Application of Reinforcement Learning in 5G Millimetre-Wave Networks

Artmiz Golkaramnay

A Thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Science (Electrical and Computer Engineering) at
Concordia University
Montreal, Quebec, Canada

April 2020

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Artmiz Golkaramnay

Entitled: Application of Reinforcement Learning in 5G Millimeter-Wave Networks

and submitted in partial fulfillment of the requirements for degree of

Master of Science (Electrical and Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

________________________________ Chair

Dr. Wei-Ping Zhu

________________________________ Examiner

Dr. Chadi Assi

________________________________ Supervisor

Dr. Mustafa Mehmet Ali

________________________________ Supervisor

Dr. Dongyu Qiu

Approved by ____

Dr. Y. R. Shayan, Chair

Department of Electrical and Computer Engineering

______ 2020 ____

Dr. Amir Asif, Dean

Faculty of Engineering and Computer Science

ABSTRACT

Application of Reinforcement Learning in 5G Millimeter-Wave Networks

Artmiz Golkaramnay

The increasingly growing number of mobile communications users and smart devices have attracted researchers and industry pioneers to the largely under-utilized spectrum in the millimeter-wave (mmWave) frequency bands for the 5th generation of wireless networks. This could provide hundreds of times more capacity as compared to 4G cellular networks. The main reason for ignoring the mmWave spectrum until now, has been its vulnerability to signal blockages and possible disconnection or interruption in service. Considering that today's mobile users expect high reliability and throughput connections, the mmWave signal sensitivity to blockages must be addressed. This research proposes to predict base stations that can service a user without disconnections, given the user's path or destination in the network.

In modern networks, reinforcement learning has been effectively utilized to obtain optimal decisions (or actions being taken) in small state-action spaces. Deep reinforcement learning has been able find optimal policies in larger network spaces. In this work, similar techniques are employed to find ways to serve the user without service disconnection or interruption. First, using dynamic programming for a fixed user path, the exact optimal serving base stations are listed. Then, using Q-learning, the network will learn to predict the optimal user path and serving base stations listed, given a fixed destination for the user. Lastly, deep Q-learning is used to approximate optimal user paths and base station lists along that path, similar to the Q-learning results, which can also be applied to networks with more sophisticated state spaces.

ACKNOWLEDGMENTS

I wish to thank everyone whose assistance was a milestone in the completion of this project.

The time and energy my supervisors, Dr. Dongyu Qiu and Dr. Mustafa Mehmet Ali, dedicated to me and this project have been beyond expectation. You have always been available, willing, and dedicated. It has been a pleasure to learn from you both.

Dr. Qiu, your tenacious and knowledgeable supervision has and will continue to inspire me. My deepest gratitude for all your teachings through each stage of this project.

Dr. Mehmet Ali, your un-bounded guidance and support, whether technical or personal, has been a gift. My sound appreciation for all your advice.

To my parents, Mohammad Rasoul and Maryam, my sister, Ana, and my boyfriend, Cody, thank you for your unwavering support and encouragement. Thank you for always believing in me and being there for me. I am blessed to be loved by you all. Baba Rasoul and Maryam, all the sacrifices you have made in life have driven me forward in life, you have always put my needs and my education above else and I am eternally grateful to you. The examples you set for me, as an engineer and as a member of society will continue to urge me to try harder, everyday.

To my mentor, Dr. McQuillan, who showed me the joy of research and the beauty in discovery, I can never thank you enough.

Last, but not least, thank you to all the faculty and staff of Concordia University, the department of Electrical and Computer Engineering, and the School of Graduate Studies for their respective services.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

5G	5th Generation (of wireless networks)
AI	Artificial Intelligence
ANN	Artificial Neural Networks
Bps	bits per second
BS	Base Station
BW	Band Width
DL	Deep Learning
DP	Dynamic Programming
DP-FP	Dynamic Programming (using a) Fixed Path
DQL	Deep Q-Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
Gbps	Giga bits per second
GHz	Giga Hertz
LOS	Line-of-Sight
LTE	Long Term Evolution
MDP	Markov Decision Process
MDR-FP	Maximum Data Rate (using a) Fixed Path
MIMO	Multiple-Input Multiple-Output
MIR-FP	Maximum Immediate Reward (using a) Fixed Path
ML	Machine Learning
mmWave	Millimeterwave
NLOS	Non-Line-of-Sight
Q-learning	Quality Learning
QoS	Quality of Service
RL	Reinforcement Learning
SINR	Signal-to-Noise-and-Interference-Ratio
SNR	Signal-to-Noise-Ratio
TD	Temporal Difference
μWave	Microwave

Chapter 1

Introduction

1.1 Introduction and Motivation

The necessity for spectrum in the world of Internet of Things (IOT) and the overcrowded sub-6GHz spectrum, has researchers exploring the previously unused millimeter-wave (mmWave) spectrum [1]. For the past few years, the mmWave frequencies have been investigated for the 5th generation (5G) wireless systems [2].

When 5G wireless networks are made available, they will have enhanced mobile broadband with high throughput rates (10 GB/s peak throughput rate [2]), but reliability and latency could become drawbacks. The sensitivity of mmWave signals to physical blockages, can cause disruptions in the connection or cut the connection off entirely [3].

If there is no line-of-sight between a serving base station and the user, given that the user is not disconnected from the network, handing the user off to another base station incurs overhead and latency issues [3]. On the bright side, interference is less of an issue in mmWave signals as compared to μWave signals, because of the highly directional nature of the these signals [4].

To address the overhead and latency issues for hand-off of the user from a non-line-of-sight (NLOS) base station to a line-of-sight (LOS) base station, this thesis research proposes to predict the user's path and base stations to connect to, for a given destination. By determining the user's path, along with information about the network's layout, any possible physical blockages that could render a base station NLOS can be predicted. Then, the connection could switch to another LOS base station that can serve the user. This work defers from previous work (such as [3]), in that the decision to switch to an LOS base station is made knowing whether there is a blockage making the base station only temporarily NLOS. The length of time that defines a blockage temporary is dependant on the hand-off overhead, or switching cost, which will be discussed in later chapters.

Predicting the base stations for the user to connect to, is done using several different methods. The following sections clarify the problem statement, introduce the 5G network under consideration, and provide a brief explanation of the basics of Markov decision processes. The later chapters will include descriptions of different machine learning algorithms, different approaches being used in the literature, provide overviews of the methods explored in this research, scenarios studied to provide numerical results of the methods used, and end with a discussion of the results and suggested future work.

1.2 Problem Statement

As mentioned, the exponentially increasing number of mobile users with higher bandwidth and lower latency demands, has called for the use of the mmWave spectrum in 5G wireless networks.

Because mmWave signals experience more specular propagation and reduced diffraction [4] as compared to µWave signals, they are more vulnerable to blockages [4]. While solutions are being considered to address the sensitivity of the mmWave to blockages [3], the suggestion being explored here is to foresee the duration and length of blockages. By knowing the layout of the network and blockage locations, it may be possible for an LOS base station to provide service to the user currently connected to an NLOS base station, if necessary. This way the mobile user can be handed off to a new BS, without experiencing a possible disconnection or interruption in service caused by an NLOS user-BS link. This, however, will expose the user's connection to hand-off overhead, referred to as switching cost in this research.

The problem is defined with mmWave 5G wireless network characteristics in mind, for a single mobile user moving at walking speed. Transmitting and receiving beams are assumed to be ideally aligned for hand-offs [4]. The network's base stations and stationary blockages are all assumed to be in a 2-dimensional space. The number of base stations in the network are determined using a base station density value that is calculated based on the coverage radius of the base stations and the size of the network. The locations of these base stations are decided randomly, constraining them to be placed inside the network. The number of blockages in the network are calculated using a blockage density of 0.001. They are located randomly inside the network, using a Poisson distribution, given the size of the network. These blockages are all assumed to be in the form of a line. Because the user's movement speed is assumed to be constant, it is ignored in the calculations regarding the user's directions and steps.

It is important to point out that the objective of this research is not to find the connections in a wireless mmWave network with the highest throughput, but in fact to find a connection that provides higher throughput considering the overhead associated with user hand-off. In some cases, this approach may translate into choosing a connection with lower throughput, but higher overall reward.

1.3 What is 5G and Why use mmWave?

With the explosive growth of mobile wireless communication devices, the 5th generation of cellular network, more commonly known as 5G, was introduced as a solution to meet rising demands. In 2014, it was predicted that traffic demand would grow 1000 times over the next decade [5]. Higher data rates (multi-Gbps) and lower latencies are two of the major requirements for 5G, which require architectural and component design shifts from 4th generation wireless network [6]. Even though, research in enhancing the 4th generation of wireless networks (4G), such as Long Term Evolution (LTE) to provide higher data rates have shown some progress, they are not long-term solutions to the continuously increasing demands of the 2020s [7]. Because spectrum is somewhat scarce in the microwave range, it has been proposed and researched to use millimetre wavelengths in 5G [6]. So, in order to provide higher throughput, 5G networks are expected to use a wide frequency band, millimeter wave, with operating frequencies in the 3.5 and 28 GHz range [8]. It is expected that a 5G network would provide a minimum of 1Gbps data rate anywhere, up to 5 Gbps for high mobility users, and 50 Gbps for pedestrian users [7].

But, mmWave signals suffer from acute pathloss due to oxygen molecules, water vapor, and rain drops in the atmosphere and so this section of the spectrum has not been exploited until now. However, having large antenna arrays that direct the energy beam will help with this issue [1]. So, as mentioned, the higher data rates in 5G mmWave networks introduce trade-offs. One of the main issues with mmWave is blockage, and another could be beam misalignment. The blockage problem occurs when the signal cannot pass through a physical obstacle, and this affects the signal-to-noise ratio (SNR) on the receiving end [8]. At higher frequencies, the blockage issues such as penetration, precipitation, and foliage losses are considerable even though the actual amounts of additional propagation losses vary depending on the material of the building, the strength of rain, or the thickness of foliage, respectively [7]. This issue, which was not a problem in µWave networks needs to be addressed so that mmWave networks can be taken advantage of in 5G.

The other issue with mmWave, beam misalignment, happens when the transmitting and receiving beam pairs are not matching/aligned, which also affects the SNR on the receiving end [8]. This issue is beyond the scope of this research.

1.4 Markov Decision Processes

This section introduces Markov Decision Processes (MDPs) which lay the foundation for machine learning, discussed in later chapters.

A Markov chain, in relation to discrete stochastic processes, is defined for processes where the outcome of any given experiment can affect the outcome of the next. In other words, knowing the past outcomes, the future outcome of a process is predicted [9].

The following definitions are considered in relation to a Markov chain:

Set of states, S = $\{s_1, s_2, s_3, \ldots, s_n\}$, where the process could start in any one of these states and would move (or transition) in order, from one state to another, with a given probability called the transition probability.

The state transitions happen at discrete times, known as steps.

If the current state is s_i, then the next state is s_j, with probability of p_{ij}. It is important to note that p_{ij} is independent of any states before s_i. Also, the probability of going from one given state to all the possible next states must add up to one. For example, if the current state is s_i and all the possible next states for s_i are s_n, s_m, and s_j with transition probabilities of p_{in}, p_{im}, and p_{ij} respectively, then:

$$p_{in} + p_{im} + p_{ij} = 1 \tag{1.1}$$

A collection of all the transition probabilities, probability of going from any given state to any possible next state in a process, can be represented as a matrix known as the transition probability matrix [9].

Next, a decision process in defined before the introduction of a Markov Decision Process (MDP). Any given sequential decision process has the following properties [10]:

1. Set of decision timesteps, this is a time frame where the decision is being considered (each state is within a given timestep)
2. Set of states, which was discussed above

3. Set of available actions. In the above example with current state of s_i, taking actions a_{in}, a_{im}, and a_{ij} make the next respective states s_n, s_m, and s_j possible.

4. Set of transition probabilities, which was discussed above, shows that from the state s_i, taking action a_{in} (resulting in state s_n) has the transition probability of p_{in}.

5. Set of state-action immediate reward/cost. This means that for every state and action pair, there will be an associated immediate reward (and/or cost, which for the remainder of this thesis will only be referred to as reward). So, s_i, taking available action a_{in}, will have reward r_{in}.

Put differently, at a given timestep and state, the decision maker (discussed at more length in the next paragraph) will decide on an action which will determine the next state as well as the reward for that state-action pair. For a general sequential decision process, the available actions and possible next states may be a result of the most recent state-action pair or all the previous states and actions that led up to the current state.

The decision maker is tasked with choosing a series of actions (given the current state and possible actions at every given timestep) that would maximize a set goal. In some cases, this goal may be to maximize the overall rewards. One way to achieve this goal could be to maximize the immediate reward at each timestep, also known as a decision rule for every timestep.

Now, a Markov Decision Process (MDP), is a sequential decision process with a certain current-memory property. This means, as shown in Figure 1.1, that only the current state and action of the agent will determine the next state (and possible reward and next actions) and not past states or actions. The only effect the past states and actions have on the next state, is indirectly, through the effect they had on the current state [10]. In Figure 1.1, the environment is referring to a collection of states.

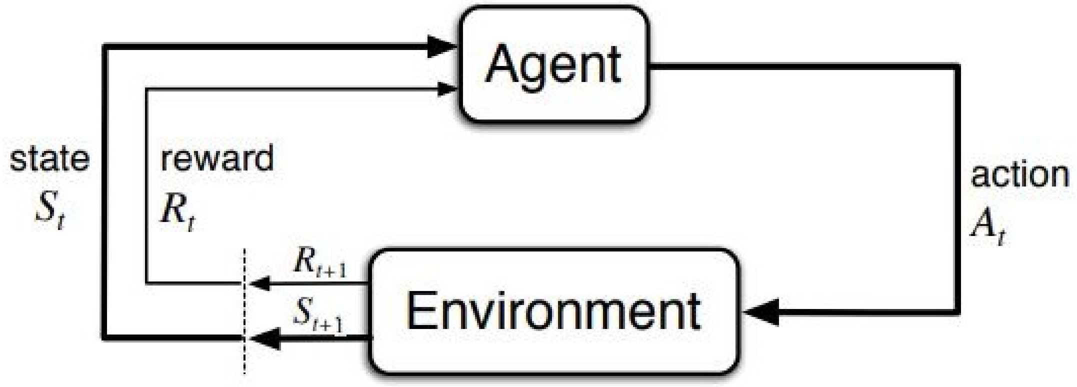

Figure 1.1 MDP setup showing the state-action relationship with reward [11].

As stated in the third chapter of [11], MDPs provide “a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards.”

1.5 Thesis Organization

So far, this chapter provided the motivation behind this thesis research, explained the problem’s setting, 5G wireless network’s challenges and opportunities, and included a brief explanation of MDPs.

