Improving IoT Connection Resiliency in Wireless Networks

A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

by

Hieu Thanh Nguyen
Bachelor of Science
George Mason University, 2021

Director: Dr. Bijan Jabbari, Professor
Department of Electrical and Computer Engineering

Fall Semester 2022
George Mason University
Fairfax, VA

DEDICATION

I dedicate this work to my parents, my sister, and my friends, who always support and comfort me when I was in the troubles. Thanks to their unconditional love, I can make this work successful.

ACKNOWLEDGEMENTS

I would love to thank my advisor, Dr. Bijan Jabbari. Without his guidance and support throughout this process, this thesis would not have been possible. Also, I wish to thank the other members of the committee, Dr. Maryam Parsa and Dr. Sai Manoj Pudukotai Dinakarrao for their valuable inputs and suggestions for my thesis: I am thankful to Dr. Liang Zhang for the guidance he has provided me in the machine learning part.

I am grateful to my parents, Luong Nguyen and Hoi Nguyen, my sister, Lien Nguyen, and my dear friend, Ha-Anh Nguyen. They have given their unconditional love and support to me when I was experiencing difficult times throughout this master's thesis. Without their help, I might not complete this work.

Lastly, I would love to thank the Electrical and Computing Engineering department at George Mason University (GMU) and the members of the Communications and Networking Lab at GMU for their help in my work.

TABLE OF CONTENS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

IMPROVING IOT CONNECTION RESILIENCY IN WIRELESS NETWORKS

Hieu Thanh Nguyen, M.S.

George Mason University, 2022

Thesis Director: Dr. Bijan Jabbari

Internet of Things (IoT) wireless network is expected to connect billions of IoT devices in the next period of modern technologies. This is because IoT applications have more and more applicability in various fields, including services, health, agriculture, and so on. However, along with the significant benefits, IoT requires low-latency and high resilience of wireless communications in order to maintain a high quality of service. IoT networks should constantly maintain a high level of resilience in wireless communication in order to sustain the increasing number of new IoT devices connected to the networks. Since IoT networks consist of thousands of devices sharing the frequency spectrum in a given local area, IoT networks also address the problem of wireless interference that results in link degradation and low network connectivity. Therefore, the performance of In this thesis, we propose two technical solutions to improve the resilience of communications in IoT networks by suppressing wireless interference.

We develop our system models that represent the interference with IoT network access and elements of graph theory for improving the resilience of connections. Our system models include node distribution following Point Poisson Process, wireless network as a graph, modeling interference in IoT network access, node criticality, and elasticity theory. Then, we utilize these models in our proposed solutions for improving the resilience of wireless communications.

In order to avoid channel interference, we implement an algorithm based on the concept of graph theory to efficiently allocate channels used by IoT devices in the network. We observe that the number of colors labeled for each node can be minimized by eliminating several less important nodes, but it is a trade-off between color reduction and network connectivity. Also, we propose an additional solution using deep deterministic policy gradient (DDPG) based on graph coloring to determine the minimum number of colors used.

Our simulation results indicate that the gain of eliminating the least important nodes is color reduction, but depending on each particular wireless network, the solution can achieve a high probability. Another proposed solution is to determine the chromatic number by using deep reinforcement learning-based channel allocation. Although several nodes in the network have the same colors, which leads to invalid/disconnected links, the number of colors using the DDPG algorithm is always smaller than the greedy coloring algorithm.

CHAPTER 1: INTRODUCTION

1.1 Overview of IoT Networks and Motivation

The growth of the Internet of Things applications has brought human beings to the next period of modern technologies, coming along with massive benefits in our daily activities. Internet of Things (IoT) is a system of physical objects combined with various sensors, software, processing data, and other technologies that have the ability to exchange data over a network without requiring human-to-human or human-to-computer interaction [1]. IoT has an ecosystem consisting of multiple web-enabled smart devices that collect, transmit, and analyze the data over a network because most of the IoT devices are assigned an Internet Protocol (IP) address. Sometimes, instead of the IoT devices using the Internet to transfer data, these devices also have the ability to communicate locally. Multiple smart devices can collaborate for the efficiency of productivity. The IoT applications are enabled by the latest technologies in Radio Frequency Identification (RFID), sensors, and protocols. With the diversity of modern technologies, the IoT can take advantage of the evolving new applications that are enabled by intelligent decision-making [2].

