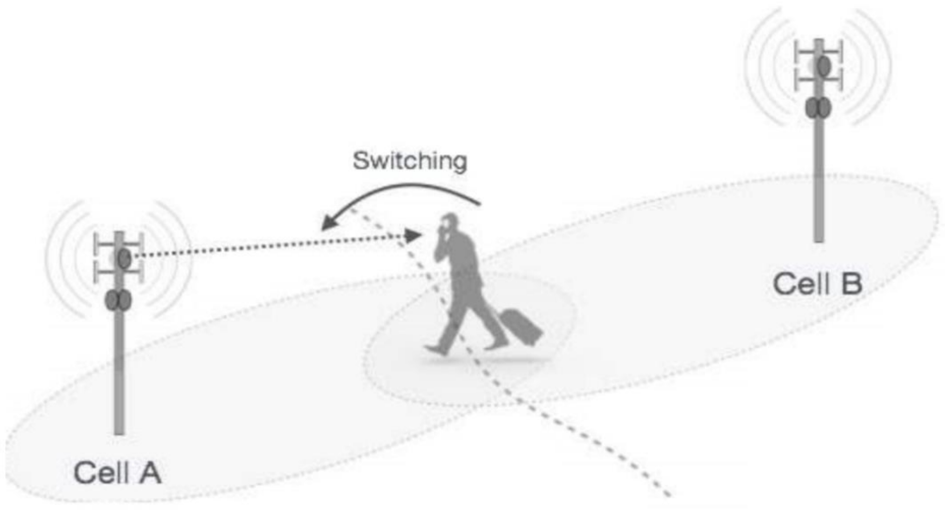
21L401 - R AKASH

20L151 - M N THARUN BALAJI

20L147 - R SURESHKUMAR

INTRODUCTION:

Handoff, also known as handover, refers to the process of transferring an ongoing mobile communication session from one cell or base station to another without interruption. Handoff is essential to maintain a continuous connection for a mobile user as they move through different coverage areas, such as when driving, walking, or using public transportation. Handoff is particularly significant in 5G networks due to their key features, such as ultra-low latency, high data rates, and massive device connectivity. The seamless handoff process ensures that these features are maintained, offering users a consistent and reliable experience.



TYPES OF HANDOFF:

There are 4 types of handoff procedures in 5g network. They are,

Intra-cell Handoff:

It involves transferring a user within the same cell, often due to changes in signal quality or sector utilization. The user remains on the same frequency and cell but may be moved to a different sector within that cell.

Inter-cell Handoff:

It happens when a user transitions from one cell to another cell, often due to moving out of the coverage range of the current cell or due to a stronger signal in the neighboring cell.

Inter-frequency Handoff:

It is essential when 5G networks use multiple frequency bands or carrier aggregation. Users need to switch between different frequency bands for optimized performance.

Inter-RAT Handoff:

It occurs when a user moves from a 5G network to a different radio access technology, such as 4G LTE or 3G, to maintain connectivity when 5G coverage is not available.

PROCESS IN HANDOFF:

Measurement and Monitoring:

Mobile devices continuously monitor the signal strength, signal quality, and other relevant parameters of the current cell and neighboring cells. Measurements are performed to assess the quality of the connection and determine if a handoff is necessary.

Threshold Crossing:

When the measured parameters fall below or exceed predefined thresholds, it indicates that the current cell's quality has degraded or a neighboring cell offers a better signal. Thresholds are set to trigger the handoff process when the signal quality is no longer suitable for maintaining a high-quality connection.

Handoff Decision:

The network evaluates the available cells based on the measurements and makes a decision to initiate a handoff. Factors considered include signal strength, signal quality, cell load, and user priority.

Target Cell Selection:

The network selects the most suitable target cell to which the mobile device will be handed over. The decision is made based on the measurements and other relevant data to ensure the best possible signal quality and user experience.

Handoff Execution:

The network sends instructions to the mobile device to switch its connection from the source cell to the selected target cell. The mobile device tunes into the new cell's frequency and begins communicating with it.

Handoff Confirmation:

Both the mobile device and the target cell confirm the successful handoff. This confirmation is essential to ensure that the mobile device is connected to the new cell and that data can continue to flow seamlessly.

Data Forwarding:

Once the handoff is confirmed, data sessions are rerouted from the source cell to the target cell. This step ensures that user applications and services experience minimal disruption during the handoff.

Quality of Service (QoS) Management:

The network continues to monitor the connection in the target cell to ensure that the user experiences the expected quality of service. It may perform dynamic adjustments and optimizations to maintain high-quality communication.

Ongoing Monitoring:

After the handoff, the mobile device continues to monitor the signal quality and other network parameters. If conditions change or if a better target cell becomes available, the handoff process can be initiated again.

Fallback Mechanism:

In cases where the handoff is not successful or if the target cell's conditions deteriorate, a fallback mechanism may be triggered to return the mobile device to the source cell or find another suitable cell.

CHALLENGES IN HANDOFF:

The challenges in handoff, especially in the context of wireless communication networks, are influenced by several factors, including ultra-low latency requirements, high mobility of users, and dynamic network conditions.

Ultra-low Latency Requirements:

- Handoff in ultra-low latency scenarios, such as 5G or mission-critical communication networks, requires extremely quick decision-making and execution to minimize service interruption.
- The challenge lies in ensuring that the handoff process, including measuring signal quality, selecting a new base station, and re-establishing the connection, is completed within the strict latency constraints.
- Any delays in the handoff process can lead to packet loss, degraded quality of service, or even service disruption, which is unacceptable in applications like autonomous vehicles, telemedicine, or industrial automation.

High Mobility of Users:

- In scenarios with high user mobility, such as vehicular communication or fast-moving devices, handoffs must be executed seamlessly and rapidly as users move from one cell to another.
- Users may switch cells quickly, making it challenging to maintain a stable connection and ensuring that the user's experience remains uninterrupted.
- Frequent handoffs can strain network resources and require efficient handoff algorithms to track and manage user movements effectively.