The rest of this thesis is organized as follows:

- Literature Review: Chapter 2 introduces dynamic programming, the different techniques of machine learning, and specifics about Q-learning and deep Q-learning. Next, related work is covered, research in understanding and improving wireless mmWave networks using machine learning methods such as reinforcement learning, and deep reinforcement learning.
- Reinforcement learning in mmWave networks: Chapter 3 begins with a sample network used in explaining algorithms being considered in this research. Then, the three approaches used to improve the user’s service status, dynamic programming, Q-learning, and deep Q-learning, are discussed in more detail. These algorithms are explained using the example network introduced in the first section of the chapter,
- Numerical results: Chapter 4 presents results for the algorithms introduced in the previous chapter to showcase different network scenarios.
- Conclusion and Future Work: The last chapter includes a discussion of the results and proposed future work.

Chapter 2

Literature Review

This chapter initially provides a brief overview of different machine learning algorithms. This is followed by an explanation of methods being explored in the literature pertaining to the utilization of machine learning algorithms in wireless networks.

2.1 Machine Learning Overview

This section introduces the different methods used in solving the research problem. Two general methods are considered, dynamic programming and machine learning algorithms. Both approaches assume that the environment can be modeled as an MDP and can be used to find solutions. Different machine learning algorithms are introduced to provide context. The algorithm of choice in the solutions introduced to solve the research problem is reinforcement learning, and more specifically Q-learning. That is because the problem requires a solution that is model-free and able to find optimal policy () by experiencing the environment based on a reward system. The definitions of model-free and optimal policy in the context of this research will be discussed in the next section.

Deep Q-learning, a sub-category of reinforcement learning and deep learning, is introduced at the end of this section and can be used to approximate the solution, as opposed to finding exact solutions, for larger network sizes.

2.1.1 Dynamic Programming

Dynamic Programming (DP) can be applied to environments modeled as an MDP to find an optimal policy for the environment. Although it may not always be easy or even possible to have a perfect model of the environment, it is a necessity for DP algorithms. A finite MDP model of the environment, plus the higher computational cost of a DP algorithm are why these algorithms are not always used, or useful [11]. Therefore, this research utilizes other methods, as well as DP, to solve the problem at hand. It may be useful to note that a finite MDP refers to an MDP with a finite number of elements in the states, rewards, and actions of the environment [11].

In DP algorithms, the Bellman optimality equation is satisfied by choosing optimal value functions that would result in optimal policies. This is done by using the Bellman equations as an update rule for the state-value function. Computing the state-value function for any policy, π, (2.1)

$$v_{\pi}(s_t) = E_{\pi}[reward_{t+1} + \gamma \times reward_{t+2} + \gamma^2 \times reward_{t+3} + \cdots \mid s_t] \qquad (2.1)$$

Where E_{π} is the expected value, given that the policy, π, is being followed. γ is the discount factor and is explained at length in section 2.1.6. The state-value function will have a unique value if $\gamma < 1$ or if it is possible to reach the terminal state from any state in the policy. Then, as mentioned, using the Bellman equation on (2.1) to make the update rule would result in (2.2):

$$v_{k+1}(s_t) = E_{\pi}[reward_{t+1} + \gamma \times reward_{t+2} + \gamma^2 \times reward_{t+3} + \cdots \mid s_t] \qquad (2.2)$$

As $k \to \infty$ the sequence $\{v_k\}$ will converge to v_{π} , if $\gamma < 1$ or if it is possible to reach the terminal state from any state in the policy. This equation is called the iterative policy evaluation [11].

2.1.2 Machine Learning

Artificial Intelligence (AI), is referred to the field of programming computers to perform tasks that are natural and instinctive to humans, but not as easy for computers. In other words, AI is a blanket term used to describe any automatic machine reasoning agent. Machine Learning (ML) is a more specific area of AI more focused on pattern recognition and data deductions (learning from provided data) through experience. Artificial Neural Networks, or ANNs, are a subclass of ML that use what is known about the formation and operation of the human brain to perform machine learning tasks, hence the neural network name [12].

ML algorithms are generally divided into the following categories:

supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [11]. These categories of ML will be discussed at more length in the following sections of this chapter. Q-learning, a sub-class of reinforcement learning will be discussed followed by an account of deep Q-learning.

2.1.3 Supervised Learning

The first class of ML algorithms is supervised learning algorithms. In supervised learning, the input to the algorithm is the data as well as its expected output. The expected or target output, which is sometimes referred to as labels, is used to assess the “correctness” of the algorithm’s output. The algorithm then uses the input data to learn and find a pattern between the input and output data. In other words, it uses the input and output variables to iteratively make predictions and find a function that maps the two, so that it can use that function to predict the output of a new set of inputs.

Supervised learning problems are usually categorized as classification or regression problems. The main difference between these two categories is the end goal of the algorithm. The main objective in classification problems is to predict the correct label and it is used for discrete outputs. In regression problems the aim is to predict a quantity which is a continuous value [12].

An example of a classification supervised learning can be to label photos as having a “cat” or “dog” in the images. In this example, the algorithm will be trained on enough images of different cats and dogs (with the correct labels) that if a picture of a new cat was provided, it would be able to categorize it as a “cat” with a higher percentage of probability than a “dog”.

An example of regression supervised learning can be predicting house prices based on an area. It may also be possible to categorize this example as a classification problem, if the house prices are divided into blocks, so that any house with value of: $0 to $200k would fall into Category1 and $200k to $400k would fall into Category2 and so on.

The problem being pursued in this research is neither a classification nor a regression problem.

2.1.4 Unsupervised Learning

The next category of ML algorithms is unsupervised learning. In unsupervised learning the input data is provided, but no expected output data is given. The algorithm would use the provided data and draw conclusions from that data’s distribution, structure, features, and patterns. Unlike supervised learning, there are no provided target outputs, not even the number of classes in a dataset. This could be because of how expensive and time consuming it may be to make labelled datasets, or that the expectations (the relationships between the provided data) are unknown. Unsupervised learning algorithms are naturally more complex as compared to supervised learning algorithms, and their results may be less accurate [12].

Unsupervised learning problems can be broken down into two general groups: clustering and association. A clustering unsupervised learning example can be to group customers by their shopping habits. And an association unsupervised learning example could be to find relationships between shoppers, so that it can be said that people buying one specific item (toast bread) tend to buy a different item (butter) as well.

Unsupervised learning is also not a suitable approach to tackle the problem motivating this work.

2.1.5 Semi-Supervised Learning

The third group in ML algorithms is semi-supervised learning. As is clear from the name, semi-supervised machine learning problems are provided with large number of input data for which only some of the data has been labelled [12]. In the “cat” and “dog” photo example from the supervised learning section, if the photos are only partially labelled and some are just pictures with no labels, then it can be considered a semi-supervised learning problem. Like unsupervised learning datasets, the partial labels in the semi-supervised problems could be because of how expensive and time consuming it would be to label or store the entire dataset.

Semi-supervised learning algorithm is not an eligible approach that can be applied to the problem in this research, because there is no labelled data whether fully or partially labelled.

2.1.6 Reinforcement Learning

The last category, and most useful to this research, of ML algorithms is reinforcement learning. In Reinforcement Learning (RL) algorithms the idea of a labelled dataset is no longer useful. That is because these algorithms use a reward system to choose actions and train on the provided data, following an explicit goal. A very popular use of reinforcement learning algorithms is in games (video games or board games such as chess) because the reward system is already part of the fabric of the problem. The point system in these settings provide the learning agent with appropriate feedback to perform better over time. In these algorithms, as is mentioned in [13], the learning agent can be an animal, a human, or a computer program and the reward could be any measure of performance for the agent such as food, water, and money. This shows the importance of choosing an appropriate reward system, since it is the reward that reinforces the action.

The reason for mentioning MDPs in the previous chapter will now be more evident. The similarities between Figure 1.1 and Figure 2.1 show that an RL algorithm is based on the fundamentals of an MDP. At a closer look it is clear that the two figures are expressing the same message about the role of action in determining next states and reward values.

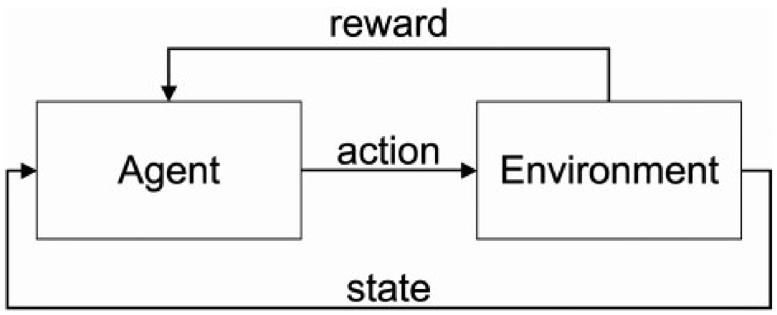

Figure 2.1. Typical reinforcement learning cycle [13]

In Figure 2.1, environment is referring to the collection of all the possible states of the agent. For instance, in a game of tic-tac-toe, the environment would be all the 9 spaces on the board, or the 64 squares on a chess board. In these algorithms, the agent will perceive the environment's state and choose an action based on the transition probabilities, or policy, which can sometimes be represented as a lookup table. The agent's action determines the next state of the agent and its reward (both given with a certain probability, given the agent's current state and the chosen action) and updates the policy accordingly [13].

As mentioned before, reinforcement learning is grouped separately from supervised or unsupervised learning. One of the challenges with supervised learning is to choose data that is diverse enough to cover all different possible characteristics of the environment the agent is expected to learn about. This is not an issue in reinforcement learning scenarios since the agent is able to draw its own conclusions from the environment and learn from its own experiences. In unsupervised learning the agent is usually looking for patterns in the provided unlabelled data, and that is different from the reward maximizing nature of reinforcement learning [11].

The goal of any reinforcement learning algorithm would be to find the best policy that would maximize the overall reward of the system. Maximizing the overall reward, is done by predicting the expected cumulative future reward, maximizing (2.3):

$$E[reward_t + \gamma \times reward_{t+1} + \gamma^2 \times reward_{t+2} + \cdots] \qquad (2.3)$$

Where t represents the current timestep, $reward_t$ is the reward at timestep t, and γ is the temporal discount factor or discount factor for short. Discount factor is how far into the future the reward is being considered, and has a value in the range [0, 1]. The smaller the discount factor, the shorter the vision of the algorithm, so $\gamma = 0$ is only considering immediate rewards and $\gamma = 1$ is considering all the available rewards. Choosing the right value for the discount factor is considered one of the challenges of applying reinforcement learning algorithms to different problems [13].

An essential feature that sets reinforcement learning apart from other machine learning algorithms is the exploration and exploitation question. In reinforcement learning, the learning agent will need to explore actions and find ones that would maximize reward. But it would also require exploiting the actions it has already tried and found to be effective at achieving its goal. It is important that the agent utilizes a mix of the two approaches in order to effectively maximize reward on the tested and proven actions, but to also try new actions that may result in higher rewards. Because of the stochastic nature of most problems that these algorithms are applied to, it is also vital to repeat the actions to obtain dependable values of the expected rewards [11]. This process is referred to as ε-greedy, since the ε variable determines the probability of taking a random action.

2.1.7 Q-learning

Q-learning is a popular model-free reinforcement learning algorithm where the learning agent does not need to have prior knowledge of the system model parameters such as the state-transition and reward models. In fact, Q-function is used to approximate the values of state-action pairs through the agent's interactions with the environment, during training [14].

The agent learns the action value function, $Q(s, a)$ or Q-value, showed in (2.4) [13]:

$$Q(s_t, a_t) = E[reward_t + \gamma \times reward_{t+1} + \gamma^2 \times reward_{t+2} + \cdots \mid s_t, a_t] \qquad (2.4)$$

In the calculation of the Q-value, the expected cumulative future reward is considered given the current state, s_t, and action, a_t. While the $reward_t$ is the immediate reward of the environment's state for this timestep, the $Q(s, a)$ is the total amount of reward the agent can expect to get in the long run, given the current state. This means that a high immediate reward would not necessarily guarantee a higher overall reward. It is possible for a state to have high immediate reward, but low overall reward, or vice versa [11]. Additionally, the value function considers the expectation of all actions according to the policy, π, while the Q-function examines a given action at a given state [14].

The recursive relationship between the Q-value of the current and next steps, (2.5), is used to evaluate the action taken in the current step, a_t. This evaluation is then utilized in choosing the best action, a, in the next state, so that the policy, or the lookup table, can be updated without having to wait for all the future reward values.

$$Q(s_t, a_t) = E[reward_t + \gamma \times max_a Q(s_{t+1}, a)] \quad (2.5)$$

In (2.5), it is shown that the current Q-value is only dependent on the value of the immediate reward and the discounted best action in the following state [13]. In (2.6) the action-value function, Q-value, can predict the optimal Q-value, independent of the policy [11]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[reward_{t+1} + \gamma \times max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (2.6)$$

Where α is the learning rate, also known as step-size in some literature, with a value in the range of [0, 1]. As can be gathered from the name, α influences the rate at which learning is happening. If learning rate is reduced over time, then the actions taken by the learning agent will become optimal, if the exploratory moves are ignored. On the other hand, if the learning rate does not lower to zero over time, then the learning agent can even adapt to an environment with states that would have different actions available at different timesteps [11]. This means that if the learning agent was trying to beat a real player, that did not always stick to the same game strategy, it would have a better chance at foreseeing the changes, if these changes are slow enough.

Algorithm 2.1 shows a breakdown of the steps in implementing a Q-learning algorithm. These are the same steps followed in writing the program for the Q-learning solution introduced in the next chapter, included in [15].

Algorithm 2.1 Q-learning algorithm for estimating $\pi \approx \pi_*$ [11]

```
Loop for each episode:
    Initialize state
    Loop for each step in this episode:
        Choose action considering state, using policy derived from Q (could be using ε-greedy)
        Take action (a), observe reward (R) and next state ($s'$)
        Update Q value $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma max_a Q(s',a) - Q(s,a)]$
        Update state with the next state ($s \leftarrow s'$)
    Until terminal state has been reached
```

In Algorithm 2.1, π_* is the optimal policy. Also, each episode goes on until the terminal state. In a video game, this is usually when the agent reaches the final state in the game, either winning or dying in the game could be a possible end terminal state. In the solution introduced in the next chapter, the terminal state for a user in a network is the goal set for the user at the start of training. The initial state of the agent is re-initialized after the end of each episode to a certain state in the environment, and considering that all states in the environment are expected to be able to reach the terminal state at some point, this is acceptable.

At this point, before moving to the deep Q-learning section, it is useful to define the term off-policy. In Q-learning algorithm, policy is updated based on the maximum reward available to actions, regardless of the policy. Therefore, Q-learning is referred to as an off-policy temporal difference (TD) learning algorithm [14]. TD learning is a combination of Monte-Carlo and dynamic programming theories [11]. More in depth explanations about TD and Monte-Carlo methods can be found in [16].