Within the century of digital transformation and the Age of Industry 4.0, an IoT network is expected to connect billions of IoT devices and provide users with the best Quality of Services (QoS) in data transmission, computation, and storage. Compared to the

previous 4th Generation (4G) cellular network, in the 5G cellular network, 500 times more IoT-connected devices can operate effectively and reliably [3]. Ergun et al. have stated a prediction that by 2025 the number of IoT-connected devices will reach up to 40 billion, and the IoT applications have received much attention from investors [4]. As a result of the massive numbers of IoT devices connected to the network, the requirement of resilience, low latency, and efficiency in wireless communication has been increased to meet the demand of the users, as well as the high demand of network traffic. In addition, a wide range of communication protocols is available to enable various IoT applications. Therefore, IoT network management is also crucial to ensure that this network can meet all the requirements. Network management in IoT networks is characterized by scalability, fault tolerance, QoS, energy efficiency, security, and self-configuration.

In order to utilize wireless networks to meet the requirement and the demand of the users, four types of wireless networks are commonly used. These networks are Wide Area Networks (WANs), Local Area Networks (LANs), Metropolitan Area Networks (MANs), and Personal Area Networks (PANs). They include a wide range of recent wireless technologies from decentralized networks to centralized networks.

A cellular network (i.e., 2G, 3G, 4G, 5G, and NextG technologies), which is one of the most common networks of wireless WANs, has received much attention from researchers to successfully adapt to IoT networks. Compared to LANs, MANs, and PANs, cellular networks provide users with a service to access and connect to others over further distances in a larger network. In end-to-end communications of these networks, the user

devices are required to maintain the connectivity to a base station, which uses radio frequency antennae to transmit the signals to other wireless devices. Base stations (BSs) have a responsibility to serve as a central point for receiving and transmitting signals between wireless devices to communicate. Since all wireless devices need to transmit and receive the signals over a base station, the QoS and performance of a wireless network can depend on a base station's capabilities, such as transmission power, spectrum, and channel availability. However, although cellular networks have a high capacity, as well as availability, IoT applications may encounter various challenges if adapted to the cellular network. First of all, the network infrastructures to deploy cellular technologies are expensive. Moreover, if the IoT network consists of thousands of IoT sensors operating simultaneously, the power consumption to transmit the signals is expected to be extremely high.

IoT networks take advantage of wireless technologies to maintain connectivity and resilient communication not only between devices but also between devices and the edges or the cloud. Unlike the other networks, IoT networks are available for IoT devices or IoT sensors to automatically transmit the data in real-time to IoT applications. In a cellular network, wireless devices are required to connect to a base station for communications. However, IoT networks can have both device-to-device and device-to-base station communications. Since such networks are heterogeneous and dynamic, IoT networks take advantage of various wireless networks (for example, Wi-Fi and Bluetooth) in order to optimize the performance of each link of the network. Maintaining a high level of IoT communication networks is dependent on various factors, some of which are objective

(e.g., network traffic and network latency), and some of which are subjective (e.g., transmitted power, channel interference, and the number of devices on the network). However, we will discuss three major concerns of IoT networks that all IoT applications have to encounter: resilience, latency, and efficiency. An example of an IoT network architecture is demonstrated in Figure 1.1. The IoT network architecture does not deploy only one type of wireless network. Multiple wireless networks can collaborate for transmitting data from local areas to global ones.