Dynamic Network Conditions:

- Wireless networks are subject to dynamic and fluctuating conditions, including signal interference, fading, and changing channel characteristics.
- These dynamic conditions can affect the quality of the connection and require continuous monitoring and adaptation of the handoff process.
- The network needs to make informed decisions about when and where to initiate a handoff to maintain a robust connection and deliver a seamless user experience.

To address these challenges in handoff, wireless communication systems employ various techniques and protocols. For example, fast and intelligent handoff algorithms are used to make efficient handover decisions based on signal strength, quality, and user mobility patterns. Additionally, technologies like beamforming and advanced antenna systems in 5G networks help improve the robustness of connections and reduce the impact of dynamic network conditions. Furthermore, edge computing and network slicing may be utilized to offload some of the processing tasks, reducing latency and ensuring a smoother handoff experience in ultra-low latency scenarios.

5G HANDOFF FEATURES:

In 4G LTE networks, handovers are primarily hard handovers, which means that the device discontinues its connection with the source cell before establishing a connection with the target cell. 5G networks support both hard and soft handovers. Soft handovers enable devices to connect to multiple cells simultaneously, improving reliability and reducing interruptions. In 5G, handover decision making is more intelligent. It considers various factors, including signal strength, signal quality, load on the cells, and the device's speed and direction. Machine learning, artificial intelligence, and reinforcement learning are used to optimize handover decisions.

5G handovers are designed to have lower latency compared to 4G. Soft handovers and ultra-reliable low-latency communication (URLLC) capabilities in 5G reduce the impact on latency during handovers. 5G operates in a wider range of frequency bands, including mmWave, mid-band, and low-band. The use of higher-frequency mmWave bands can enable faster handovers, as well as support for multiple connections simultaneously. 5G networks often employ massive multiple-input, multiple-output (MIMO) technology, which improves beamforming and spatial multiplexing. This enhances the performance of handovers in 5G networks.

5G HANDOFF USING REINFORCEMENT LEARNING:

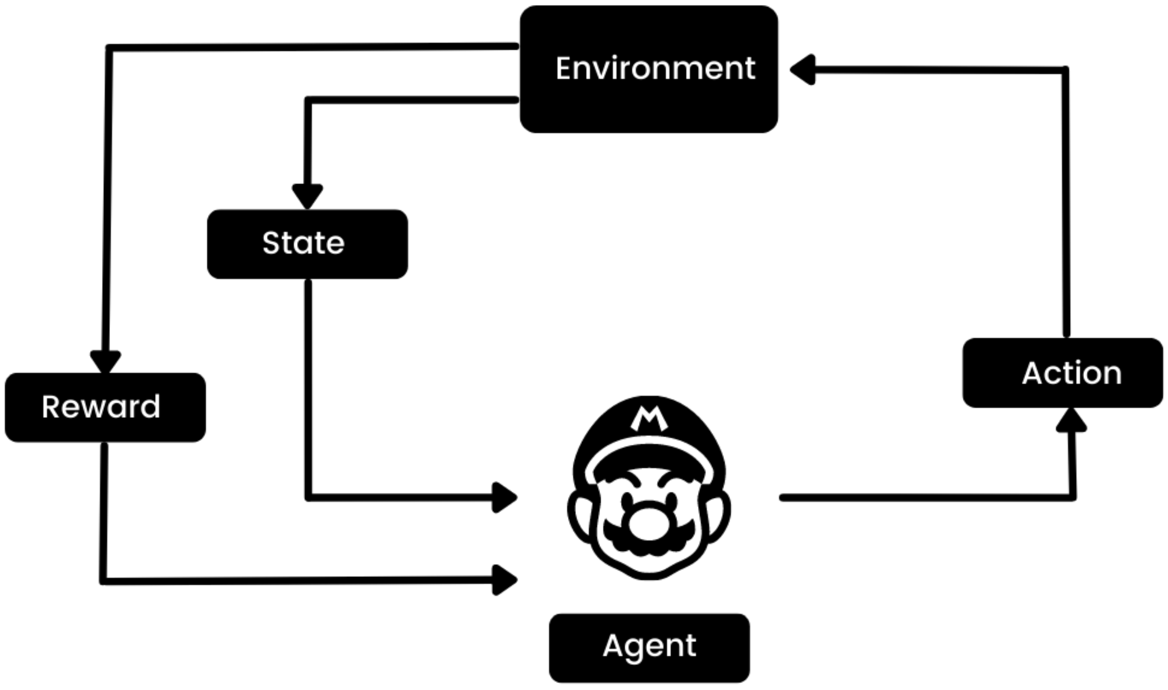
Since 4G networks have comparatively a larger signal radius, the handover from one node to another happens intermittently. Unlike the Fourth-generation network, 5G wireless signals are transmitted through a large number of closely located gNodeBs. So, after the deployment of pure standalone 5G networks, there will be a possibility of frequent inter gNodeB handovers (shifting from one gNodeB to another). Handovers decrease performance and consume a lot of energy. Now, we are aiming to optimize the number of handovers by using reinforcement learning.

This being a real-time problem, we would not have a training data set to make the agent learn. Instead, the agent has to interact with the environment to collect information. Therefore, we use reinforcement learning techniques to optimize the number of handovers which in turn maintains hassle-free connectivity with lesser number of overall handovers. Handovers also cause the ping pong effect. It is the handover happening to and fro between two gNodeBs frequently when the user equipment is present in the range of two gNodeBs. This problem is also addressed by optimising the handovers using reinforcement learning.

We are creating a 3x6 grid world environment with multiple antennas close to each other. The agent moves around randomly and learns about all the available antennas at each state. We are using the famous Q-learning method of reinforcement learning to construct a Q table after performing 5000 iterations. The antenna with the highest Q value should be chosen to optimize handovers.

REINFORCEMENT LEARNING:

Reinforcement learning (RL) is the part of the machine learning ecosystem where the agent learns by interacting with the environment to obtain the optimal strategy for achieving the goals. It is quite different from supervised machine learning algorithms, where we need to ingest and process that data. Reinforcement learning does not require data. Instead, it learns from the environment and reward system to make better decisions.