2.1.8 Deep Q-learning

Deep Q-Learning (DQL) is a cross-section between deep learning (DL) and Reinforcement learning's Q-learning. Deep learning refers to training networks with more hidden layers (more than one) that are able to learn in order, meaning that simple concepts are learned in the lower layers and more abstract concepts in the higher layers of the deep network [12]. The weighted combination of hidden units, their non-linear activation functions, and the output units are referred to as layers [17].

One of the advantages of considering DQL algorithms (because it is a Reinforcement learning algorithm) is that there is no longer a need to have a finite MDP, to achieve an optimal policy. Deep Q-Networks (DQNs) can approximate an optimal policy for environments with very large or even infinite state spaces [11]. In [18], the authors combine an RL method, Q-learning, with deep learning in order to obtain control policies directly from the states, actions, and rewards of the environment. There are several issues, in no particular order, that are raised by combining RL and Q-learning methods:

1. Delay between actions and resulting rewards (which in some cases may be thousands of timesteps [18]) in RL algorithms
2. Deep learning algorithms assumption of independent data samples, while the input states in RL algorithms may seem highly correlated
3. Changes in data distribution over time in RL vs. fixed underlying distribution in DL

In [18] these issued are addressed by utilizing a convolutional neural network, experience replay memory, with stochastic gradient descent to update the weights of the neural network.

Experience replay memory will randomly select transitions already visited by the learning agent, so that the correlation between the data and changes in data distribution are smoothed over.

DQL is used as the final approach in solving the research problem of this work. This solution will approximate the optimal policy but is applicable to larger networks that dynamic programming or Q-learning fail to solve.

Before exploring these solutions in the next chapter, it is important to investigate the work that has already been done in relation to applying ML algorithms to wireless communication networks, in the next section.

2.2 Applications of RL in Wireless Networks Literature

This section presents a survey of previous research in the applications of machine learning methods in the field of communications and networks. In particular, the work on the application of RL and deep RL in the area of wireless communication networks, 5G, has been summarized. The order in which the articles are presented is not indicative of the importance or relevance of their respective research to the problem being addressed in this work.

Utilizing Reinforcement learning algorithms has become very popular in the recent years. A diverse group of problems from training agents to beat humans at games [18], resource management in computer systems [19], speech recognition [20], to applications in wireless network communications [21] use deep reinforcement learning (DRL) techniques to maximize their intended goal for their respective environments. Some of the research using RL algorithms that has motivated or guided the work in this thesis are as follows.

In recent years, because of the ever-growing mobile data usage, there has been an exponential need for mobile network infrastructure development. As a result, network energy consumption has grown, bringing up a need for capping or controlling the energy consumption for various environmental and economical concerns [21]. Studies have shown that base stations (BSs) consume 60 – 80% of the total energy consumed by a network [22], and so it seems like a reasonable source of power usage to manage in order to lower overall network power usage. One of the ways to manage base station energy utilization has been BS sleeping, where the BSs with lower traffic volume are placed in a low-power sleep mode, with different duration sleep schedules [21]. While faster or slower sleep cycles are chosen based on the network utilization, it is important to keep in mind the quality-of-service (QoS) criteria such as delays and outages when deciding on the length of the sleep cycles. In [21], a dynamic BS sleeping algorithm based on deep RL is proposed, with action-wise experience replay to stabilize the different traffic phases (high traffic phase vs. lower traffic phase). An adaptive reward scaling technique to rescale rewards received from mobile networks is taken advantage of, to match the output network's response range, so these rewards are clipped to values in the [-1, +1] range. The main reason for using a deep Q-learning algorithm is to approximate the action-value function instead of visiting every state-action pair to find exact values for the action-value function, because of the higher dimensions of their state spaces. Utilizing DQNs with experience replay and adaptive reward scaling has proven to have more sleeping gain, time averaged per step reward, than regular Q-learning or DQN [21].

Because of the sensitivity of mmWave signals to blockages and possible disconnection and re-connection delays, reliability and latency are the main issues with mmWave wireless networks. In [3], uses a deep learning algorithm to predict a possible blockage accurately and hand the mobile user to the other BS without disconnecting the session or incurring high latencies. Only two available BSs are considered to service a mobile user, so the assumption is that if one BS is NLOS then the other must be in LOS of the user.

Traditional network resource allocation methods rely on modeling the network environment and user demands [23]. In the complex networks of today with random and unpredictable user behaviours, it would be difficult to model and predict these networks accurately. A basic networking problem, traffic engineering, which is to maximize a utility function by forwarding the data traffic in a set of source and destination nodes, is addressed in [23] with a model-free approach using a variation of deep reinforcement learning algorithm. The model-free approach addresses the issue with traditional network resource allocation methods, and the DRL algorithm is also able to deal with sophisticated state spaces (that includes throughput and delay of each communication session). The deep deterministic policy gradient used in this work makes it possible to apply DRL algorithm to the continuous control problem of traffic engineering. This method significantly lowers end-to-end delay, improves network throughput, and easily adapts to network changes as compared to other widely used network resource allocation methods [23].

Using mmWave and μWave resources together in a dual-mode framework could be a solution to overcoming the blockage concerns in mmWave networks, while taking advantage of the available bandwidth at both mmWave and μWave frequency bands [24]. In [24] the μWave band, the user application selections and scheduling is set as a one-to-many matching game between the μWave and user applications. Then the mmWave band utilizes a Q-learning algorithm to find a scheduling policy while also finding the user equipment's LOS probabilities. The two bands are working together to improve overall user QoS in delay-sensitive applications such as HDTV and videoconferencing. The goal is to service as many users on the mmWave band as possible, to lighten the load on the μWave band, while keeping in mind the delay constraints of each user's needs. Because of the dynamic nature of the environment, the LOS probability of a user and BS may not just be calculated and averaged. So, a Q-learning framework with changing LOS probabilities is proposed to address the mmWave band base stations. This dual-band context-aware scheduling approach has proven to show performance improvements compared to proportional fair (with minimum rate requirements) and round robin schedulers [24].

Because of the exponential increase in the number of devices that stream data, traditional routing protocols are challenged since they base their calculations on shortest path from source to destination without considering network state parameters such each router's remaining buffer size [25]. To solve this issue, [25] utilizes DRL algorithms for router selection. Two methods are proposed, categorizing all data transmission tasks into two groups: one with source routers and destination routers (source-destination multi-task deep Q-network or SDMT-DQN) and the other with just the destination routers (destination-only multi-task deep Q-network or DOMT-DQN). These two methods proved to lower congestion probability and increase network throughput compared to regular deep learning algorithm and Open Shortest Path First (OSPF) routing protocol as the number of training steps are increased [25]. While SDMT-DQN would require more neural networks than DOMT-DQN, both methods have significantly fewer number of neural networks than regular deep learning. Having (only a few) greater number of neural networks will incur a slight advantage where SDMT-DQN has a lower congestion probability and higher throughput as compared to DOMT-DQN [25].

Q-learning algorithm can find optimal policies when the state-action space is small, but in more realistic scenarios these state-action spaces are too large to use Q-learning algorithms successfully. Therefore, deep Q-learning techniques with two key features are used in practical, more complicated models. As explained in [14], these two features are experience replay and a target Q-network. To implement experience replay, the algorithm updates the replay memory that is initially populated with randomly generated values, with transitions of the environment (s_t, a_t, r_t, s_{t+1}) where t is the timestep, and $s, a,$ and r are the state, action, and reward respectively. The algorithm will then randomly select a number of samples (transitions) from the replay memory, called minibatches, and train. This will allow for a diverse training on old and new observations, as well as eliminating correlations between transitions since they are more independently distributed [14]. The target Q-network feature is to ensure the stabilization of the algorithm, by constantly updating the target Q-network value and slowly updating the primary Q-network value. A deep reinforcement learning algorithm with the two mentioned features is in Algorithm (2.2)

Algorithm 2.2 Deep Q-learning algorithm with experience replay and fixed target Q-network [14]

```
Initialize replay memory
Initialize the Q-network, $Q$, with random weights, $\theta$
Initialize the target Q-network, $\hat{Q}$, with random weights, $\theta'$
For all episodes do:
    Choose action considering state, using ε-greedy (with probability ε select a random action,
    otherwise select $a_t = argmaxQ^*(s_t, a_t, \theta)$ )
    Take action $a_t$: observe immediate reward $r_t$, and next state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay memory
    Select random samples $(s_j, a_j, r_j, s_{j+1})$ from replay memory
    The weights of the neural network are optimized by using stochastic gradient descent with
    respect to the network parameter $\theta$ to minimize loss:
        $[r_j + \gamma max_{a_{j+1}} \hat{Q}(s_{j+1}, a_{j+1};\ \theta') - Q(s_j, a_j;\ \theta)]^2$
    Update $\hat{Q} = Q$ after every fixed number of steps
End for loop
```

By now, it should be clear that deep learning, and more specifically deep reinforcement learning techniques, can be promising for solving mobile networking problems [17]. One of the main advantages of these algorithms is that they can deduce high-level features from complex structured data [17]. Deep learning, by definition, is able to handle large amounts of data and while the performance of other machine learning algorithms may not improve much by increasing the data size, deep neural networks could even prevent model overfitting by training on larger datasets [17]. Also, with the development of ML dedicated toolboxes such as Tensorflow or (Py)Torch, the need for building deep learning models from scratch has been eliminated [17].

Chapter 3

Reinforcement Learning in mmWave networks

In this chapter the algorithms used in finding an optimal policy for the problem proposed in chapter 1, are discussed. These solutions are introduced in the order they have been developed, which is also the order in which the complexity of the methods increase. After the introduction of a sample network for the problem, the dynamic programming solution is covered. This solution is only applicable if the user's path is already known (data rates received from every BS along the user's path is already calculated). The goal of the program is to find an optimal list of BSs for the user to connect to that would ensure service continuity, while the user is within the network boundary.

Next, is the Q-learning approach. In this method, given a certain goal/destination for the user (inside the network) the program will return an ideal path for the user to take to reach the goal, as well an optimal list of BSs similar to the dynamic programming solution. This technique will work for small networks (where the state-action space can be mapped to the Q-matrix).

The last method being considered is the DQL algorithm, where an exact solution to the problem may not be found but every state-action pair does not need to be visited in order for the approximation algorithm to predict an optimal path and BS list for the user, given a goal/destination inside the network.

3.1 Introduction of a Sample Network

A sample network is introduced in this section to facilitate the explanations made in the following sections of this chapter. The following network size assumptions are made for simplicity in the examples. This does not mean that the solution only applies to a network with these measurements.

A 150×150 square shaped network is assumed, with the bottom-left corner of the network at $[-60, -60]$ and the top-right corner at $[+90, +90]$, as shown in Figure 3.1.

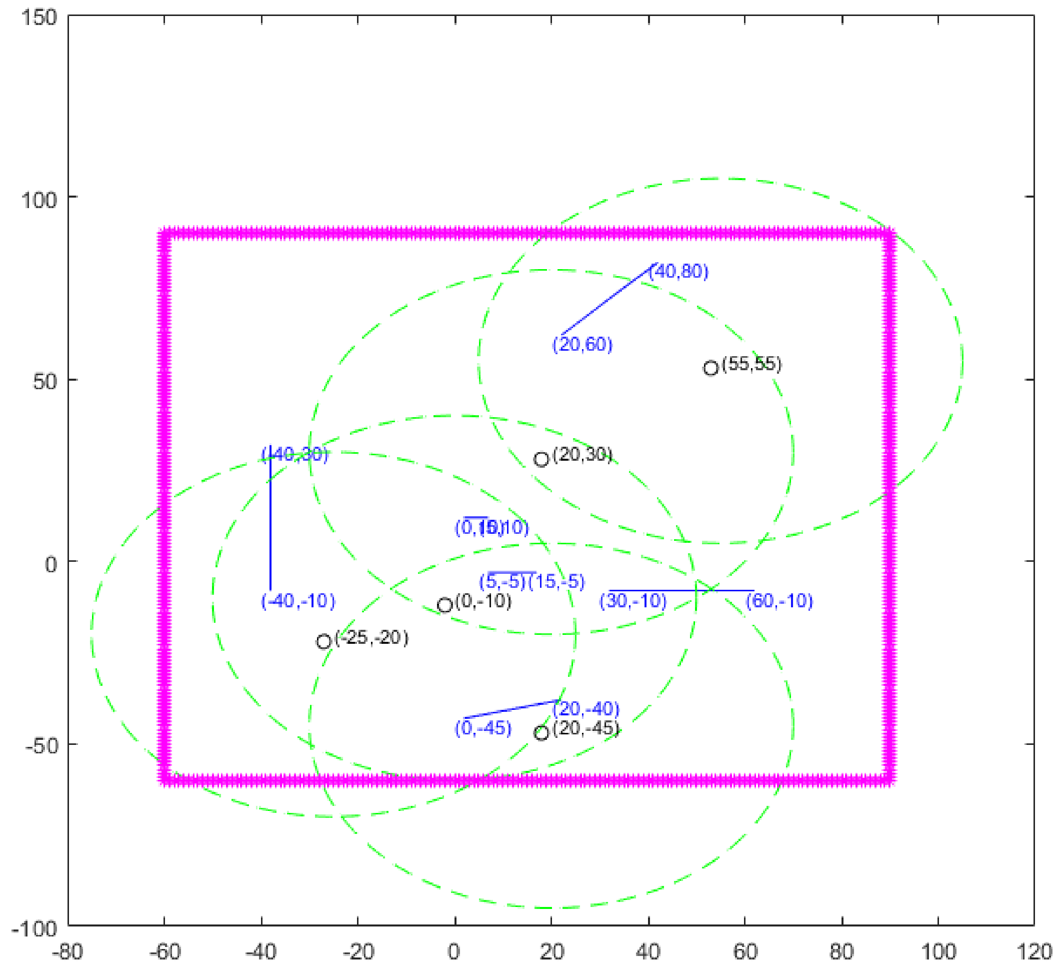

Figure 3.1. Sample 150 × 150 *network with 5 base stations and 6 blockages*

In Figure 3.1, the magenta star-shaped lines represent the boundaries of the network. Each black circle is the location of a randomly located base station, with the coordinates of the BS written in the same color. Even though a cell boundary may be unclear in mmWave signals because of the sensitivity of these signals to blockages [4], a dense mmWave tier network ("A network is dense when the infrastructure density is comparable to the blockage density" [26]) with an average cell radius of 50 m [26] (50 units on Figure 3.1) is assumed, where the green-dashed lines show the coverage area of each BS. The blue-solid lines are depicting blockages, assuming linear two-dimensional blockages, with the coordinates of each end of the blockage line represented in the same color blue. In Figure 3.2, a closer view of the network, bounded by the network's dimensions, can be seen.