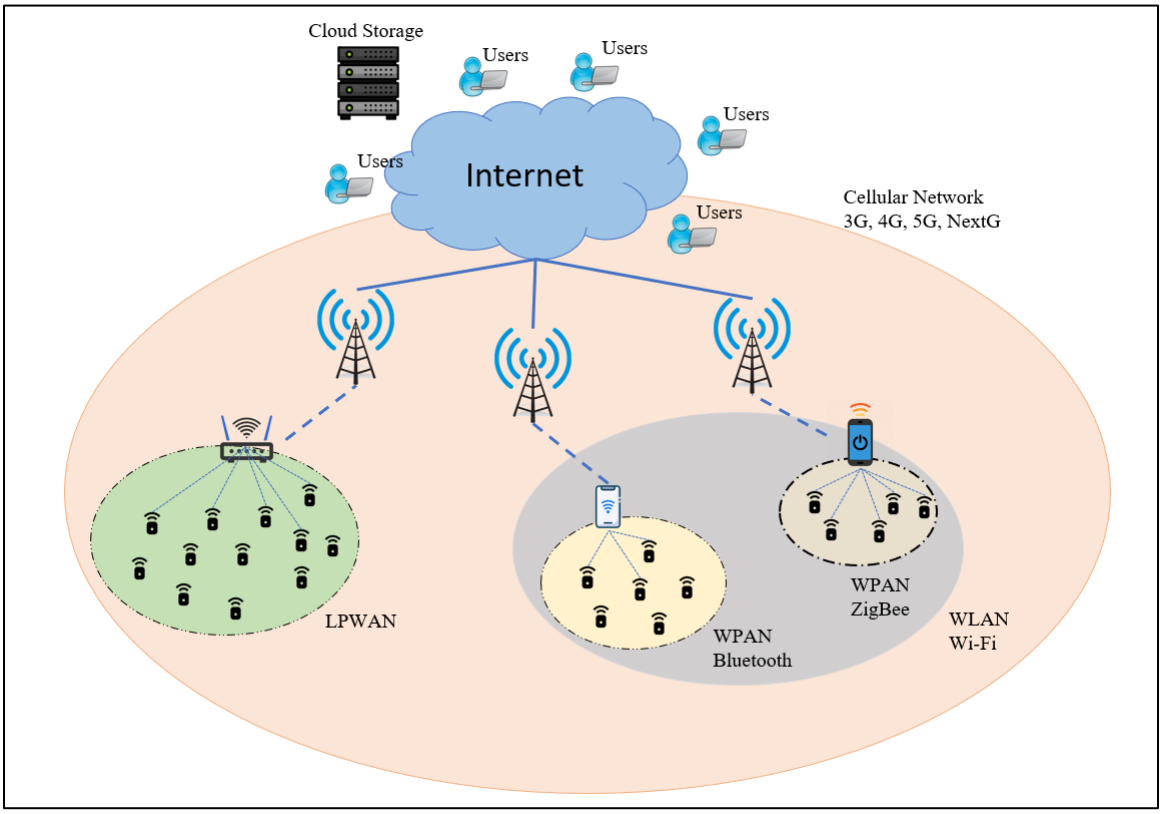

Figure 1.1: IoT Network Architecture

1.2 Problem Statement and Proposed Solutions

The applications of IoT have increasingly become an important part of our lives because of their incredible benefits, such as healthcare monitoring, smart city designs (smart building and smart grid), autonomous vehicles, and so on. Along with the wide coverage of IoT applications, IoT network protocols, which are used to communicate and transmit the data, must optimize higher resilience and efficient paths with minimum latency to send the information between nodes.

1.2.1 Resilience

According to network perspectives, a resilient system can be defined as a system that has the ability to immediately recover to normal after the network has suffered cyber-attacks, node failures, or hardware issues. The self-recovery systems can not only provide normal services immediately after the threats occur but also identify vulnerable parts of the network [5]. In IoT networks, there are many IoT applications, which are required to operate in real-time. Thus, archiving the resilience of the communication system is crucial for successfully deploying and operating an IoT application. The principles of a resilience strategy for real-time networked systems must follow the framework: defend, detect, remediate, recover, diagnose, and refine [6]. In [7], Sterbenz has proposed “islands of resilience and corridors of resilience” to achieve resilience in a complex multilevel structure of the IoT network. To conclude, resilience is one of the most important characteristics of IoT networks to avoid the unavailability of IoT applications.

1.2.2 Latency

Network latency describes delays in communication over a network. It sometimes takes longer than expected time to transmit the packets of data from the source to the destination. The possible causes of network latency are overloaded network traffic, selection of transmission mediums, round trip time, and so on. With the emergence of IoT applications in terms of latency and reliability in communication networks, reducing latency needs to take priority to order to maintain the QoS of IoT applications. In recent years, many applications of IoT networks, such as self-driving vehicles, smart grids, and live streaming applications, may require Ultra-Reliability Low-Latency Communications (URLLC). URLLC is one of the most innovative technical approaches that is supported by 5G technologies. For low latency in wireless communications, some perspectives, including packet length, transmitters/receivers, and access protocols, need to be considered. For example, latency mainly depends on packet length because shorter packets can be processed more quickly than longer packets. The transmission delay, propagation delay, nodal processing, and queueing can be significantly reduced when simple and short packets are transmitted between the nodes. However, reliability and latency may be a trade-off because reducing the packet size for low latency can lead to decreased reliability [8]. In some IoT networks, the total transmission delay must be guaranteed in an interval time. Therefore, many latency reduction techniques have been proposed and published by researchers. Third Generation Partnership Project (3GPP) [9, 10] has stated that optimizing latency in wireless communications is a part of the evolution of 4G and LTE networks. In [11], the authors have demonstrated two potential latency reduction techniques for NB-

IoT. These are short transmission time intervals and semi-persistent scheduling. Ultimately, low latency is a desirable goal for all wireless networks to achieve because it provides a better experience for users.