For example, in the Mario video game, if a character takes a random action (e.g. moving left), based on that action, it may receive a reward. After taking the action, the agent (Mario) is in a new state, and the process repeats until the game character reaches the end of the stage or dies. This episode will repeat multiple times until Mario learns to navigate the environment by maximizing the rewards.

We can break down reinforcement learning into five simple steps:

- The agent is at state zero in an environment.
- It will take an action based on a specific strategy.
- It will receive a reward or punishment based on that action.
- By learning from previous moves and optimizing the strategy.
- The process will repeat until an optimal strategy is found.

Q-LEARNING:

Q-learning is a model-free, value-based, off-policy algorithm that will find the best series of actions based on the agent's current state. The “Q” stands for quality. Quality represents how valuable the action is in maximizing future rewards.

The model-based algorithms use transition and reward functions to estimate the optimal policy and create the model. In contrast, model-free algorithms learn the consequences of their actions through the experience without transition and reward function.

The value-based method trains the value function to learn which state is more valuable and take action. On the other hand, policy-based methods train the policy directly to learn which action to take in a given state.

In the off-policy, the algorithm evaluates and updates a policy that differs from the policy used to take an action. Conversely, the on-policy algorithm evaluates and improves the same policy used to take an action.

TERMINOLOGIES IN Q-LEARNING:

States(s): The current position of the agent in the environment.

Action (a): A step taken by the agent in a particular state.

Rewards: For every action, the agent receives a reward and penalty.

Episodes: The end of the stage, where agents can’t take new action. It happens when the agent has achieved the goal or failed.

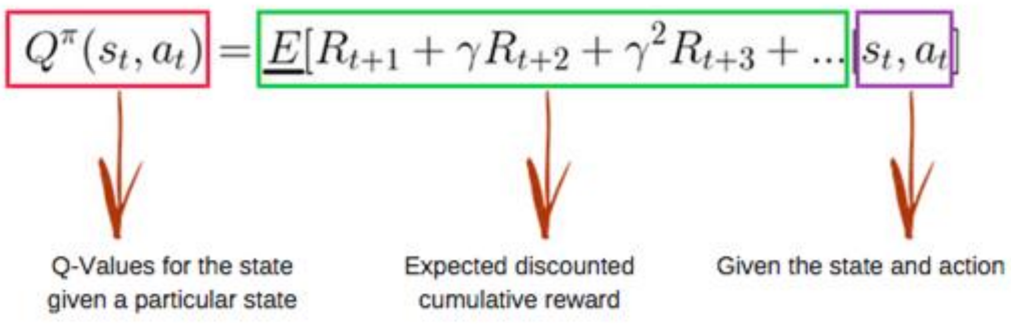
Q(St+1, a): Expected optimal Q-value of doing the action in a particular state.

Q(St, At): It is the current estimation of Q(St+1, a).

Temporal Differences (TD): Used to estimate the expected value of Q(St+1, a) by using the current state and action and previous state and action.

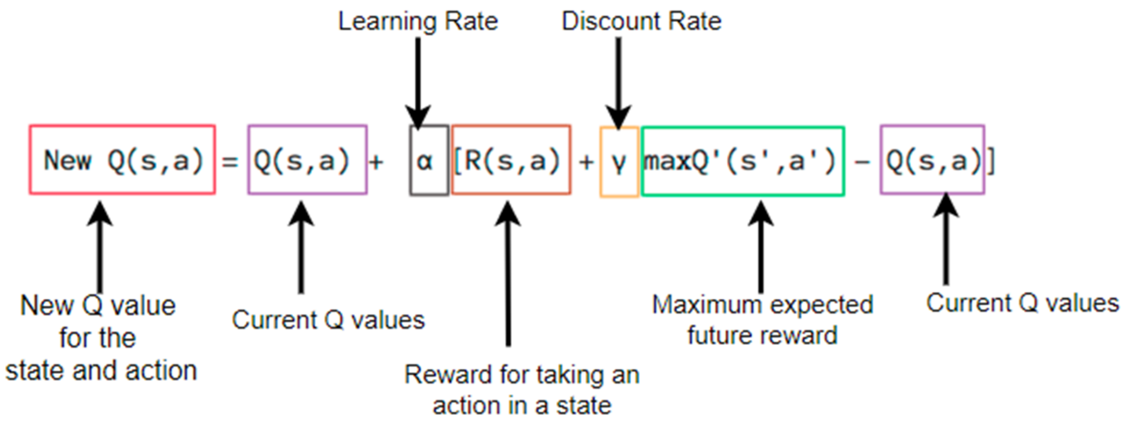
Q-Table: The agent will use a Q-table to take the best possible action based on the expected reward for each state in the environment. In simple words, a Q-table is a data structure of sets of actions and states, and we use the Q-learning algorithm to update the values in the table.

Q-Function: The Q-function uses the Bellman equation and takes state(s) and action(a) as input. The equation simplifies the state values and state-action value calculation.

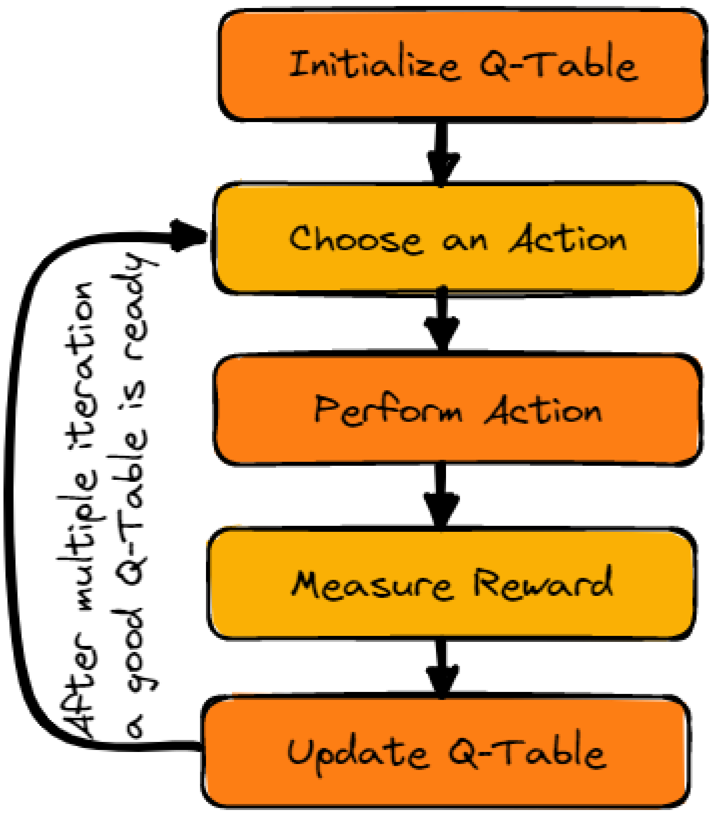

$Q^{\pi}(s_t, a_t) = \underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | s_t, a_t]$
Q-Values for the state given a particular state
Expected discounted cumulative reward
Given the state and action