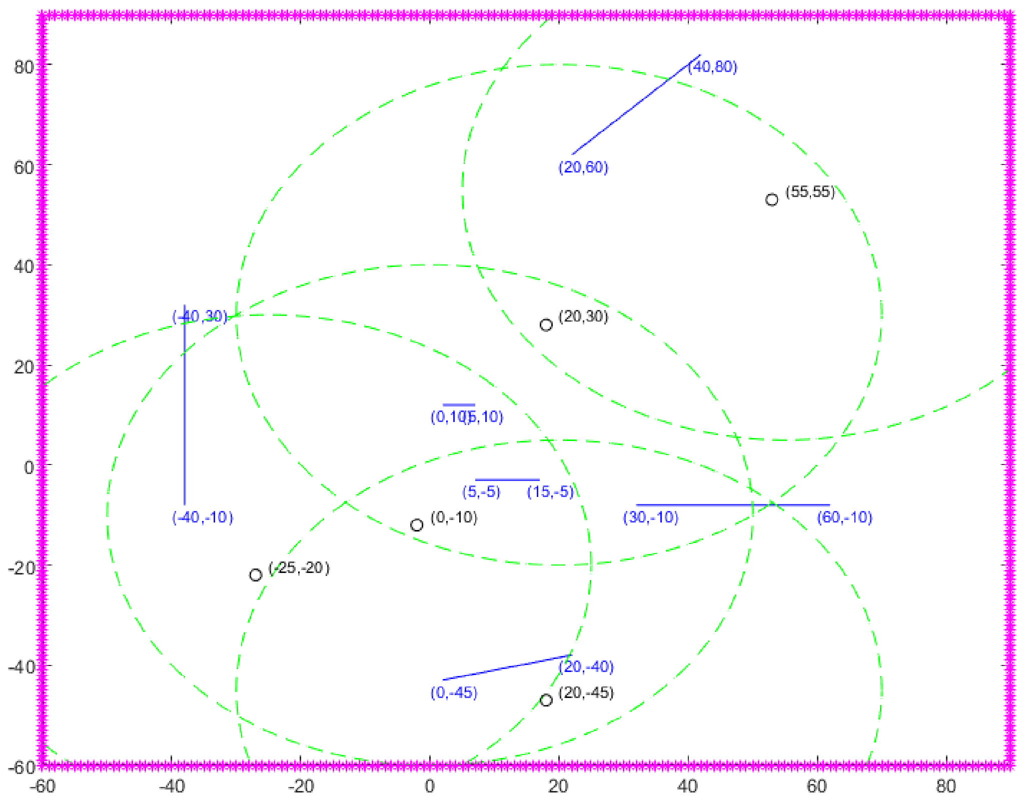

Figure 3.2. Bounded snapshot of the network in Figure 3.1

The size of one step for the user in this network is expected to be 5 units (5 marks on the x-axis or y-axis on Figure 3.2). At each step, the user can move in one of the four primary directions: up, right, down, or left (the user cannot take diagonal steps). It is assumed that the user can step on the BSs and through or on blockages.

When referencing a particular BS or blockage in the remainder of this chapter, Table 3.1 and Table 3.2 will provide the references, respectively.

Table 3.1 Base station coordinates of the network in Figure 3.2

BS number	**BS coordinates in Figure** 3.2
BS0	[-25, -20]
BS1	[0, -10]
BS2	[20, -45]
BS3	[20, 30]
BS4	[55, 55]

Table 3.2. Blockage coordinates for the network in Figure 3.2

Blockage number	**Blockage coordinates in Figure** 3.2
BL0	((-40, -10), (-40, 30))
BL1	((0, 10), (5, 10))
BL2	((5, -5), (15, -5))
BL3	((0, -45), (20, -40))
BL4	((30, -10), (60, -10))
BL5	((20, 60), (40, 80))

3.2 Dynamic Programming

This section discusses the details in implementing a dynamic programming solution to the research problem. This solution is applicable to any network size, but it may take a very long time for the calculations to be done on larger networks. Although this solution will result in best possible base station (BS) connection lists, the lengthy processing times for realistic network sizes is the reason other approaches to solving the problem were considered. Furthermore, this solution is only applicable to 1-dimensional network spaces, which means that the user's path is pre-determined for this method. But it can be applied to any user path, given that data rates from each BS along that path are known. The result of this program is a list of BSs that the user can connect to and expect to stay connected to for the entirety of pre-determined path length, achieving the highest possible data rates.

This solution assumes a pre-determined fixed path from source to destination (assuming $[-60, -60]$ to $[90, 90]$). So, the problem becomes finding the "best" list of base stations to connect to, given this fixed path. To define "best" list BS, it is important to note that the BS with the highest data rate at each given step is not necessarily going to result in the optimal BS list. It is the overall highest reward (data rate and switching overhead) of all the steps on the path that would result in an optimal BS list.

To clarify this, lets assume that the user is at [0, -40], in Figure 3.2, where the highest data rate is received from BS1, $datarate_{01}$ (data rate for user location 0, received from BS1). If the user takes a step to the right, [5, -40], then BS1 will no longer be LOS, and at the value of $datarate_{11}$, will not provide the highest data rate. Assume that at [5, -40] the highest data rate will be received from BS2, $datarate_{12}$. Then, if the user were to take a step up, to [5, -35], BS2 will be NLOS, having data rate value of $datarate_{22}$, and BS1 will provide the highest data rate again, at $datarate_{21}$. Considering the overhead associated with switching BSs, switching of BSs twice in two steps would be justifiable only if:

$$datarate_{12} - datarate_{11} > switching\ overhead\ cost \tag{3.1}$$

and

$$datarate_{21} - datarate_{22} > switching\ overhead\ cost \tag{3.2}$$

There even may be a third BS that would result in the higher data rate in these three steps, if it meant for the user to stay connected to the same BS, hence not incurring any switching costs/penalties. To simplify the decision-making process for similar situations, a constant value is assigned to the overheard cost of switching from one BS to another.

Therefore, "Best" list of BSs can be defined as the list of BSs that would result in the highest Reward. And immediate Reward for all steps in a list would be defined as in (3.3):

$$Reward = \sum_{all\ steps} datarates\ received\ from\ each\ BS\ on\ the\ list - (number\ of\ BS\ switches\ \times switching\ cost\ constant) \tag{3.3}$$

In order to find the best BS list, for a series of *i* steps in a network with *k* number of BSs, where:

$$l < m < m + 1 \ \leq i \tag{3.4}$$

and bs_i will denote the BS that the user is connected to at step *i,* the following lemma is introduced.

Lemma: if B_m is the best list of BSs for all steps up to and including step *m*, then B_l is the best list of BSs for all the steps up to and including step *l*, if there are no BS switches between steps *l* and *m*.

Proof: Assume that there is a list, $B_l{}'$, that would have a higher overall reward ($Reward_{B_l}{}'$) than B_l ($Reward_{B_l}$) where they have the same steps, but a different list of BS connections:

$$Reward_{B_l}{}' = \sum_{l\ steps} datarates\ received\ from\ each\ BS\ on\ B_l{}'\ list - (number\ of\ BS\ switches\ \times switching\ cost\ constant) \tag{3.5}$$

and

$$Reward_{B_l} = \sum\nolimits_{l\ steps} datarates\ received\ from\ each\ BS\ on\ B_l\ list - (number\ of\ BS\ switches\ \times switching\ cost\ constant) \tag{3.6}$$

Given that the reward for part of the B_m list, only for all the steps between l and m (including step *m* but not including step *l*) is known as S_{m-l}:

$$Reward_{S_{m-l}} = \sum\nolimits_{(m-l)\ steps} datarates\ received\ from\ BS\ on\ steps\ l\ thru\ m \tag{3.7}$$

Then, the reward for all of B_m is:

$$Reward_{B_m} = Reward_{at\ step\ m(with\ B_l)} = Reward_{B_l} + Reward_{S_{m-l}} - switching\ cost\ constant \tag{3.8}$$

In the $B_l{}'$ list, there are two possibilities for the BS at step *l*, $bs_l{}'$:

1. BS at step l is the same BS as the BS in step m, $bs_l{}' = bs_m$, so that there is no switching cost incurred:

$$Reward_{at\ step\ m(with\ B_l')} = Reward_{B_l}{}' + Reward_{S_{m-l}} \tag{3.9}$$

And, from comparing the right-hand side of (3.8) and (3.9), it is clear that:

$$Reward_{at\ step\ m(with\ B_l')} > Reward_{B_m} \tag{3.10}$$

Equation (3.10) is contradicting the assumption that B_m is the best list of BSs for all steps up to and including m.

2. BS at step l is a different BS than the BS in step m, $bs_l{}' \neq bs_m$, so there is a switching cost,

$$Reward_{at\ step\ m(with\ B_l')} = Reward_{B_l}{}' + Reward_{S_{m-l}} - switching\ cost\ constant \tag{3.11}$$

And by comparing the right-hand side of (3.8) and (3.11), it can be seen that:

$$Reward_{at\ step\ m(with\ B_l')} > Reward_{B_m} \tag{3.12}$$

Equation (3.12) is contradicting the assumption that B_m is the best list of BSs for all steps up to and including m.

Now that the introduced lemma is accepted, it can be used to decide on the BS for step *m+1*, given that B_m and B_l are known.

The BS for step m+*1*, bs_{m+1}, can be any of the *k* available BSs. So, there can be three possible values for bs_{m+1}:

1. bs_{m+1} is the BS with the highest data rate for the user at step m+*1*:

$$Reward_{B_{m+1}} = Reward_{B_m} + datarate_{bs_{m+1}} - switching\ cost\ constant \tag{3.13}$$

2. bs_{m+1} is the same BS as the BS at step m, $bs_{m+1} = bs_m$ (this can be considered a special scenario of the third case):

$$Reward_{B_{m+1}} = Reward_{B_m} + datarate_{bs_{m+1}} \tag{3.14}$$

3. bs_{m+1} is the same BS as the BS at step m, $bs_{m+1} = bs_m$, when they have been both changed to a different BS than previously made B_m the best list. The change in the BS list could go back as far as step *l*, but because of the lemma there would be no need to change the BS list past prior to step *l* (including step *l*). The change is assumed to have gone as far back as step *a* where:

$$l < a \ \leq m < m + 1 \tag{3.15}$$

So, looping over all the possible values of a, the a-dependant reward is calculated at:

$$Reward_{B_{m+1}} = Reward_{B_l} + (datarates_{bs_{l\ thru\ a}} + datarates_{bs_{a\ thru\ m+1}} - switching\ cost\ constant) \tag{3.16}$$

Then, the values from (3.13), (3.14) and (3.16) are compared, and the BS list with the highest reward will be chosen as the new best BS list for B_{m+1}.

Going about finding the best BS list in this manner for all the steps thru *i* will result in the highest-reward-yielding BSs for the user to connect to, given a fixed path. This dynamic programming approach results in exact BSs list that would maximize overall reward. The code for this program can be found in [15].

3.3 Q-learning

As mentioned in the previous section, the dynamic programming solution would find the list of highest-reward BSs, given a fixed path. If every path from user's starting point to its destination are to be studied, the program would need exponentially longer running times. The time needed to explore all of the user's paths and possible BS lists would be even longer, in a larger network. In an attempt to find solutions less time consuming and still as reliable, a Q-learning algorithm solution was pursued. The Q-learning method is predicting the optimal user path and BS list, given a destination for the user inside the network. This is done by choosing actions that yield the highest overall reward, based on the γ variable, discount factor, discussed in chapter 2.

In this algorithm, each step of the user was dealt with as a state in an MDP and a state-action Q-table was produced to predict the highest long-term reward BS list for a given goal.

The following definitions are considered in the design of the Q-learning algorithm.

State: The state of the system is determined by the current *x-y* coordinates (location at timestep *t*) of the user as well as the user's previously tagged BS. The previous tagged BS for a user at $[x_t, y_t]$ is the BS providing service to the user at timestep *t-1*. The previous tagged BS for a user at timestep 0, initial state of the user, is assumed to be the BS providing the highest data rate to the user at timestep 0. State would be in the form of (3.17).

$$S_t = [x_t, y_t, bs_{t-1}] \tag{3.17}$$

The state space of the user's locations would be all the possible integer locations in the network.

Action: At each given location inside the network, the user has two actions to decide for:

1. Action BS: refers to the BS serving the user at timestep *t*. This value can be any of the 5 BSs listed on Table 3.1.

2. Action direction: refers to the direction the user will take to reach the goal. This can be any of the four primary directions, as long as the user will stay within the network's borders after taking said direction. Action's direction value determines the user's next timestep's *x-y* coordinates.

The combination of the two actions would result in an action space that would be in the form of (3.18).

$$A_t = [action\_BS_t, action\_direction_t] \tag{3.18}$$

Reward: For each action of the user there exists a reward, given the state of the user. This reward value, (3.19), is calculated using the BS portion of the action, and the previously tagged BS part of the state. If $action\_BS_t \neq bs_{t-1}$ then there will be a switch in the BS serving the user, from bs_{t-1} to $action\_BS_t$, with a switching cost. This would mean that the immediate reward of the user for timestep *t* will be calculated using (3.19).

$$R_t = \ datarate_{action\_BS_t} - switching\ cost\ constant \tag{3.19}$$

But, if $action\_BS_t = bs_{t-1}$ then the same BS will continue to service the user at timestep *t*, and immediate reward will be calculated using (3.20).

$$R_t = datarate_{action\_BS_t} \tag{3.20}$$

The switching cost constant, the value assigned to the overhead in user hand-off, is chosen so that the effect of the data rate in the calculation of (3.19) is not diminished. It is important to choose a value that is not too low and can correctly represent the overhead cost associated with user hand-off. This value should also not be too high, because considering the reward calculated in (3.19) the learning agent will be reluctant to an action that would hand the user off from one BS to another and receive smaller reward in return.

To motivate the Q-learning algorithm to find the shortest path to the goal, the reward value for the goal state (i.e. location $[90, 90]$ in Figure 3.2) is set to a large constant integer. The constant is changed as the size of the network changes, to accommodate for the backpropagation of the reward in (2.5), for a given discount factor.

Using (2.6), the Watkin's Q-value update equation, and choosing a discount factor value close to one (between 0.8 and 0.9) and learning rate of 1.0, the long-term Q-value is calculated for every state-action pair and updated in the Q-table. The requirements for correct convergence of the Q-learning algorithm, that every state-action pair is visited and updated, are met [11].

This Q-learning approach results in exact user's shortest path to the goal and BSs list that would maximize overall reward, if all the state-action pairs of the environment have been visited by the learning agent. The code for this program can be accessed in [15].