1.2.3 Efficiency

In terms of efficiency in IoT network communications, we may need to take into account a variety of perspectives, such as energy or spectrum. However, this section will focus on the spectrum efficiency of IoT networks. When the number of nodes increases in the network, the number of required spectrum frequencies also increases. Therefore, it causes the spectrum frequencies (i.e., network channels) to be increasingly scarce. Similar to most existing wireless networks, IoT networks have to optimize spectrum efficiency. The spectral efficiency of wireless networks can be divided into two strategies. The first strategy is to deal with channel interference when two nodes are using the same channel to transmit the signals. The method must be applied to properly use available channels in the network to limit the negative impact of the interference. For example, by using power domain non-orthogonal multiple access multiplexing, Khan et al. have provided a novel resource management strategy to increase the overall spectrum efficiency in IoT networks [12]. Next, the second strategy is to detect both external and internal interferences occurring, called spectrum sensing. One example of spectrum sensing used to improve channel selectivity is demonstrated in [13]. To sum up, the efficiency of IoT networks is required to achieve for IoT applications to provide clients with the best service.

1.2.4 Proposed Solutions

In this thesis, we focus on improving the performance of IoT wireless networks. We must assure that high network connectivity and a low probability of the network being disconnected need to be achieved. In other words, no node is separated or unreachable from the network. Moreover, the links between each node (i.e., IoT device) must be operated effectively to supply enough connections in the network. Hence, in order to maintain a high level of network connectivity and overall performance, we propose a technical solution to effectively allocate channels of the network using the concept of graph theory. It is called graph coloring, where two adjacent nodes must use the distinct frequency spectrum for transmitting the signals. We then indicate the importance of the nodes in the network by measuring the eigenvector centrality scores. Based on the node centrality and the graph coloring, we next eliminate the least important nodes of the network to avoid interference. The concepts of graph coloring, eigenvector centrality, and node elimination will be presented in Chapter 3, and the simulations are done in Chapter 5.

In the real world, the nodes distribution (i.e., IoT devices distribution) is dynamic and unpredictable because of their mobility. It is important to keep updating a new network strategy in real-time. In other words, the routing optimization must change based on the status of the network at that time. Therefore, we also propose an additional solution for channel allocation using machine learning-based graph coloring, which can learn from the historical data and update a new strategy in the IoT network to optimally utilize the number

of available channels. We will discuss more details of the machine learning (ML) technique in Chapter 4 and the simulation in Chapter 5.

1.3 Outline and Contribution of Thesis

In this thesis project, we will implement a graph coloring approach for channel allocation in IoT networks. In addition, we propose machine learning-based graph coloring to optimize the resilience of the networks and maintain the QoS for IoT users.

In Chapter 2, we address the problem which can negatively impact IoT networks – wireless interference. Furthermore, we present some prior research to solve the IoT problems that can degrade the performance, as well as network connectivity. Lastly, we give a discussion of our solution approaches to avoid interference in IoT network access.

Chapter 3 demonstrates the graph coloring-based channel allocation. In addition, we present system models used to run our simulations and obtain the simulation results. We will present the following perspectives in the system models: node distribution using Point Poisson Process, graphs as wireless network models, modeling interference in IoT networks, node criticality, and elasticity theory. Our approach to suppress channel interference is eliminating the less important nodes from the network.

In Chapter 4, we introduce deep reinforcement learning-based channel allocation. The background of deep reinforcement is provided to deeply understand all required components that need to be defined. Also, we demonstrate the deep learning network

process to train a model. Accordingly, we define our system model with the state, the action, and the reward corresponding to wireless networks.

Chapter 5 demonstrates several perspectives of our system model. Next, we introduce a graph coloring algorithm for solving channel allocation in IoT networks. We discuss the graph coloring theory and a greedy graph coloring algorithm that is used to determine the chromatic number of a graph. Based on the graph coloring implementation, we reduce the number of colors labeled by applying node importance. In addition, the concept of a deep neural network is discussed before we propose our solution for channel allocation using deep reinforcement learning-based graph coloring in an IoT network. Also, using our approaches can increase and maintain a higher level of resilience in IoT networks.