UPDATING Q-TABLE:

We will update the function Q(S_t, A_t) using the equation. It uses the previous episode's estimated Q-values, learning rate, and Temporal Differences error. Temporal Differences error is calculated using Immediate reward, the discounted maximum expected future reward, and the former estimation Q-value. The process is repeated multiple times until the Q-Table is updated and the Q-value function is maximized.


Learning Rate
Discount Rate
New Q(s,a) = Q(s,a) + α [R(s,a) + γ maxQ'(s',a') − Q(s,a)]
New Q value for the state and action
Current Q values
Reward for taking an action in a state
Maximum expected future reward
Current Q values

Q-LEARNING ALGORITHM:

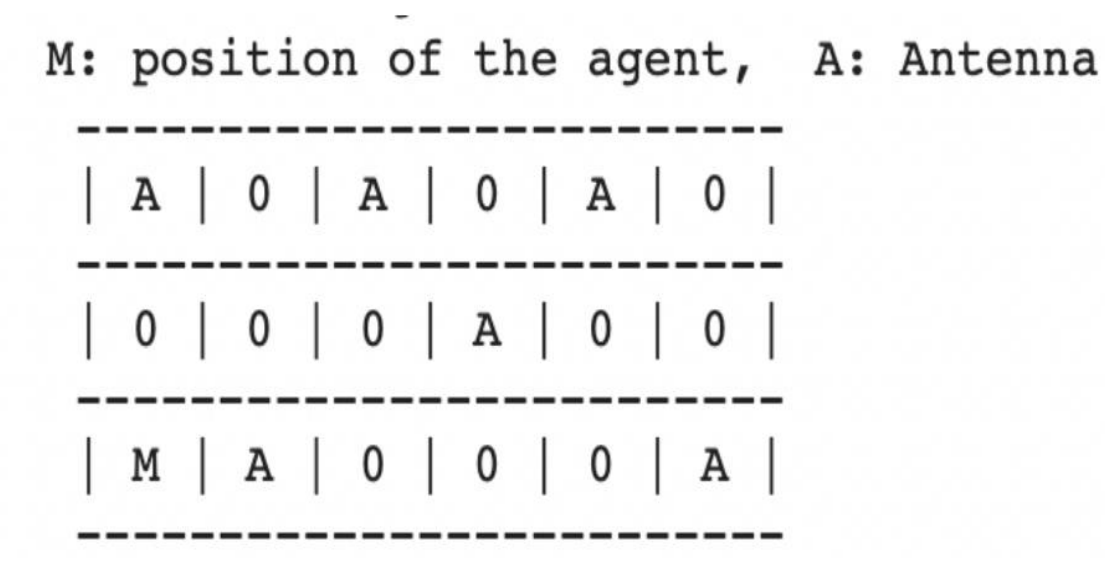


Algorithm 1 Q-Learning

```
0: Q($s$,$a$) initiliazed arbitrarily
0: for each episode $i$ do
0:     State $s$ initialized
0:     for each step $j$ do
0:         action $a$ from state $s$ using policy derived by Q
0:         action $a$ taken
0:         reward $r$ and state $s'$ observed
0:         $Q$(s,a) $= Q$(s,a) $+ \alpha[r + \gamma max_{a'} Q$(s',a')-$Q$(s,a)$]$
0:         $s = s'$
0:     end for
0: end for=0
```

ENVIRONMENT:

We would need a real-world environment with 5G antennas and mobile phones to implement our challenge. To simulate such a scenario, we constructed a 2D grid world with a 3 x 6 grid size that can be understood as a real-world path, and a cell phone serves as our RL agent. Multiple antennas are positioned across the path in this 3 × 6 grid, ensuring that the agent has access to at least one antenna no matter where the agent is.

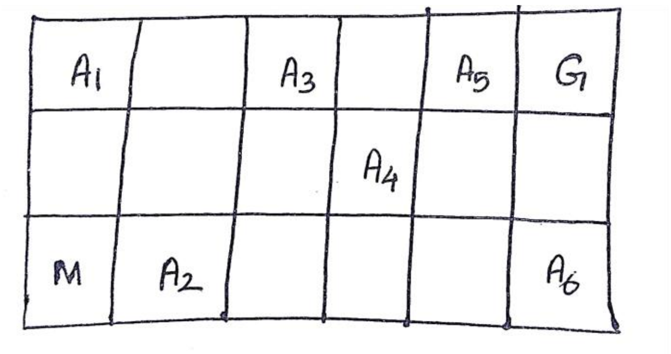


```
M: position of the agent,  A: Antenna
 -------------------------
 | A | 0 | A | 0 | A | 0 |
 -------------------------
 | 0 | 0 | 0 | A | 0 | 0 |
 -------------------------
 | M | A | 0 | 0 | 0 | A |
 -------------------------
```

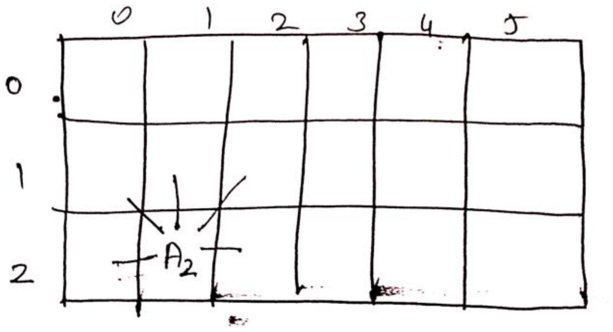
Start position - (2,0)

Goal position - (0,5)

We have 6 antennas in our environment where each antenna is placed in each column. Each antenna covers the neighbouring grids from its position.