3.4 Multi-Layer Perceptron and Deep Q-Learning

The final approach used to find a solution to the blockage problem in mmWave signals, takes advantage of function approximators to closely estimate the optimal pattern of a solution, as opposed to finding exact solutions. One of the short comings of Q-learning algorithms can be size constraints, where it can not be applied to environments with large state-action spaces. This is the reason that most research on applying ML to communications problems utilize Deep Q-learning algorithms, a function approximator, for larger networks. For the purposes of this research, it was deemed necessary to try function approximators for larger, more realistic networks. For example, in the 150×150 network of Figure 3.2, there are $30 \times 30 = 900$ possible user locations (because the user takes steps of size 5 units and $\frac{150}{5} = 30$). From Table 3.1, assuming 5 available BSs, and the state definition in (3.17), there are a total of $30 \times 30 \times 5 = 4500$ states in the network. From the action definition in (3.18), 5 available BSs, and 5 directions for the user to choose from (four primary directions of UP, RIGHT, DOWN, and LEFT and the option of no movement – STAY only for when the user has reached its goal location), there are a total of $5 \times 5 = 25$ actions for each of those states. So, the Q-table for this example would be a matrix with $states \times actions = 4500 \times 25$ rows and columns. This is already a very large and computationally expensive matrix, and the size of this example network is not even realistically large. To address this, a solution with a function approximator, more specifically Multi-Layer Perceptron (MLP), shown in Figure 3.3, was studied.

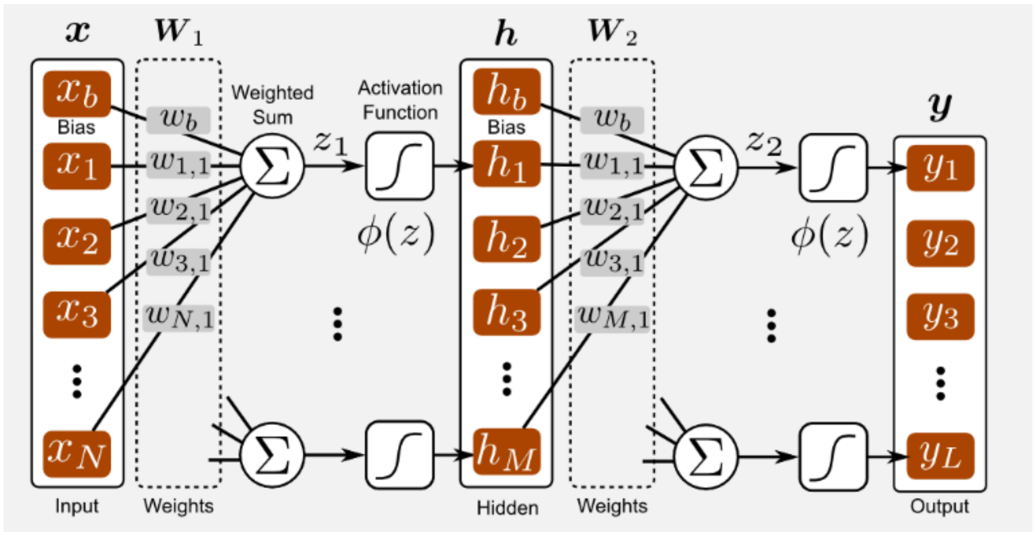

Figure 3.3. Layout of an MLP with N inputs, M hidden units, and L outputs [27]

Evaluating larger networks are made possible by eliminating the need to populate Q-table values directly, and instead approximating the values using a function approximator with 5 input neurons, one hidden layer, and 1 output layer. The 5 inputs are the states and actions of the Q-learning table, $[x_t, y_t, bs_{t-1}, action_{BS_t}, action_{Direction_t}]$. These values are fed into a fully connected neural network, with sigmoid activation function. It is important to use a non-linear activation function at either the input or the hidden layer steps, because if a linear activation function is used, then the MLP is just a perceptron (the simplest learning machines) [28].

In the feed-forward neural network of Figure 3.3, an MLP, inputs are multiplied by weights, W1, added to the biases, and then passed to the sigmoid activation function. This process is shown in (3.21):

$$h_1 = \Phi(x_b w_b + x_1 w_{1,1} + x_2 w_{2,1} + x_3 w_{3,1} + \cdots x_N w_{N,1} = z_1) \tag{3.21}$$

Where Φ is the activation function. The output of this is fed to the second layer's respective node, $h_1$in the hidden layer. A similar process with different weights, W_2, and biases for the hidden layer, results in the outputs of the neural network, $y_1$thru y_L. There are two key components to an MLP algorithm, an error measure and the update rule. Error is measured as the mean squared difference of the target and the neural network's output. Here, target is calculated similar to the Q-table update rule in Algorithm 2.1, using the reward, discount factor, and Q_{max} values. Error, which is referred to as loss in some of the literature, is calculated using (3.22) [18]

$$L_i(\theta_i) = E_{s,a}[(y_i - Q(s, a,; w_i))^2] \tag{3.22}$$

Where *y* is the target for iteration *i*, *s* is the state, *a* is the action, and *w* is the weights of the neural network.

The update rule will update the weights of the neural network, by minimizing the error in (3.22).

Along with using the above target definition, in this research, a variation of this algorithm was used, with Q-table values as the target values, to assess the effectiveness of the target values being calculated during the training of the MLP learning algorithm.

In the MLP algorithm approach, the rewards are still being calculated using (3.19) and (3.20) explained in section 3.3. The neural network's aim at maximizing discounted overall reward would translate into the user taking longer route to reach the destination and collecting more reward in the process. The point of using a much larger integer constant for the goal reward was to motivate the learning agent to send the user towards the goal, in fewer steps. But this approach does not seem to work as the size of the network grows. So, to assist with this contradictory reward system (minimizing number of steps to reach goal versus collecting more reward on the way to goal), the MLP is replaced with a DQN. A DQN, as discussed in the previous chapter, is a combination of deep learning and RL. The deep in DQN refers to number of layers in the neural network, the greater the number of layer, the deeper the neural network [12]. Another difference between MLP and DQN can be the use of convolution in DQN, where an image convolution refers to an element-wise multiplication of two matrices, followed by a sum of the elements [12]. Other distinctions between MLPs and DQNs would be the use of batch training and separate target and policy networks in DQNs.

So, the following measures are taken to address the contradictory reward system in MLP:

1. The MLP is replaced with a DQN without extra layers and convolution, with batch training. Batch training refers to choosing batches of transitions randomly, from the replay memory (referred to as experience replay is some literature), to train.
2. The DQN will update a fixed target network (which is used in the weight updates) every fixed number of steps
3. Considering that the goal is set for a given training, i.e. goal is located at $[90, 90]$ on the network layout in Figure 3.2, no matter where the user is, only two directions of movement would be allowed: UP and RIGHT
4. Instead of using the state and action definitions in (3.17) and (3.18), respectively , to define inputs, utilize a slightly different input definition of $[x_t, y_t, bs_{t-1}]$ (no actions included in the neural network's input) where the output of the neural network has $5\ BSs \times 2\ directions = 10$ neurons.

In 1, convolution is not used, because it is mostly utilized in deep learning applications for image processing and computer vision. With the changes in 1 and 2 the DQN has been empirically proven to converge [29]. Considering all the introduced measures, DQNs can approximate shortest path to the goal, as well as a list of BSs to connect to, with a certain level of accuracy (as compared to the Q-learning results), depending on the data rate values received from the serving BSs in the network. The code for this program is accessible in [15].

Chapter 4

Numerical Results

This chapter reviews all the approaches to the research problem, introduced in the previous chapter, and presents results of applying these approaches to different scenarios.

To recap, the research problem is defined as finding a solution for the sensitivity of mmWave signals to blockages. Because users in mmWave networks may experience disconnections and service interruptions, due to physical blockages in the network's layout, it is important to find reliable connections. For a given user, this work suggests finding solutions that could predict an optimal user path and BSs to connect to, if the user's destination is known.

In the following sections, each scenario will be explained, along with the expected and obtained results. The last section of this chapter will recap the results of this research problem and all the explored scenarios.

4.1 Scenario A: data rate assignment to a fictitious 5x5 network

In this scenario, the network is assumed to have a size of 5×5, with 25 user locations and 5 BSs. The point of this scenario is to demonstrate the functionality of the different algorithms, in a special case with short and alternating blockages. To create these special cases, a list of data rates is created, with no corresponding network.

In each subsection of scenario A, the data rates used are generate randomly using a uniform distribution, with the following assumptions: two of the BSs with relative smaller data rates, i.e. BS0 and BS1 in the range of [1, 3]. One BS with higher data rates, i.e. BS2 in the range of 5. And two BSs with alternating (from one step to another) low and high data rates, i.e. BS3 and BS4 alternating in the range of 1 high in the range of 6 (highest values in the network). The data rate generation will be explained with more detail in the following section.

This special case could not be achieved in a randomly generated network, since it is designed to simulate the existence of a multitude of blockages in the user's path. These blocks would make BS3 and BS4 NLOS in alternating steps, which otherwise are LOS and provide the highest data rates in the network. This way, it is expected that BS2 would be the ideal BS to connect to, if the goal is to maximize data rates at the same time as limiting unreliable NLOS connections and fewer BS switches.

4.1.1 Scenario A-1: Comparing Q-learning (with $\gamma = 1.0$) to DP, max immediate reward action selection, and max data rate action selection

The dynamic programming solution assumes that the user's path to destination is already known, so the action only indicates the BSs for the user to connect to. In the known user path, given the data rates received from each base station in the network, the program returns a list of best BSs to serve the user. This list is obtained with the goal of maximizing the overall reward of the system. The overall reward of the network is calculated using the data rates received at every step of the user (from the serving BS), minus the switching cost overhead, in case of a switch, for all the steps in the user's path. Simply put, the overall reward is the sum of the rewards of every step, considering all steps from start to destination. And the reward of every step is calculated using 3.19 and 3.20, depending on whether there is a change in the serving BS.

In the Q-learning program, the user's path is no longer known. It is only assumed that the user's destination is known but its initial position may be assumed to be anywhere inside the network. The actions chosen in this program will provide a user path and BS connections list, that would maximize the user's overall discounted reward in the network. Because of the importance of the discount factor, γ, in finding the optimal solution, two γ values of 1.0 (section 4.1.1) and 0.9 (section 4.1.2) have been considered. When $\gamma = 1.0$ (when all available rewards are being considered) it is possible to compare the results of the BS list obtained by the Q-learning to the dynamic programming results, if the path generated by QL is used in the dynamic programming solution.

After using the Q-learning algorithm (with γ of 1.0 and switching cost of 2) to find the optimal user path (Fixed Path1, FP1), this path is used to compare BS lists obtained by Q-learning and dynamic programming (DP-FP1). These approaches are then compared to a method similar to dynamic programming, where the goal is to maximize the immediate reward of each state. This means that the actions chosen at every step (which BS to connect to) are only based on the knowledge of the reward of the same step, given the user's path. Next, they are compared to an approach with maximum data rate at every step.

Therefore, for a Fixed Path (FP1), the sum of the total rewards received is used to compare the results of the following cases:

1. Choosing actions (BSs) that would maximize overall reward, Dynamic Programming following Fixed Path1, DP-FP1
2. Choosing actions (BSs) with Maximum Immediate Reward following Fixed Path1, MIR-FP1
3. Choosing actions (BSs) with Maximum Data Rate at every step following Fixed Path1, MDR-FP1

For a fictitious dense mmWave square network with 25 possible user locations, the user can start anywhere in the network and is expected to reach the destination at the North-East corner of the network in the fewest possible steps. Data rates are generated randomly using a uniform distribution for every BS in this network, and for all of the 25 possible user location, with the following characteristics:

- The data rates received from two (of the five) base stations, BS_0 and BS_1, were randomly generated using a uniform distribution with values between [1.0, 3.0],
- One base station, BS_2, with data rates in the range of [5.1, 5.5],
- And two base stations, BS_3 and BS_4 with alternating data rates of [1.1, 1.5] and [6.1, 6.5]. For example, if for a given step, if BS_3's data rate is in the range [1.1, 1.5] then for the same step, BS_4's data rate is in the range of [6.1, 6.5].

The data rates on Table 4.1, the result of the above scheme, are then used in calculating the rewards. These rewards are then utilized in determining the optimal user path and BS list, which yields the highest overall reward at destination.

The BS lists generated by the fixed path and Q-learning methods, using the data rates and rewards explained above, are presented on Table 4.2 along with their respective total rewards and sum of data rates.

Table 4.1. Data rates (units/step) received from every BS for every possible user location

step number	BS0	BS1	BS2	BS3	BS4
0	1.1970	1.8310	5.1627	6.2192	1.3163
1	2.8510	1.2761	5.2112	1.2410	6.2378
2	1.6282	2.1514	5.2806	6.2132	1.3766
3	1.6034	1.3935	5.1615	1.2430	6.2173
4	2.3682	1.2965	5.1378	6.1091	1.1318
5	2.0503	1.1239	5.1935	1.2228	6.2401
6	2.9310	1.8799	5.2381	6.2884	1.4459
7	2.1371	2.1882	5.2846	1.4528	6.3073
8	1.8659	1.1262	5.2536	6.2370	1.2424
9	2.6793	2.0366	5.1527	1.2915	6.1798
10	1.2595	1.0604	5.1362	6.2050	1.3792
11	2.7429	2.1727	5.2562	1.1546	6.2252
12	1.1064	2.0034	5.2019	6.3212	1.4617
13	1.5033	1.0851	5.1927	1.2103	6.4932
14	2.4345	2.3173	5.1767	6.4944	1.4876
15	1.8813	2.7374	5.1083	1.2610	6.3538
16	1.0553	1.2679	5.2723	6.3503	1.4654
17	2.8534	2.9366	5.2876	1.1091	6.3435
18	1.7933	1.1399	5.2373	6.4011	1.1229
19	2.6061	1.5928	5.2995	1.2139	6.3996
20	2.9660	1.2993	5.2071	6.3860	1.3035
21	1.2693	2.1694	5.1388	1.4290	6.2668
22	2.4679	1.4715	5.2317	6.4248	1.3358
23	2.3396	2.3133	5.1658	1.3275	6.1114
24	1.7863	1.9794	5.2554	6.4296	1.1111

Table 4.2. BS list and total reward for Q-learning, for fixed path dynamic programming, maximum immediate reward, and random action selection results using switching cost of 2

step number	BS list for QL	BS list for DP-FP1	BS list for MIR-FP1	BS list for MDR-FP1
0	2	2	3	3
1	2	2	4	4
2	2	2	3	3
3	2	2	4	4
4	2	2	3	3
5	2	2	4	4
6	2	2	3	3
7	2	2	4	4
8	2	2	3	3
Total Reward:	47.2207	47.2207	40.8725	40.8725
∑ data rates (*units*)	47.2207	47.2207	56.8725	56.8725

Table 4.2 shows the results of the different fixed-path cases (DP-FP1, MIR-FP1, MDR-FP1) compared to Q-learning using a switching cost of 2, while Table 4.3 demonstrates the same comparison results using a switching cost of 3. This change in switching cost is expected to affect the total reward, in cases where there are base station switches in the BS list.