Finally, in Chapter 6, we analyze our simulation results and arrive at our main conclusions about our contributions. In addition, we provide some suggestions for future research based on what we have done.

CHAPTER 2: INTERFERENCE IN IOT NETWORK ACCESS

2.1 INTRODUCTION

Outside interference in wireless networks, including IoT networks, is one of the most challenging obstacles to overcome. Interference has received much attention from researchers because network performance is influenced by this factor. Interference occurs when two or more devices transmit or receive a wireless signal using the same frequency spectrum. The quality of the wireless signal may be degraded due to network interference. Channel interference may come from common sources like Bluetooth devices, wireless video cameras, cellphones, and so on. Specifically, for IoT networks, since there are many IoT devices or sensors autonomously operating in a particular area to collect and transmit the measured data, wireless interference is a major problem, which prevents wireless communication between each node (i.e., IoT devices). In this thesis, we divide interference into two categories: external interference and internal interference. External interference is caused by devices coming from outside our network, while internal interference is caused by devices in our network. Therefore, external interference is more difficult to manage and reduce the interfering problem than internal interference. IoT devices that are participating in the network cause internal interference because they may use the same channel to transmit the data. In practice, wireless interference cannot be eliminated in the network. We can limit the adverse consequences of channel interference by channel allocation.

To solve the interfering problem of wireless networks, the devices must avoid transmitting and receiving the signals using the same frequency channel. In other words,

adjacent nodes must use different frequencies to send and receive wireless signals. Since the network channel (i.e., bandwidth) is a scarce resource, it is important to efficiently allocate the channel in the network to maintain the availability and maximize the resilience of the network connectivity. Therefore, the detection of interference is an applicable approach to avoid the channels that have been already used by other nodes. Therefore, channel allocation and interference detection approaches are generally proposed as a way of improving channel interference in the network.

Regarding external channel interference, since we cannot manage the components belonging to an outside network, including physical hardware (e.g., servers, routers, transmission medium, connecting devices, etc.) and software (e.g., operating system and network protocol), it is more feasible to avoid the channels that have been already taken by detecting and analyzing interference of other wireless networks. In particular, spectrum sensing is an effective technique for the detection and analysis of channel interference. Wireless interference is detected by hypothesis testing, which evaluates the presence and absence of the observed signal by comparing it to a pre-established threshold. If the output obtained from the observed signal is greater than the threshold, the presence of the signal is assumed. On the other hand, the absence of the signal is assumed if the output of the observed signal is less than the threshold. The general model of spectrum sensing is illustrated in Figure 2.1:

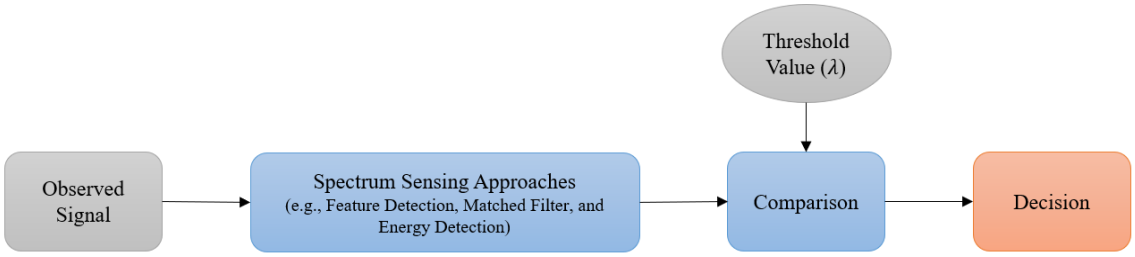

Figure 2.1: General Block Diagram of Spectrum Sensing

There are three common types of spectrum sensing approaches as follows:

- **Feature Detection:** According to [14], the principal technique of feature detection is finding particular statistical features in a given communications signal. These features usually have repeating characteristics in many telecommunication signals. The theoretical knowledge of second-order statistics, which is derived from the autocorrelation function, is applied to design a feature detector for spectrum sensing.