We determine the signal availability for each coordinate in the grid first, and we assume that our antenna's signal intensity is 1 in all directions, including the diagonals.



ENVIRONMENT PARAMETERS:

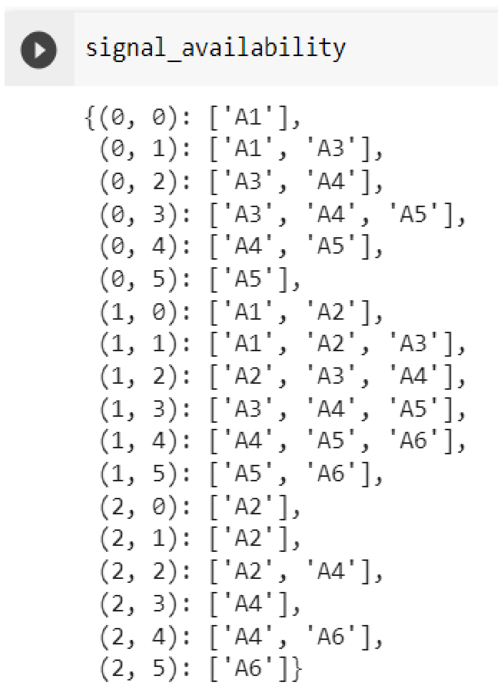
State space - coordinate points of the grid world

Action space - {right, left, up, down}

Reward:

```
if <change of antenna>:
      reward = −1   (Handover)
      handover = 1
else:
      reward = 1   (No Handover)
      handover = 0
```

The signal availability in each grid is defined as follows,



```
signal_availability
```

```
{(0, 0): ['A1'],
 (0, 1): ['A1', 'A3'],
 (0, 2): ['A3', 'A4'],
 (0, 3): ['A3', 'A4', 'A5'],
 (0, 4): ['A4', 'A5'],
 (0, 5): ['A5'],
 (1, 0): ['A1', 'A2'],
 (1, 1): ['A1', 'A2', 'A3'],
 (1, 2): ['A2', 'A3', 'A4'],
 (1, 3): ['A3', 'A4', 'A5'],
 (1, 4): ['A4', 'A5', 'A6'],
 (1, 5): ['A5', 'A6'],
 (2, 0): ['A2'],
 (2, 1): ['A2'],
 (2, 2): ['A2', 'A4'],
 (2, 3): ['A4'],
 (2, 4): ['A4', 'A6'],
 (2, 5): ['A6']}
```

SIMULATION PARAMETERS:

Episodes - 5000

Timesteps per episode - 150

Epsilon - 0.82

Learning rate - 0.1

Discount factor - 0.95

IMPLEMENTATION:

The main idea is that the agent walks through the grid world moving from point A to point B. There are multiple closely spaced gNodeBs along the way the agent is traveling. In a pure 5g network, the gNodeBs have to be placed close to each other because of their small range. The agent performs a random walk and sees a lot of available gNodeBs to connect to. It has to connect to the best available gNodeB and keep moving towards the goal. Agents can then distill the results from planning into a learned policy. To achieve our goal we should find the optimal action-selection policy, since our agent performs a random walk, we have to apply the optimal action-selection policy on the antenna which the agent has to connect to reduce handovers.

We use Q-Learning to optimize our antenna selection. Q-Learning is a value-based reinforcement learning algorithm that is used to find the optimal action-selection policy using a Q function. The value function Q should be maximized as much as possible. The Q table assists us in determining the appropriate course of action for each state. It aids in maximizing the expected payoff by allowing you to choose the best of all available actions. The expected future payoff of that action at that state is returned by Q(state, action). This function can be approximated using Q-Learning, which uses the Bellman equation to iteratively update Q(s, a). We begin by investigating the environment and updating the Q-Table.

The agent will begin to exploit the surroundings and take better actions once the Q-Table is ready. The Epsilon greedy strategy concept comes into play next to explore and exploit the environment. In the beginning, the epsilon rates will be higher. The agent will explore the environment and randomly choose actions. This occurs logically since the agent does not know anything about the environment. As the agent explores the environment, the epsilon rate decreases and the agent starts to exploit the environment.

One of the advantages of Q-Learning is that it can compare the expected utility of various actions without the need for a model of the environment. Reinforcement Learning is a method of problem-solving in which the agent learns without the assistance of a tutor. In a reinforcement environment, the agent’s main signal for learning from his actions is the so-called reward, a number that tells him if his last action was good (or) not. Q-Learning is a type of Reinforcement Learning algorithm that doesn’t require a model of the environment and may be utilized in real-time.