Table 4.3 BS list and total reward for Q-learning, fixed path dynamic programming, maximum immediate reward, and action selection results using switching cost of 3

step number	BS list for QL	BS list for DP-FP1	BS list for MIR-FP1	BS list for MDR-FP1
0	2	2	3	3
1	2	2	4	4
2	2	2	3	3
3	2	2	4	4
4	2	2	3	3
5	2	2	4	4
6	2	2	3	3
7	2	2	4	4
8	2	2	3	3
Total Reward:	47.2207	47.2207	32.8725	32.8725
∑ data rates (*units*)	47.2207	47.2207	56.8725	56.8725

The total reward values in Table 4.2 and Table 4.3, show that the dynamic programming approach results in the highest total overall reward as compared to the other three, as expected. The list of BS numbers also indicates that the dynamic programming method results in a fewer number of switches (no switches in this case) as compared to the other methods, which results in its higher total reward at destination. This confirms that the design of the dynamic programming has been successful at finding the optimal highest overall reward in relation to BS selection, given the proof provided in the earlier chapters for the dynamic programming's design and the comparison in this section. On the other hand, the MIR-FP1 and MDR-FP1 methods result in a higher sum of the data rates for this given path. As mentioned earlier, this is acceptable because the aim of this research is to find solutions that would result in highest overall reward (considering switching cost) and not necessarily highest throughput. The sum of the total data rates obtained in Table 4.2 is the same as the sum of the total data rates obtained in Table 4.3, because the change in the switching cost is not affecting the tagged BSs, and only affects the total rewards of MIR and MDR methods. Also, the total data rates are the same as the total reward in these two tables, for Q and DP-FP1, because of the lack of switches in their respective BS lists. Another important take-away from these tables is that the Q-learning and dynamic programming (given the Q-learning fixed path) solutions result in the exact same BS lists and total rewards. This is because both algorithms focus on finding actions (BSs) that would result in maximum total reward for the user.

The difference in the total rewards of the methods that suggest BS switches of Table 4.2 and Table 4.3, MIR-FP1 and MDR-FP1, is equal to the difference in the switching costs, $3 - 2 = 1$, multiplied by the number of switches in the solution. So, for example, for MIR-FP1 where there are 8 BS switches, the difference between the total rewards of Table 4.2 and Table 4.3 for this approach is shown in (4.5)

$$\begin{aligned}[][MIR - FP1(switching\ cost = 2)] - [MIR - FP1(switching\ cost = 3)] \\ = 40.8725 - 32.8725 = 8\end{aligned} \tag{4.5}$$

Although In this scenario it just so happens that the MIR-FP1 and MDR-FP1 methods result in the same BS lists and total rewards, they may at times result in different BS lists and therefore different total reward values.

4.1.2 Scenario A-2: Comparing Deep Q-Learning and Q-learning (with $\gamma = 0.9$) to maximum immediate reward action selection

This section studies the effect of a 0.9 discount factor on the performance of Q-learning and deep Q-learning. The goal of these algorithms will be to maximize the discounted total reward. Because the rewards are being discounted, this scenario can no longer be compared to the dynamic programming approach that considers all available results in the optimal policy calculation. Action selection with max immediate reward, which is basically $\gamma = 0$, is used to evaluate the total rewards of the two DQL and QL algorithms. Let FP2 and FP3 denote the resulting paths of QL and DQL, respectively. In this scenario, similar to previous sections, using FP2 and FP3 in finding BS lists that provide the highest immediate reward at each step, MIR-FP2 and MIR-FP3 respectively.

So, the comparison cases are:

1. Choosing actions (directions and BSs) that would maximize overall reward (QL)
2. Choosing actions (BSs) that maximize immediate reward, given the optimal path chosen in the Q-learning approach (MIR-FP2)
3. Choosing actions (directions and BSs) that would estimate a maximized overall reward (DQL)
4. Choosing actions (BSs) that maximize immediate reward, given the optimal path chosen in the deep Q-learning approach (MIR-FP3)

The data rates from Table 4.1, and network assumption of section 4.1 are used with a switching cost of 2 to develop the BS lists and total reward on Table 4.4, and with a switching cost of 3 to develop the results on Table 4.5. Table 4.4 and Table 4.5 show that Q-learning results in the highest possible total reward as compared to the other methods. In fact, the Q-learning's total reward is the highest possible achievable total reward, given the destination of the user and the network's layout. Also, the total reward of the Deep Q-learning method (an approximation approach) is higher than the total reward of MIR-FP3.

Table 4.4 BS list and total reward for Q-learning, fixed path3 max immediate reward, deep Q-learning, and fixed path4 immediate reward results using switching cost of 2

step number	BS list for QL	BS list for MIR-FP2	BS list for DQL	BS list for MIR-FP3
0	2	3	3	3
1	2	4	2	4
2	2	3	3	3
3	2	4	2	4
4	2	3	2	3
5	2	4	2	4
6	2	3	2	3
7	2	4	2	4
8	2	3	2	3
Total Reward:	47.2207	32.5843	39.8271	32.5000

Table 4.5. BS list and total reward for Q-learning, fixed path3 max immediate reward, deep Q-learning, and fixed path4 immediate reward results using switching cost of 3

step number	BS list for QL	BS list for MIR-FP2	BS list for DQL	BS list for MIR-FP3
0	2	3	3	3
1	2	4	2	4
2	2	3	2	3
3	2	4	2	4
4	2	3	2	3
5	2	4	2	4
6	2	3	2	3
7	2	4	2	4
8	2	3	2	3
Total Reward:	47.2207	40.8725	46.0733	40.7954

Comparing the total deep Q-learning reward with switching cost of 2 (Table 4.4) and switching cost of 3 (Table 4.5) shows that the higher switching cost, restricts switching and results in higher reward. It is important to note that if the switching cost value is too high (as compared to the available data rates), it could result in a network with no switches and staying connected to a BS even if it is NLOS, which would lower the total reward.

Also, it is clear that the QL and DQL methods which have chosen different user paths, have resulted in different MIR-FP2 and MIR-FP3 total rewards, for either switching cost. So, for all the different possible paths to go from start to destination, the QL would result in the path that would make base stations available to the user, that would result in the highest discounted reward. On the other hand, the DQL approach, will approximate the optimal path and therefore approximate the optimal BS list.

Finally, the total reward of the deep Q-learning algorithm is within acceptable range for an approximator, within 10% of the total reward achievable by implementing Q-learning on the same problem. In fact, from the total rewards on Table 4.5 and equation (4.6),

$$\frac{|DQL_{total\ reward} - QL_{total\ reward}|}{DL_{reward}} = \frac{1.1474}{47.2207} = 2.43\% \tag{4.6}$$

the DQL reward is within 2.43% of the QL reward.

Now that the performance of the DQL algorithm is within acceptable range, a discussion of the difference in applicable network sizes and in programming running times between DQL and QL algorithms is necessary. These topics are covered in the next scenarios.

4.2 Scenario B: 200 × 200 Network with Data Rate Calculation

The results presented in this section, scenario B, show the performance of the three algorithms, discussed in the previous chapter, on a 200×200 dense mmWave network. The number of base stations in this network is determined using a dense mmWave network definition by [26], where the average number of LOS base stations (for an average user) is at least 1. To this end, a random number, 7, is chosen using a uniform random variable within the range of (4.1),

$$2 \times \frac{Network\ Area}{base\ station\ cell\ coverage} = 2 \times \frac{200 \times 200}{\pi \times 50^2} \tag{4.1}$$

Where the base station coverage is assumed to be a circle with radius of 50 units. Also, the number and location of the blockages are chosen at random using a Poisson distribution, keeping a blockage density of 0.001.

Figure 4.1 is the layout of this network, where the solid black circles identify the location of the BSs and the dashed green line circles are the coverage radius of each BS. The short blue lines represent the blockages and the magenta star-shaped lines outline the boarders of the network. mmWave communications use directional beams for transmission, thus if the line connecting user and a base station intersects with the line of a blockage then the link of the user to that base station becomes NLOS. The use of two-dimensional instead of three-dimensional blockages such as buildings may be a good compromise. As ignoring the height of the base stations may increase the blockage, on the other hand ignoring the area of blockages reduces the blockage.

Also, the user is assumed to have a step size of 5 units. Therefore, there are a total of 1600 user locations in this 200×200 network.

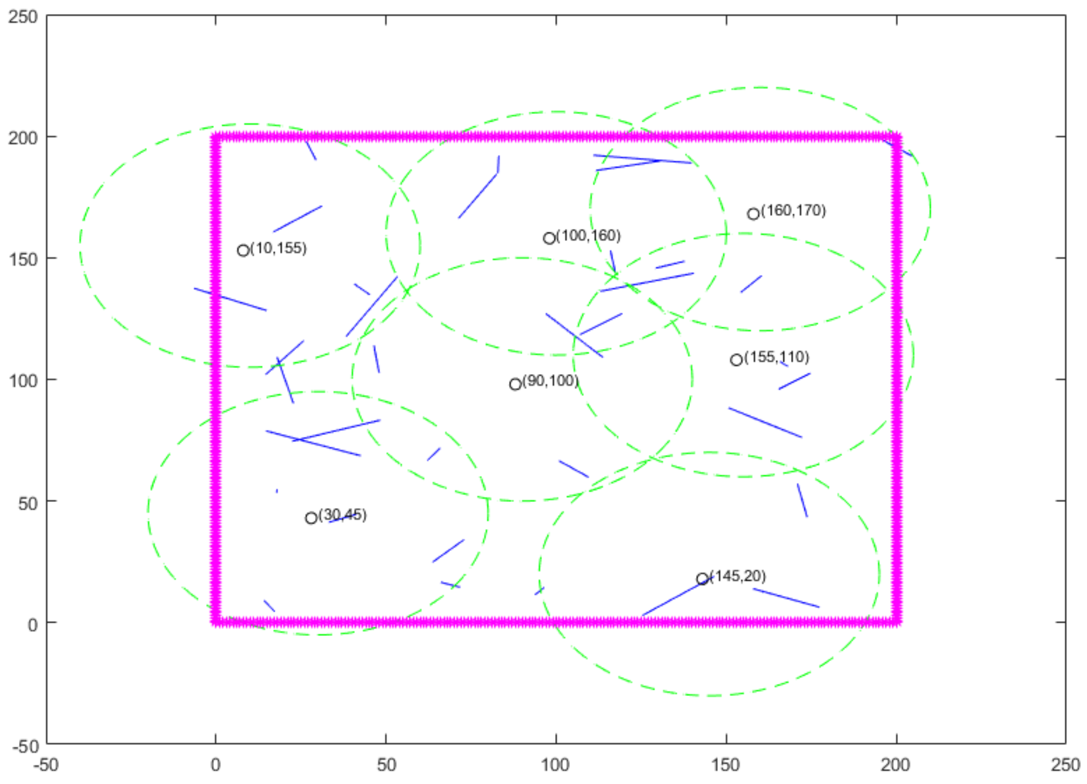

Figure 4.1 Layout of a 200x200 network with 7 base stations and 36 blockages

The network layout (locations of BSs and blockages as well as blockage lengths) is used in the data rate calculations for the 1600 possible user locations, using independent Nakagami fading for LOS (N_L) and NLOS (N_N) links [26]. Each data rate is calculated using (4.2) when the signal-to-noise-and-interference ratio (SINR) value is $SINR < 100$,

$$rate\ (bps) = BW \times \log(1 + SINR) \tag{4.2}$$

And (4.3) when $SINR \geq 100$,

$$rate\ (bps) = BW \times \log(1 + 100) \tag{4.3}$$

Where BW is the bandwidth and is assumed to be 2 GHz, and SINR is calculated using (4.4) from [26].

$$SINR = \frac{|h_0|^2 M_r M_t L(R_0)}{\sigma^2 + \sum_{l>0} |h_l|^2 D_l L(R_l)} \tag{4.4}$$

In (4.4):

h: A normalized gamma variable dependent on whether the BS is LOS or NLOS to the user

M_r: Main lobe gains of the mobile station

M_t: Main lobe gains of the base station

$L(R)$: Path loss gain

σ^2: Thermal noise power

$\sum_{l>0} |h_l|^2 D_l L(R_l)$: Sum of the interferences with directivity gain variable, D_l

For more detail on the above variables, refer to [26].

These data rates, calculated using (4.2) and (4.3), are then used in comparing the performance of the three approaches discussed in the previous chapter: dynamic programming, Q-learning, and deep Q-learning.

4.2.1 Scenario B-1: Comparing Q-learning performance with different number of training steps

In this section, Q-learning algorithm is used to determine the optimal user path and BS list on the network introduced in the previous section, the network in Figure 4.1, given the user's start and destination points. The goal of this scenario is to show the importance of choosing the number of training steps that would return optimal results.

The algorithm runs for a variable number of training steps, using $\gamma = 1.0$ and switching cost of 2 (for when a BS switch occurs). The Q-learning agent is trained first with 1.5×10^6 training steps, then with 6×10^6 training steps, and lastly with 9×10^6 training steps. Figure 4.2 shows the steps the user will take, after each training, when it is tested on the user starting at the origin with a destination at $[195, 195]$. As it can be seen in Figure 4.2, different number of training steps result in different paths for the user, for the same start and destination pair.

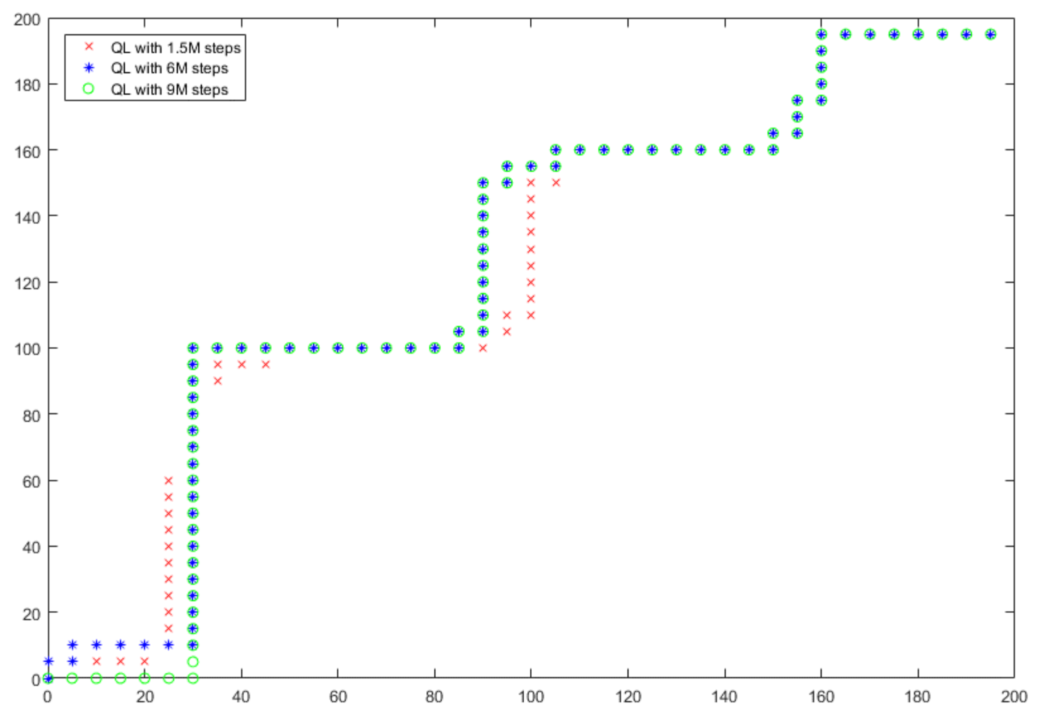

Figure 4.2. User's path from origin to destination at [195, 195], using Q-learning (with γ = 1.0 and switching cost of 2) variable number of training steps

The actions the QL algorithm chooses, determine the user's direction and BS list. The different user paths, shown in Figure 4.2, and different BS lists (list of all the BSs the user will connect to for each step on the path) have resulted in the difference in their respective total rewards. Shown in Table 4.6, the highest total reward (highest reward user path and BS list) corresponds to the training that uses 9×10^6 training steps. The trainings with fewer steps, 1.5×10^6 and 6×10^6 steps, have a smaller overall reward, which is to be expected. This is because the longer the training, the more experience the learning agent gathers from the state-action pairs in the environment.