- **Energy Detection:** Energy Detection is one of the simplest approaches for detecting channel interference. The operation of energy detection is to use received signal power for identifying the presence or absence of a signal. The principle of this approach is to compare the average energy of IoT devices with the threshold value. The average energy of IoT devices can be computed by the following Figure 2.2 [15]:

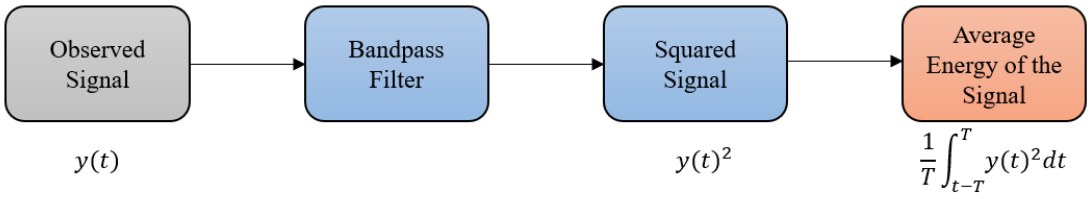

Figure 2.2: Energy Detection Block Diagram

In the diagram, the signal is represented by $y(t)$. The bandpass filter is used to capture the signal within an observed range of frequencies while eliminating unwanted frequencies of the signal. Then, the signal is squared for calculating the average energy of the signal.

- **Matched Filter Detection:** This approach firstly requires a matched filter based on the primary signal. Consequently, the matched filter detector needs to have prior knowledge of the primary signal. The matched filter correlates the input signal with time-shifted, and then the outcome is used to compare with the threshold in order to determine the presence or absence of the observed signal. The matched filter can perform well when the cognitive radio has a priori knowledge of primary waveform systems. Therefore, the results of the matched filter detection are not accurate if the provided information of primary signal systems is marginally incorrect [16]. The general model of matched filter detection is illustrated in Figure 2.3.

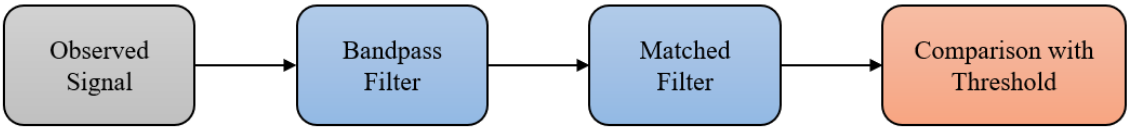

Figure 2.3: General Model of Matched Filter Detection

When it comes to internal interference, channel allocation is one of the best solutions. After we eliminate the channels that cause external interference, we can obtain a list of available channels that the nodes of the network can use to transmit or receive wireless signals to avoid interference. In order for all nodes to be assigned to channels, allocating the available channels needs to be optimized, but it must not cause interference. Three possible scenarios may happen when the channels are allocated to each node of the network:

1. **The number of available channels is fewer than the number of channels needed.** In this case, the wireless network is disconnected because no channel is currently available for the nodes of the network to transmit or receive the data.

2. **The number of available channels is greater than the number of channels needed.** In this situation, we want to optimize the utility of accessible channels by

finding the minimum number of channels used to avoid channel interference. The network, of course, is fully connected because all nodes are reachable.

3. **The number of available channels is equal to the number of channels needed.** This means the number of channels provides exactly the demand of the wireless network. The minimum number of accessible channels is equivalent to the number of channels needed. Therefore, the network is fully connected, but in the real world, it is not realistic because the number of IoT devices is always much more than the number of available channels.

Graph coloring is one of the most common techniques used for channel allocation. In Chapter 3, we will introduce graph theory and apply it to the process of allocating channels in a wireless network.

2.2 Impact of Interference on IoT Performance

In the previous section, we introduced the definition of channel interference and two technical approaches to tackle the interfering problem in wireless networks. Next, we will discuss the consequences of interference in IoT network access.

Since IoT nodes usually need to prioritize power efficiency, the power constraint restricts the transmitted power of each IoT node. On the other hand, the maximum operational distance, which is the maximum distance that a node has the capability to

transmit a wireless signal, must depend on the transmitted power of the node. In other words, maximum operational distance is directly proportional to transmitted power. Nevertheless, in order to maintain a high level of network connectivity so that one IoT node can be reachable from any other nodes belonging to the network, a node must have at least one link connection with another node. Along with the limited transmitted distance, IoT nodes may need to have a high node density to shape a fully connected IoT network. However, as the node density of the IoT network increases, the probability of interference occurring increases as well. Therefore, all scenarios need to be carefully considered to avoid wireless interference in IoT networks.