CODE:

```
import numpy as np
import random
import matplotlib.pyplot as plt
Rows = 3
Cols = 6
Start = (2,0)
Goal = (0,5)
Antennas={
    "A1":{
        "position":(0,0),
        "Range":[(1,0),(1,1),(0,1),(0,0)]
    },
    "A2":{
       "position": (2,1),
       "Range":[(2,0),(1,1),(2,2),(1,0),(1,2),(2,1)]
    },
    "A3":{
        "position":(0,2),
        "Range":[(0,1),(1,1),(1,2),(1,3),(0,3),(0,2)]
    },
    "A4":{
        "position":(1,3),
        "Range":[(0,2),(1,2),(2,2),(2,3),(2,4),(1,4),(0,4),(0,3),(1,3)]
    },
    "A5":{
        "position":(0,4),
        "Range":[(0,3),(1,3),(1,4),(1,5),(0,5),(0,4)]
    },
    "A6":{
        "position":(2,5),
        "Range":[(1,4),(2,4),(1,5),(2,5)]
    }
    }
pos =[(0,0),(2,1),(0,2),(1,3),(0,4),(2,5)]
signal_availability = dict()
lis = []
for r in range(Rows):
  for c in range(Cols):
    for i in Antennas.keys():
      for j in Antennas[i]["Range"]:
       if j == (r,c):
        lis.append(i)
    signal_availability.update({(r,c):lis})
    lis = []
Actions = ['r','u','l','d']
Q = {}

signal_availability

class Environment:
  def __init__(self):
    self.action_space = Actions
    self.rows = Rows
    self.cols = Cols
    self.state = Start
    self.Start = Start
    self.Goal = Goal
    self.Antennas = Antennas
    self.done = False
    self.grid = np.zeros((self.rows,self.cols))
    self.grid[self.Goal] = 2
    for i in pos:
      self.grid[i] = -1
  def reset(self):
    self.state = Start
    self.done = False

  def step(self,action,antenna,eps):
    r,c = self.state
    old_antenna = antenna
    if action =='u':
      r -= 1
    elif action =='d':
      r +=1
    elif action =='r':
      c +=1
    elif action =='l':
      c -=1

    if np.random.rand() < eps:
      for i in signal_availability.keys():
        if i == (r,c):
          antenna = random.choice(signal_availability[i])
    else:
      for i in signal_availability.keys():
        if i == (r,c):
          antenna = max(Q[(r,c)],key=Q[(r,c)].get)
    new_antenna = antenna

    if r>=0 and r <=self.rows -1 and c>=0 and c <=self.cols -1:
      self.state = (r,c)

    if old_antenna != new_antenna:
      reward = -1 #Handover
      handover = 1
    else:
      reward = 1 #No Handover
      handover = 0

    return self.state, reward, handover, self.done

class Agent:
  def __init__(self):
    self.rows = Rows
    self.cols = Cols
    self.actions = Actions
    for r in range(Rows):
      for c in range(Cols):
        Q[(r,c)] = {}
        for i in signal_availability.keys():
          if i == (r,c):
            for ant in signal_availability[i]:
              Q[(r,c)][ant] = 0

  def antenna_selection(self, state, action, eps):
    r, c = state
    if np.random.rand() < eps:
      for i in signal_availability.keys():
        if i == (r,c):
          antenna = random.choice(signal_availability[i])
    else:
      for i in signal_availability.keys():
        if i == (r,c):
          antenna = max(Q[(r,c)],key=Q[(r,c)].get)
    return antenna

  def action_selection(self, state):
    r, c = state
    action = np.random.randint(len(Actions))
    action = Actions[action]
    return action

  def Q_update(self, state, antenna, next_state, reward):
    next_max = max(list(Q[next_state].values()))
    Q[state][antenna] = (1 - lr) * Q[state][antenna] + lr * (reward + gamma* next_max)

  def reset(self):
    Q = {}
    for r in range(Rows):
      for c in range(Cols):
        Q[(r,c)] = {}
        for i in signal_availability.keys():
          if i == (r,c):
            for ant in signal_availability[i]:
              Q[(r,c)][ant] = 0

Episodes = 5000
lr = 0.1
gamma = 0.95
```

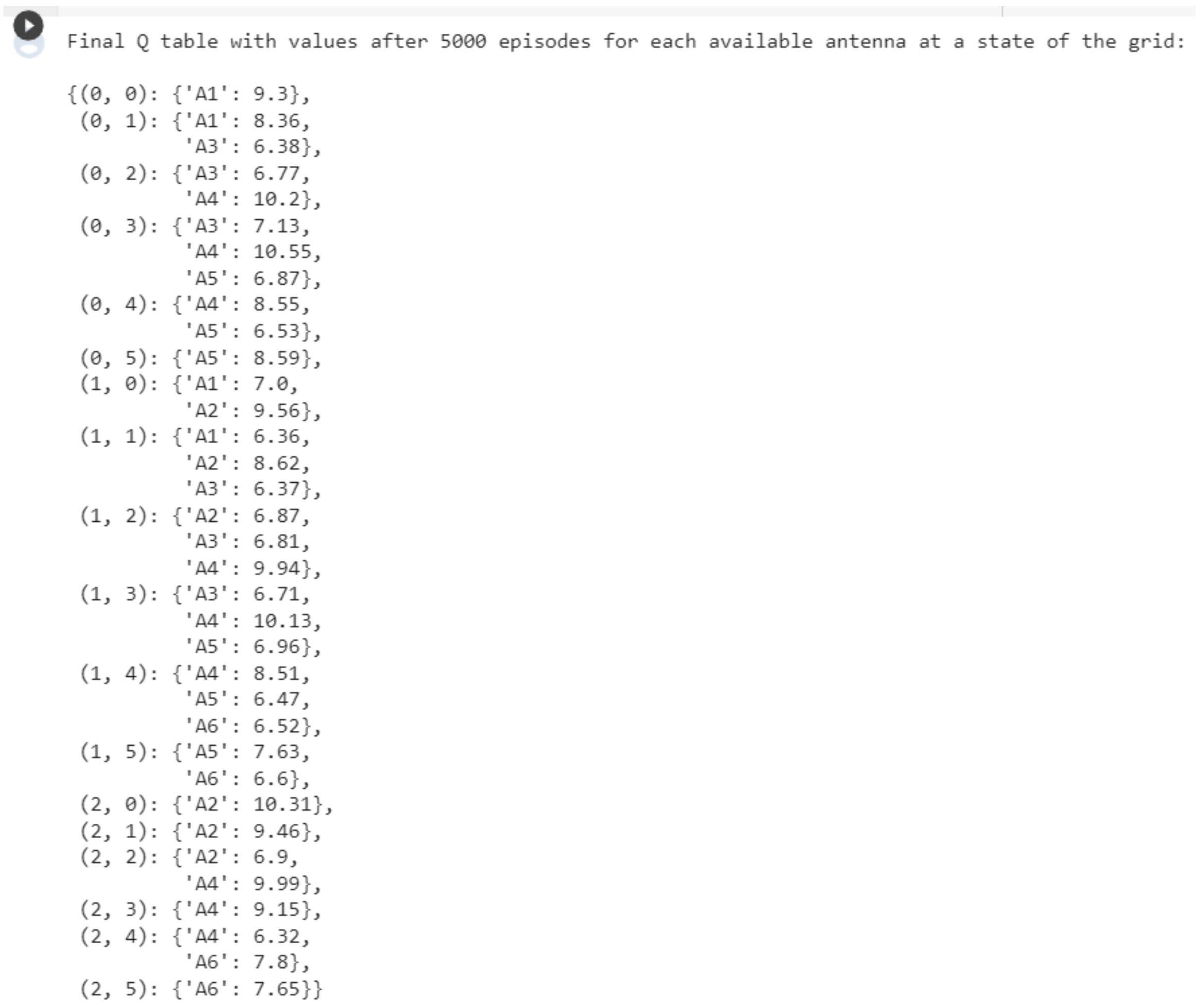
```
eps = 0.82

env = Environment()
agent = Agent()
agent.reset()
max_steps_per_episode = 150
Rewards = []
steps_taken = []
Handovers = []
eps_decaying_start = 0
eps_decaying_end = Episodes//1.5
eps_decaying = eps/(eps_decaying_end - eps_decaying_start)

for ep in range(Episodes):
  if eps_decaying_start < ep < eps_decaying_end:
        eps -= eps_decaying
  counter = 0
  env.reset()
  done = False
  state = env.state
  Reward_ep = 0
  Handover_ep = 0
  while not done and counter < max_steps_per_episode:
    action = agent.action_selection(state)
    antenna = agent.antenna_selection(state, action, eps)
    next_state, reward, handover, done = env.step(action,antenna,eps)
    agent.Q_update(state, antenna, next_state, reward)
    Reward_ep += reward
    Handover_ep += handover
    state = next_state
    counter +=1
  print("Episode :", ep, "Reward :", Reward_ep, "Handover :", Handover_ep)
  Rewards.append(Reward_ep)
  Handovers.append(Handover_ep)
  steps_taken.append(counter)

plt.plot(range(Episodes),Rewards)
plt.xlabel("Episodes")
plt.ylabel("Rewards")
plt.show()

plt.plot(range(Episodes),Handovers)
plt.xlabel("Episodes")
plt.ylabel("Handovers")
plt.show()

import pprint
for r in range(Rows):
    for c in range(Cols):
        for i in signal_availability.keys():
          if i == (r,c):
            for ant in signal_availability[i]:
              Q[(r,c)][ant] = round(Q[(r,c)][ant],2)

print("Final Q table with values after {} episodes for each available antenna at a
state of the grid: \n".format(ep+1))
pprint.pprint(Q,width=1)
```