Table 4.6. Total reward of running QL algorithm for different training step sizes, considering γ of 1.0 and switching cost of 2

	QL with **1.5M** training steps	QL with **6M** training steps	QL with **9M** training steps
Total Reward	283.2923	411.6053	438.1602

Next, the same variable training steps scenario is done with γ = 0.9. This time the training steps are chosen to be 1.5×10^6, 9×10^6, and 12×10^6 to show that increasing the number of training steps is only effective until the highest possible reward is achieved (however, the gain in total reward is not that high for going from 1.5M to 9M steps in Table 4.6). After that, increasing the number of training steps will not positively affect the total reward.

So, in order to draw this comparison, the QL agent is trained on the same network in Figure 4.1. After the training, each neural network is tested on the same user, starting at the origin and heading to a destination at $[195, 195]$. This time, as shown on Figure 4.3, the trainings with 9M and 12M training steps result in the exact same user path. Also, it can be seen on Table 4.7 that these two trainings result in the exact same total reward values. This means that training for the extra 3M steps did not expose the learning agent to any new experiences.

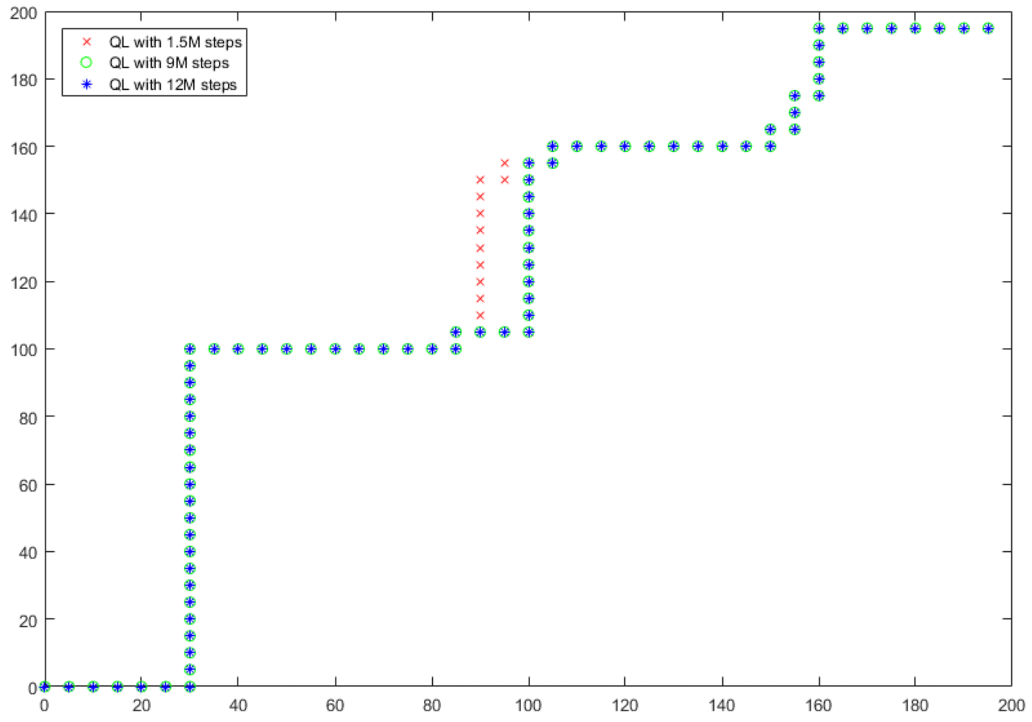

Figure 4.3. User's path from origin to destination at [195, 195], using Q-learning (with γ of 0.9 and switching cost of 2) variable number of training

Table 4.7. Total reward of running QL algorithm for different training step sizes, considering γ of 0.9 and switching cost of 2

	QL with **1.5M** training steps	QL with **9M** training steps	QL with **12M** training steps
Total Reward	412.0859	433.2178	433.2178

This scenario confirms that the number of training steps are important in the optimality of the Q-learning results. If they are too small, because training happens on randomly selected state-action pairs, it would mean that there are state-action pairs that the user has not yet experienced. If they are chosen to be too high, there may not be any extra experiences for the learning agent to benefit from.

4.2.2 Scenario B-2: Comparing Q-learning to DP, maximum immediate reward action selection, maximum data rate action selection, and random action selection

In this section, to compare the QL and DP results, the user paths generated by the QL algorithm with 9M training steps and γ = 1.0 (Fixed Path4, FP4), in section 4.2.1, will be used to compare BS lists with DP (DP-FP4). This is because, as mentioned before, the dynamic programming approach will return an optimal BS list for a fixed-path user. The Q-learning and dynamic programming total rewards are then compared to three other methods to ensure that they always result in the highest overall rewards. These three methods that use the same FP4 (the optimal path generated by the QL algorithm, FP4, is the path being followed in all the fixed path methods) are:

1. Maximum Immediate Reward action selection using Path4 (MIR-FP4), chooses actions (BSs) that would maximize the user's immediate reward
2. Maximum Data Rate action selection using Path4 (MDR-FP4), chooses actions (BSs) that provide the highest data rate at every given user step
3. Random action selection using Path4 (Random-FP4), chooses a random action (BS) for every user step

Where reward is calculated using (3.19) and (3.20), depending on whether there is a BS switch, and total reward is the sum of the rewards of all the user's steps.

Table 4.8. Total rewards of QL compared to fixed-Path1 methods

	QL with 9M training steps & $\gamma = 1.0$	DP-FP4	MIR-FP4	MDR-FP4	Random-FP4
Total Reward:	438.1602	438.1602	434.5552	430.7748	-28.9333

Table 4.8 shows that the total reward of QL and DP methods are the same, for the same path. Also, their rewards are higher than the rewards of the other fixed-path methods. In fact, with the goal of maximizing highest total reward, Q-learning and dynamic programming are the best solutions. They do however, each have their own drawbacks. Dynamic programming can only find highest BS list, and the user's path would have to be pre-determined. Q-learning will take many steps to train, until every single state-action pair has been visited. This would render Q-learning not effective in cases where every state-action pair is not known, or that the environment is so large that it is not feasible to visit every state-action.

4.2.3 Scenario B-3: Comparing Deep Q-Learning and Q-learning (with γ = 0.9) to maximum immediate reward action selection

This section will demonstrate DQL's performance on the 200×200 network in Figure 4.1, and compares the results with the QL results of scenario B-1. The reasons for the DQL not accurately predicting the optimal BS list and user path for this network are discussed, along with measures that were tried to overcome these reasons.

Because DQL is an approximation method, it is not necessary to visit every state-action pair before an optimal solution is predicted. This is why DQL is preferred in networks with larger state sizes. But, in order to draw parallels between the DQL and QL results, the same number of training steps are used in the training of DQL, 1.5×10^6 and 9×10^6 training steps. Using DQL, considering a discount factor of 0.9 and switching cost of 2, the results on Table 4.9 were obtained.

The total reward values from DQL are then compared to maximum immediate reward results, given the same path that the DQL suggests for a user that starts at origin with a destination of $[195, 195]$. The results of MIR with fixed paths, Fixed Path5 (FP5) for 1.5M training steps, and Fixed Path6 (FP6) for 9M training steps, are also included on Table 4.9.

Table 4.9 Total reward of DQL (with γ of 0.9) for different number of training steps compared to MIR-FP5 and MIR-FP6

	DQL with **1.5M** training steps	MIR-FP5	DQL with **9M** training steps	MIR-FP6
Total Reward	2.9925	48.6388	2.2473	21.0717

Table 4.9 shows that not only the total rewards acquired by DQL is not highest, in the long-run, but with a larger number of training steps, the path suggested by DQL has actually made the total reward for MIR to drop from 48.6388 (with FP5) to 21.0717 (with FP6). This means that the DQN has not been able to learn from the experiences in the environment to obtain an optimal user path and BS list, given a destination. The following are two possible reasons for this.

1. One reason that a DQN is not converging could be because of the dataset. As one of the requirements of a Deep feedforward network to correctly approximate a function, is for the dataset to be continuous [30]. The data rates in this scenario are not continuous.
2. Another reason could be the activation function, sigmoid function's, limitations at correctly assessing values close to the edges of its range, 0 and 1. It seems that the activation function is not sensitive to the range-edge values. The RELU activation function was also unsuccessful, using the same data rates.

To verify the mentioned possible reasons, a constant value was added to the calculated data rates, to ensure that they were not falling on range-edge 0. Also, to better show detailed results, the user's starting location was limited to the destination (so the user starts at destination and only takes one step, distance to destination is 0). The discounted reward values for every possible action for this step, using DQL, are then compared to discounted Q-values from QL, shown in Table 4.10. In Table 4.10 and Table 4.11 the number of training steps for DQL and QL is set to 3×10^5 steps and switching cost is set to 0.8. In both tables, Q-values that are calculated as 1.0 are corresponding to actions that require no switch in the BS, and the Q-values that are calculated as 0.2 are corresponding to actions that require a switch in the BS.

Table 4.10 shows that with the new data rate values, the average DQL discounted reward values are within a 0.35% range of the QL discounted Q-value results for the actions with switching, and within a range of 0.01% of the QL discounted Q-value results for the actions without switching. This means that with the user starting at the destination, DQL is finding the optimal solution.

Table 4.10 DQL discounted reward and QL discounted Q-values for all possible actions when user's distance to destination is 0

Action (BS)	Action (direction)	DQL discounted reward values	QL discounted Q-values
0	UP	1.0001	1.0
1	UP	0.2057	0.2
3	UP	0.2011	0.2
4	UP	0.1853	0.2
5	UP	0.2003	0.2
6	UP	0.198	0.2
0	RIGHT	0.2043	0.2
1	RIGHT	1.0000	1.0
2	RIGHT	0.2058	0.2
3	RIGHT	0.2008	0.2
4	RIGHT	0.2033	0.2
5	RIGHT	0.1942	0.2
6	RIGHT	0.1891	0.2

Next, the user is allowed to be at a short distance from the destination, taking two steps to reach the destination, distance to destination is 2, and the results of DQL and QL discounted Q-values (same as discounted reward values) for these steps are compared for every action in every state, shown in Table 4.11. This time, the average DQL discounted reward values are within a 510.1% range of the discounted Q-values of QL for actions with switching, and within a range of 33.26% for actions without switching. This shows that by adjusting the lower data rates (values close to the lower end of the sigmoid function’s range), adding a constant to these data rates, DQL still cannot find optimal solutions because the data rates are not continuous.

Table 4.11 DQL discounted reward and QL discounted Q-values for all possible actions when user’s distance to destination is 2

Action (BS)	Action (direction)	DQL discounted reward values	QL discounted Q-values
0	UP	1.4082	1.0
1	UP	1.2343	0.2
3	UP	1.3309	0.2
4	UP	1.2473	0.2
5	UP	1.3507	0.2
6	UP	1.0899	0.2
0	RIGHT	1.2520	0.2
1	RIGHT	1.2569	1.0
2	RIGHT	1.0680	0.2
3	RIGHT	1.3824	0.2
4	RIGHT	1.1874	0.2
5	RIGHT	1.1523	0.2
6	RIGHT	1.2349	0.2

4.3 Scenario C

This scenario’s aim is to compare the performance of Q-learning and deep Q-learning algorithms on larger networks, with data rates that are generated randomly using a uniform distribution, to eliminate the problem in the previous section.

Scenario C-1 will compare the results of Q-learning and deep Q-learning algorithms for a network size that the Q-learning algorithm can reasonably return results for. Next, scenario C-2, will demonstrate the performance of deep Q-learning for a network size that is larger than what QL algorithm can effectively find optimal solutions for. In this case, DQL can find better than random solutions.

4.3.1 Scenario C-1: Comparing Deep Q-Learning to Q-learning on a 150 × 150 network (with $\gamma = 0.9$)

So far, it is known that the Q-learning method returns accurate results, if enough training steps have passed. That is because if any state-action pairs are not covered during training, those state-action pairs that have been visited cannot result in an optimal policy. On the other hand, the DQL algorithm returns meaningful results in fewer steps. Deep Q-learning, as an approximation method, can approximate the value of state-action pair that it has not yet visited. The accuracy of the approximation will depend on learning parameters such as the number of hidden layers in the DQN and the value of γ, the discount factor, and the activation function(s). The ability for DQL to approximate states that have not been visited, helps with this approach being used in cases where Q-learning is too costly or even impossible to perform.

But, one of the drawbacks of utilizing DQNs is the time spent running the program, if the computations are not performed on a GPU. Another is that DQNs are ideal for continuous datasets, and in application where the values in the datasets are not continuous this method could fail to approximate accurately. This section focusses on drawing comparisons between the two algorithms, and their run-times.

In a 150×150 dense mmWave wireless network, bounded between $x = 0 \; and \; 150$ and $y = 0 \; and \; 150$, the user starts anywhere in the network and is expected to reach the destination at $[150, 150]$ in the fewest possible steps. There are a total of $30 \times 30 = 900$ user locations in this network, since the user takes steps of size 5 units.

The data rates are generated randomly for every one of the 5 BSs and all 900 possible user locations in this network, without the layout of BSs and blockages. These data rates are then used to determine the optimal user path and BS list, which yields the highest overall reward using Q-learning and deep Q-learning algorithms.

After training both algorithms for 450K steps, the results for testing a user that starts at origin with a destination of $[150, 150]$ and switching costs of 2 and 3 are compared. The program running time and total rewards for these trainings are presented on Table 4.12, for switching cost of 2. Table 4.12 shows the run-time of the Q-learning program to be 3.9776 seconds. This is significantly smaller than the 348.6955 seconds it takes for the deep Q-learning program to run, for the same number of steps. Although both run-times are not remarkably long, in other words it does not take days to run these programs, they are a good indicator of why Q-learning is preferred in smaller and medium-sized networks.