It is evident that the link connectivity of wireless networks plays an important role in IoT performance. Interference is one of the major concerns of wireless networks that leads to poor IoT performance and a low level of resilience. The actual impact of the interference can be degradation of the link performance, as well as node denial, which means the node is unable to access the network. In other words, due to channel interference, a node can be completely disconnected from the remaining nodes of the IoT network. From a wireless network perspective, the network is defined as a disconnected network topology when one of the nodes belonging to the network cannot be reachable. Increasing the resilience of IoT networks is expected to maintain high link connectivity. Thus, link connectivity and resilience of IoT networks are directly proportional to each other.

2.3 PRIOR RESEARCH WORKS

Optimizing the resilient communication of wireless networks has many positive effects on various fields, including academia, commerce, and the military. The diverse set of wireless technologies can be used to optimize the overall performance of the network. In this section, we will review several prior research works which focus on resource allocation approaches and graph coloring algorithms for wireless networks. All these works aim to improve the resilience of wireless communication by minimizing channel interference.

2.3.1 Resource Allocation Approaches

In recent years, with the rapid growth of technological advances in wireless communications, network optimization has received more attention from researchers. Guaranteeing QoS is one of the major concerns for the services in wireless communications. Maintaining a high level of performance for wireless communication depends on various parameters of the QoS (e.g., data transmission rate, transmission delay, throughput, and so forth). Channel allocation is the most common problem in wireless communications, including IoT networks. Since IoT applications have increased gradually, IoT networks are expected to increase IoT devices, which are required to connect to the IoT network. Nevertheless, the number of channels (i.e., bandwidth) is a very scarce resource. Effectively utilizing the available channels is necessary to guarantee the performance of the links.

Traditional Channel Allocation Methods

In fact, the bandwidth can be divided into a set of available radio channels, which are used for communication between IoT devices and base stations. The techniques that are used to divide the radio spectrum into different channels are frequency division (FD), time division (TD), and code division (CD). In FD, the total bandwidth available is divided into a set of non-overlapping frequency bands, which ensures that none of the channels have the same frequency bands. TD is a technique for dividing the channel's signals into different time slots for transmitting. In CD, the channel is separated by using different coding schemes. Other techniques tend to be designed based on the combination of the three techniques mentioned above.

When it comes to channel allocation, there are three categories of channel allocation techniques, which are used widely: statics channel allocation (i.e., fixed channel allocation), dynamic channel allocation, and hybrid channel allocation. In fixed channel allocation (FCA), the strategy is to allocate the same number of channels to each cell (i.e., a node or IoT device) for the channels to be uniform in the network. For example, the FCA algorithms are simple borrowing methods and localized channel-sharing methods. In contrast to FCA, dynamic channel allocation (DCA) is a technique to encounter a dynamic wireless network, in which the nodes (i.e., IoT devices) are not fixed. Hence, the channel allocation must consider the current network conditions to optimize the performance. Compared to FCA, DCA has a better performance when the network traffic is light. Some proposed DCA algorithms are dynamic load balancing based DCA method, dynamic

frequency-time channel allocation, and reused partitioning based on dynamic channels [17].

Separately, many other algorithms of channel allocation have been proposed by researchers. Dealing with channel allocation problems in a cognitive radio network, Sharna et al. proposed a game theory to implement [18]. Game theory has been applied to various areas of human activity, such as economics, finance, regulation, and so forth. The goal of the game theory algorithm in a cognitive radio network is to maximize the overall performance of the network by achieving a Nash Bargaining solution, which is a fair outcome that all players of the game agree. It has three components: 1) a set of players, 2) a set of actions, and 3) a utility function. The utility functions are evaluated by the signal-to-interference-plus-noise ratio (SINR). Furthermore, as stated by [19], Papazoglou et al. gives a survey of other channel allocation approaches, including genetic algorithms, swarm intelligence, and ant colony optimization. Both methods are based on computational intelligence. In other words, the best solutions for the optimization of channel allocation can be implemented based on the condition of the network. In addition, Zhang et al. have proposed a solution for channel allocation, which is the frequency spectrum division of a 30-MHz channel. The 30-MHz channel will be divided into three different ranges to use on purpose. The priority of network traffic determines which range of the frequency band can be assigned. This solution has been applied to car-to-car communication called Intelligent Transportation System (ITS), which is a part of the IoT application [20].