CODE EXPLANATION:

- Since we're using a random walk strategy, our agent starts at coordinate (2,0) and can do any action from our predetermined action space. The agent must choose an antenna to connect to from its signal availability vector in addition to picking an action after each step.
- When the agent takes a step, our goal is to optimize the antenna selection. As previously said, we will use Q-Learning to achieve this goal, which means there will be a Q-table that will be updated after each step using the Bellman equation based on the action taken by the agent during its exploration phase.
- We keep track of the epsilon value that degrades as the exploration progresses to leverage our antenna selection. We set epsilon to 0.82 and set the number of steps done by an agent in one episode before resetting to the start state to 150. If the value of np.random.rand() is less than 0.82, the agent chooses an antenna at random and goes forward; otherwise, the agent chooses the best antenna at that particular state. In this case, the Antenna with the highest Q-value for that coordinate is chosen using the signal availability vector and Q table.
- If the antenna we choose is the same as the one we already have connected to, there will be no handover between the gNodeBs, and the reward will be +1; otherwise, the reward for a handover will be -1. We created a negative reward for handover to encourage the agent to learn and seek the highest possible reward by going to the destination with the fewest possible handovers.
- Following the previously established Q-learning method, the agent first picks an action, after which an antenna is selected depending on available antennas at that state. The Q-table updates at every episode.
- We are keeping track of the accumulated rewards at each episode. We also define a Handovers counter to determine whether or not the amount of handovers is being optimized.
- To ensure that the agent learns successfully, we now run the process for 5000 episodes with 150 steps every episode.

RESULTS:

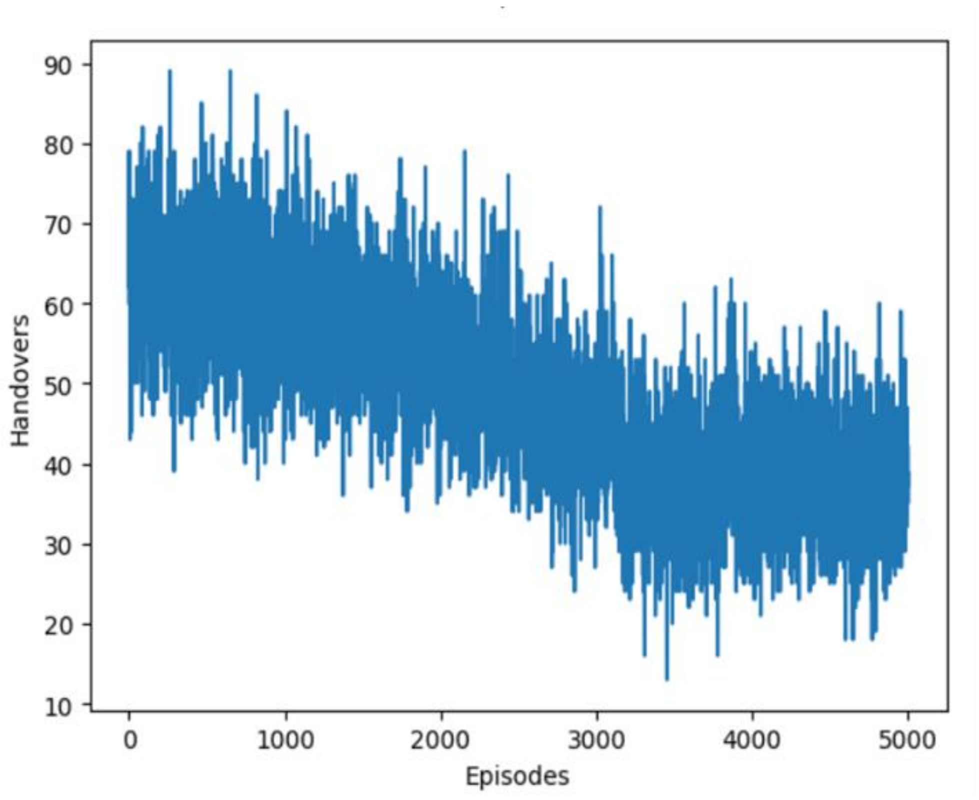
Q-values for different antennas in each grid after training,



```
Final Q table with values after 5000 episodes for each available antenna at a state of the grid:

{(0, 0): {'A1': 9.3},
 (0, 1): {'A1': 8.36,
          'A3': 6.38},
 (0, 2): {'A3': 6.77,
          'A4': 10.2},
 (0, 3): {'A3': 7.13,
          'A4': 10.55,
          'A5': 6.87},
 (0, 4): {'A4': 8.55,
          'A5': 6.53},
 (0, 5): {'A5': 8.59},
 (1, 0): {'A1': 7.0,
          'A2': 9.56},
 (1, 1): {'A1': 6.36,
          'A2': 8.62,
          'A3': 6.37},
 (1, 2): {'A2': 6.87,
          'A3': 6.81,
          'A4': 9.94},
 (1, 3): {'A3': 6.71,
          'A4': 10.13,
          'A5': 6.96},
 (1, 4): {'A4': 8.51,
          'A5': 6.47,
          'A6': 6.52},
 (1, 5): {'A5': 7.63,
          'A6': 6.6},
 (2, 0): {'A2': 10.31},
 (2, 1): {'A2': 9.46},
 (2, 2): {'A2': 6.9,
          'A4': 9.99},
 (2, 3): {'A4': 9.15},
 (2, 4): {'A4': 6.32,
          'A6': 7.8},
 (2, 5): {'A6': 7.65}}
```

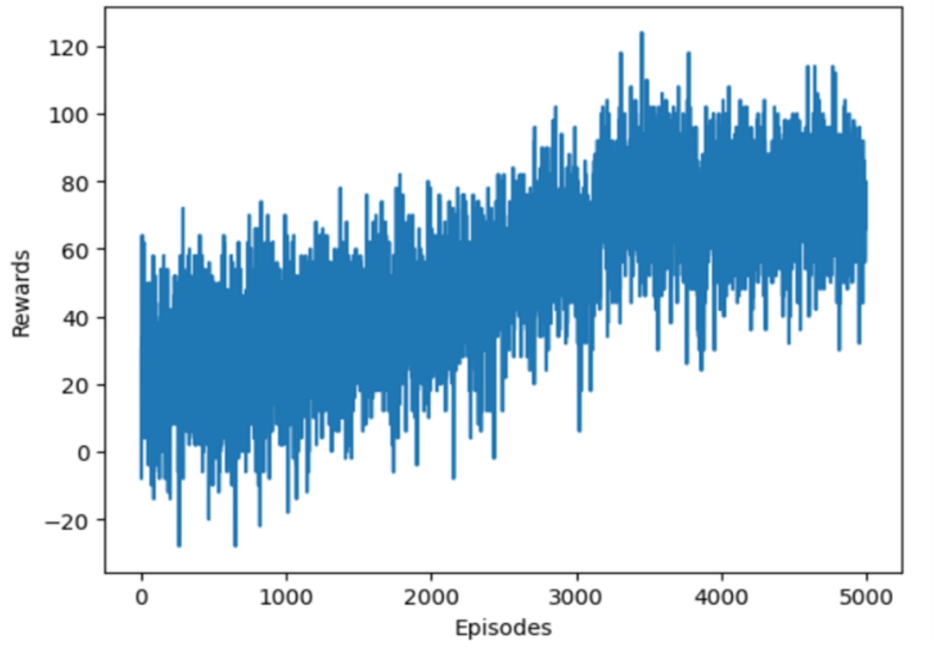
Because this is a random walk experiment, there is no stable visualization to show that the agent has reduced the number of handovers. Instead, we evaluate our algorithm based on the Q-table values. If the values of the Q-table are updated suitably, it means that this policy can be applied to any fixed path and the agent performs with fewer handovers than before.

Episodes vs Handovers:



As the episodes increases, number of handovers is decreased.

Episodes vs Rewards:



As episodes increases, Rewards obtained for each antenna selection increases.

CONCLUSION:

We showed how the agent started from the initial state and performed random walks in all possible directions to learn about different states in the grid world environment and reduce the number of overall handovers in the dense 5G network. We showed how the final Q-table can be utilized to select the best antenna at a particular state of the grid world. The results also indicate how the final Q-table can be used to minimize the ping pong effect caused by a gNodeB pair. The number of episodes can be increased and also the epsilon value for choosing antennas can also be modified to make the algorithm more robust. DynaQ learning can be applied instead of Q-Learning to make the model learn in fewer iterations. This model can be deployed in the 5G enabled mobile phones of users to learn about the neighboring gNodeBs. The problem of the selection of link beams in a 5G network can also be addressed using the proposed model.

REFERENCES:

BASE PAPER:

https://drive.google.com/file/d/1F_e6_DQMz2mRmAPaIoJefKd-0ZV8ZIae/view?usp=sharing

REFERENCE PAPERS:

https://drive.google.com/file/d/1c_i-rpPVyddknlyHIg94A_CGYbEBVzLb/view?usp=sharing

https://drive.google.com/file/d/1A-wdAxhK-g-Xz6QTPEPwJxKc2s49a753/view?usp=sharing

Q-LEARNING:

https://youtu.be/iKdlKYG78j4?si=heq1K7kS4DsPVF0k

https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial

https://github.com/gowtham-chandrasekaran/Handover-Optimisation-in-5G-using-Reinforcement-Learning

https://github.com/jpshlima/lstm-handover/tree/main#welcome-to-deep-learning-based-handover-prediction-for-5g-and-beyond-networks-repository