Table 4.12. Program run-time and total reward of QL & DQL in 150×150 *network with switching cost of 2*

	Q-learning	Deep Q-learning
Total Reward:	328.9408	312.3125
Total Training Running Time(s):	3.9776	348.6955
Total number of training steps:	450000	450000

Also, the total reward of the DQL is 312.3125, which is 5.06% lower than the QL total reward. This value is still within the acceptable 10% range of the QL total reward.

Next, Table 4.13, demonstrates the run-times and total rewards of QL and DQL on the same network, for a switching cost of 3.

Table 4.13 Program run-time and total reward of QL and DQL in a 150×150 *network with switching cost of 3*

	Q-learning	Deep Q-learning
Total Reward:	338.9446	309.6918
Total Training Running Time:	4.0077	335.1640
Total number of training steps:	450000	450000

In this scenario, both programs run within the same time frame that they ran for the switching cost of 2 scenario. And although these times are still also not remarkable long (not hours or days long), DQL takes more than $80 \times$ longer than QL to train on the same number of steps. And results in total reward that is 8.63% of the QL total reward.

4.3.2 Scenario C-2: Deep Q-learning for a very large network (with $\gamma = 0.9$)

In this section, a network with large state space is used to demonstrate the performance of DQL when QL is ineffective.

As mentioned in earlier sections, the number of BSs in a network are determined by the size of the network as well as the BSs' coverage area. So, assuming a circular cell with radius of $50\ m$ and a $1500 \times 1500\ m^2$ network, and using (4.7)

$$\frac{Network\ Area}{BS\ coverage\ area = \pi r^2} = \frac{1500 \times 1500}{\pi(50)^2} = 286 \tag{4.7}$$

There will need to be at least 286 BSs to cover the entire area. For simplicity, it is assumed that there are 300 BSs in the network, without the real layout of BSs and blockages on the network. And, with a user step size of 5, this is a network with 300×300 possible user locations. So, this network will have $300 \times 300 \times 300 = 27 \times 10^6$ states and $300 \times 2 = 600$ actions. The number of actions corresponds to the number of BSs in the network, for every direction the user can move in. As stated earlier, the user can move in two directions: UP or RIGHT.

For this network, using Q-learning would require a matrix with 27×10^6 rows and 600 columns. Populating a matrix of this size, which would result in a memory error, is computationally expensive and is referred to as the curse of dimensionality [29]. Therefore, DQL that does not require the population of such a large matrix, is used as an alternative approach.

DQL is an approximation method, therefore every state-action does not need to be visited for the program to return optimal user path and BS lists. Using switching cost of 2, discount factor of 0.9, 20 hidden layers, and only 100×10^3 training steps, the algorithm is expected to predict a user path and BS list that would return high discounted rewards. During testing, the user starts at origin and reaches the destination at $[1495, 1495]$ within the minimum number of steps, 599 steps, which results in a total reward of 4045.15.

Because QL and DP cannot be used to evaluate the results of the DQL, the random action selection method was used to show that DQL will perform better than if each step's action was chosen at random. For the same 599 steps that it takes the user to go from origin to destination in the DQL solution, a random action is chosen at every step, and the rewards obtained from all the steps are added for a total reward of -1194.0. This value is acceptable because in random action selection method, the user will connect to any BS in its path, without regard to the data rate received from that BS or the cost of switching to another serving BS. It is clear that the total reward of the DQL method, 4045.15 is higher than the total reward of choosing actions at random, -1194.0.

The above results show that because the reward values are not continuous, DQL cannot find optimal solutions to the problem. But they may find better-than-random results, when it is too expensive to apply QL to the problem, in a fraction of the number of training steps it would take to run QL.

4.4 Results

The scenarios in the previous sections of this chapter were all designed to demonstrate the characteristics of the solutions to the research question.

The outcomes of these scenarios reveal that given a fixed path for the user, the dynamic programming approach results in the highest total reward. It is also exhibited that given a fixed destination; the Q-learning algorithm will result in the highest total reward, identical to the results obtained by dynamic programming (using the same path provided by the Q-learning algorithm). After training, during testing of the algorithms, if the Q-learning and deep Q-learning algorithms are given the same starting and destination points for the user inside the network an optimal path and BS list will be returned.

Next it is shown that the run-time for the DQL training program is longer than QL's training run-time, but fewer number of training steps are required to obtain acceptable results. The results of the DQL are within an acceptable 10% range of the exact solution, QL. Also, for large networks where QL cannot be used, DQL can provide BS lists and user paths that are better than randomly choosing BSs for the same user path.

For smaller networks, Q-learning has been successful at predicting temporary blockages, by learning the layout of the network through a reward system. Suggesting user paths and BSs to connect to, in that path, determined by maximizing overall discounted reward, ensures a reliable and low-overhead connection for the user. For larger networks, even though deep Q-learning may not approximate optimal solutions, it has been more successful than a random selection of actions.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The growing need for spectrum in the world of IOT and the demand for reliable and high throughput mobile connections has motivated researchers and industry pioneers to explore the previously unused mmWave spectrum for 5G wireless systems. 5G wireless networks will have enhanced mobile broadband with high throughput rates, but reliability and latency could become drawbacks. The sensitivity of mmWave signals to physical blockages, can cause disruptions in the connection or cut the connection off entirely. Because of the highly directional nature of these signals, a direct LOS between the user and the serving BS is necessary. If this link becomes NLOS, given that the user is still connected to the network, user hand-off to another base station will incur overheard and latency issues.

To address the blockage issue in 5G mmWave wireless networks, the research problem studied solutions to predict all blockages in a network and assess if a hand-off is cost-efficient. This means that if the blockage is considered temporary, given the cost of switching the user from the current BS to another BS, then it may be beneficial to the overall reward of the user if the current BS continues to serve the user. The decision to consider a blockage temporary or not, will also be based on the data rate received from the serving BS, as well as all other available BSs in the user's current location.

To address the research problem, the following approaches were studied:

1. Dynamic Programming, which only provides the optimal BS list,
2. Q-learning, which provides the optimal path and BS list, but is only used for smaller networks,
3. Deep Q-learning, which approximates the optimal path and BS list of the user and can be applied to larger state-action spaces.

The first approach, dynamic programming, can predict the optimal BSs to connect to, given that the user's path is already known. The BSs considered optimal are chosen with the definition of a temporary blockage in mind. In the case of an NLOS link, the data rates received from the BS currently serving the user are compared to the data rates received from all other available BSs in the network, and considering the cost of a hand-off, an optimal BS is chosen for that step. A list of BSs is considered optimal, once the user has reached its destination and all the rewards have been calculated with the goal of maximizing the overall reward.

Approaches 2 and 3, dependent on the size of the network and the trend of the data rate values, can be utilized in finding optimal path and BS connections for a given destination in the network. In these approaches the goal is to maximize the discounted total reward, which is a combination of the data rates and BS hand-off overhear, referred to as switching cost.

For more sophisticated state spaces, DQL, approach 3, has been shown to perform better than random action selection. The total reward obtained by DQL is higher than the total reward value of choosing each step's action at random, given a fixed destination inside the network.

The reward system used in these approaches is defined to teach the learning agent the long-term impact of the blockages in the network. The learning agent will gain experience by visiting different state-action pairs inside the network and learn to find optimal solutions given that experience.

Dynamic programming, can provide the optimal BS lists and predict a blockage's effect length on the user, given the user's path. Q-learning can predict a blockage's effect and choose a path for the user accordingly. It will also provide an optimal BS list for this path, given the user's fixed destination. Deep Q-learning can approximate the optimal user path and BS list, but the accuracy of the results obtained by this method depends highly on the continuity of the values in the dataset provided to the learning agent. Using a dataset with randomly generated values from a uniform distribution, this approach can provide results better than choosing actions at random.

5.2 Future Work

From the work on this research, the user's path and ideal BS connections can be predicted, given the user's destination inside the network. This information can be used in future research to provide a framework for BS sleep patterns. Considering that 60 - 80% of network power consumption is for base station usage [22], this could prove to be an effective way to save network energy usage.

In the current solutions provided, it is assumed that the mobile user is walking, and a speed is not considered in the user's movements. In future work, the user's speed could be considered as a parameter, allowing the user to be moving at variable velocity.

Another similar variation of the problem to be considered could be with multiple users in the same network, in a 3-dimensional environment so that the BSs, blockages, and even the users could be at variable heights.

References

[1] S. Niknam, B. Natarajan and R. Barazideh, "Interference Analysis for Finite-Area 5G mmWave Networks Considering Blockage Effect," *IEEE Access,* vol. 6, pp. 23470-23479, 2018.

[2] G. R. M. a. T. S. Rappaport, "Millimeter-Wave Base Station Diversity for 5G Coordinated Multipoint (CoMP) Applications," *IEEE Transactions on Wireless Communications,* vol. 18, no. 7, pp. 3395-3410, July 2019.

[3] A. &. B. I. &. A. S. Alkhateeb, "Machine Learning for Reliable mmWave System: Blockage Prediction and Proactive Handoff," in *Global Conference on Signal and Information Processing (GlobalSIP)*, Anaheim, 2018.

[4] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong and J. C. Zhang, "What will 5G be?," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS,* vol. 32, no. 6, pp. 1065 - 1082, 2014.

[5] N. Bhushan, J. Li, D. Malladi, R. Gilmore, D. Brenner, A. Damnjanovic, R. T. Sukhavasi, C. Patel and S. Geirhofer, "Network densification: the dominant theme for wireless evolution into 5G," *IEEE Communications Magazine,* vol. 52, no. 2, pp. 82-89, February 2014.

[6] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta and P. Popovski, "Five disruptive technology directions for 5G," *IEEE Communications Magazine,* vol. 52, no. 2, pp. 74-80, February 2014.

[7] W. Roh, J.-Y. Seol, J. Park, B. Lee, J. Lee, Y. Kim, J. Cho, K. Cheun and F. Aryanfar, "Millimeter-wave beamforming as an enabling technology for 5G cellular communications: theoretical feasibility and prototype results," *IEEE Communications Magazine,* vol. 52, no. 2, pp. 106-113, February 2014.

[8] W. Na, B. Bae, S. Cho and N. Kim, "DL-TCP: Deep Learning-Based Transmission Control Protocol for Disaster 5G mmWave Networks," *IEEE Access,* vol. 7, pp. 145134-145144, 2019.

[9] C. M. G. a. J. Snell, Introduction to Probability, second revised ed., Providence, RI: American Mathematical Society / The CHANCE Project, 2006.

[10] M. L. Puterman, Markov Decision Processes Discrete Stochastic Dynamic Programming, Hoboken, New Jersey: John Wiley and Sons, 2005.

[11] R. S. Sutton and A. G. Barto, Reinforcement learning An Introduction, 2 ed., Cambridge, MA: The MIT Press, 2018.

[12] A. Rosebrock, Deep Learning for Computer Vision with Python, 1st ed., vol. 1, Columbia, SC: PyImageSearch, 2017.

[13] K. Doya, "Reinforcement learning: Computational theory and biological mechanisms," *HFSP,* vol. 1, no. 1, pp. 30-40, 2007.

[14] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Communications Surveys and Tutorials,* vol. 21, no. 4, pp. 3133-3174, 2019.

[15] A. Golkaramnay, "RLformmWave," GitHub, March 2020. [Online]. Available: https://github.com/Artmiz/RLformmWave/releases/tag/v1.0. [Accessed March 2020].

[16] R. S. Sutton and A. G. Barto, "Temporal-Difference Learning," in *Reinforcement Learning An Introduction*, Cambridge, The MIT Press, 2018, pp. 119-140.

[17] C. Zhang, P. Patras and H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," *IEEE Communications Surveys and Tutorials,* vol. 21, no. 3, pp. 2224-2287, 2019.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," DeepMind Technologies, 2013.

[19] H. Mao, M. Alizadeh, I. Menache and S. Kandula, "Resource Management with Deep Reinforcement Learning," in *HotNets '16: Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, Atlanta, 2016.

[20] T. Rajapakshe, R. Rana, S. Latif, S. Khalifa and B. W. Schuller, "Pre-training in Deep Reinforcement Learning for Automatic Speech Recognition," ArXiv, 2019.

[21] J. Liu, B. Krishnamachari, S. Zhou and Z. Niu, "DeepNap: Data-Driven Base Station Sleeping Operations through Deep Reinforcement Learning," *IEEE Internet of Things Journal,* vol. 5, no. 6, pp. 4273-4282, 2018.

[22] M. A. Marsan, L. Chiaraviglio, D. Ciullo and M. Meo, "Optimal Energy Savings in Cellular Access Networks," in *IEEE International Conference on Communications Workshops*, Dresden, 2009.

[23] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu and D. Yang, "Experience-driven Networking: A Deep Reinforcement Learning based Approach," in *IEEE Conference on Computer Communications*, Honolulu, 2018.

[24] O. Semiari, W. Saad and M. Bennis, "Joint Millimeter Wave and Mircowave Resources Allocation in Cellular Networks with Dual-Mode Base Stations," *IEEE Transactions on Wireless Communications,* vol. 16, no. 7, pp. 4802-4816, 2017.

[25] R. Ding, Y. Xu, F. Gao, X. Shen and W. Wu, "Deep Reinforcement Learning for Router Selection in Network with Heavy Traffic," *IEEE Access,* vol. 7, pp. 37109-37120, 2019.

[26] T. Bai and R. W. Heath, "Coverage and Rate Analysis for Millimeter-Wave Cellular Networks," *IEEE Transactions on Wireless Communications,* vol. 14, no. 2, pp. 1100-1114, 2015.

[27] M. Patacchiola, "Dissecting Reinforcement Learning-Part.8," Github, 28 December 2018. [Online]. Available: https://mpatacchiola.github.io/blog/2018/12/28/dissecting-reinforcement-learning-8.html. [Accessed 03 February 2020].

[28] M. Minsky and S. A. Papert, Perceptrons: An Introduction to Computational Geometry, The MIT Press, 1988.

[29] R. Atallah, C. Assi and M. Khabbaz, "Deep Reinforcement Learning-based Scheduling for Roadside Communication Networks," in *15th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, Paris, 2017.

[30] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Cambridge: The MIT Press, 2016.

[31] J. Kim, J. Park, S. Kim, S.-L. Kim, K. W. Sung and K. S. Kim, "Millimeter-Wave Interference Avoidance via Building-Aware Associations," *IEEE Access,* vol. 16, p. IEEE Access, 2018.

[32] W. Fischer and K. Meier-Hellstern, "The Markov-modulated Poisson process (MMPP) cookbook," *Performance Evaluation,* vol. 18, no. 2, pp. 149-171, 1993.