Resource Allocation-based Machine Learning

In recent years, the applications of ML have encompassed various fields from academia to industry. Since ML has high applicability, it has received much attention from researchers to improve the performance of existing solutions. Therefore, many works focus on developing and implementing ML techniques to effectively allocate resources in wireless networks. From a wireless network perspective, resources refer to power, energy, time slot, or bandwidth (i.e., channels). Besides that, since wireless networks consist of mobile devices, it is difficult to determine the best routing options. However, ML can learn from the historical routing paths or collected data to achieve the best solution. In wireless networks, there are many different ML applications developed to allocate power [21, 22] and channels [23]. In addition, the deep neural network has been successfully applied to cognitive radio modulation recognition [24]. In this work, we also develop a deep reinforcement learning model to determine channel allocation in IoT networks.

Regarding channel allocation approaches, adjacent channel interference and co-channel interference are two major interference problems that need to be taken into account. While adjacent channel interference is a result of a signal in an adjacent channel, co-channel interference is interference caused by two nodes of the network using the same channel to transmit the signals.

2.3.2 Graph Coloring for Wireless Networks

One of the potential solutions for channel allocation in wireless networks is using graph coloring algorithms. A wireless network can be modelized as a graph with a set of vertices and a set of edges (see Figures 2.4 & 2.5). In a modeling graph, a vertex is a node

or an IoT device, and an edge is a link, which is used to connect every two vertices. The fundamental idea of graph coloring theory is the labeling of vertices or edges in order for two adjacent vertices or edges to avoid having the same color. In other words, each channel of the network is assigned to each color. The major goal of graph coloring is to minimize the number of colors used for labeling vertices or edges. The smaller number of colors, the smaller number of channels that are used to transmit data in the network. Since minimizing the number of colors is an NP-hard problem, almost all available graph coloring algorithms are sub-optimal solutions, and they have a high level of complexity. In graph theory, the minimum number of colors needed to label all vertices is called the chromatic number. In past decades, various researchers have proposed different graph coloring algorithms to optimize and determine the chromatic number. These proposed methods for graph coloring are edge table scheme [25], algorithm ECG [26], Dsatur [27], tabu search [28], simulated annealing [29], and greedy algorithm [30].

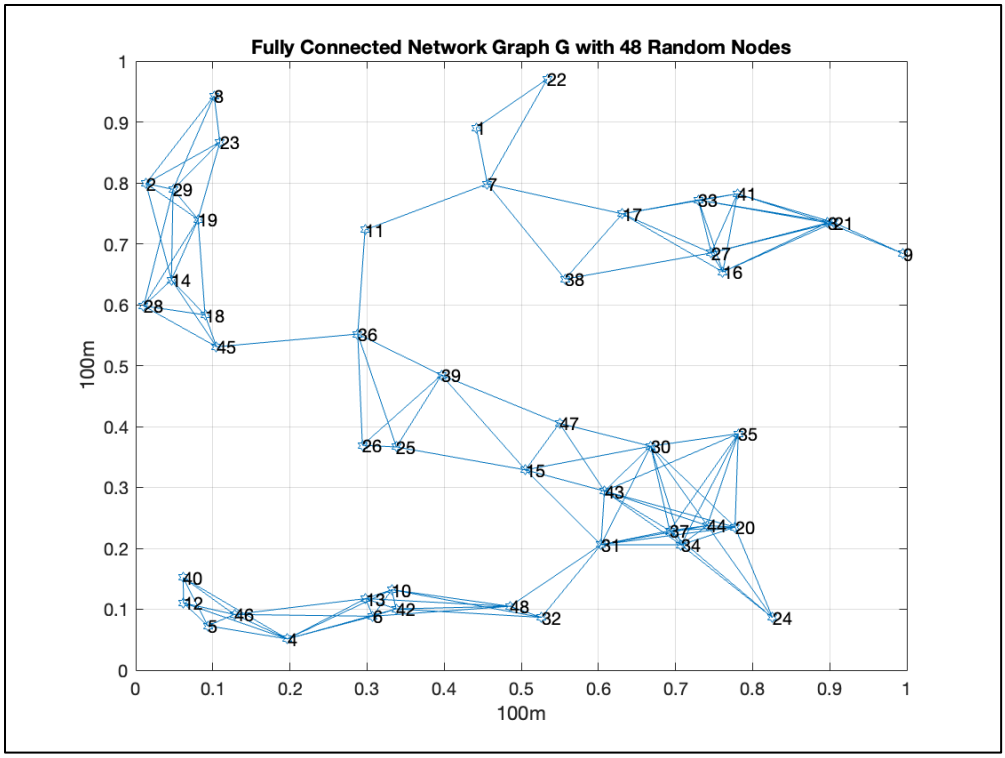

Figure 2.4: Fully Connected Wireless Network as Graph G

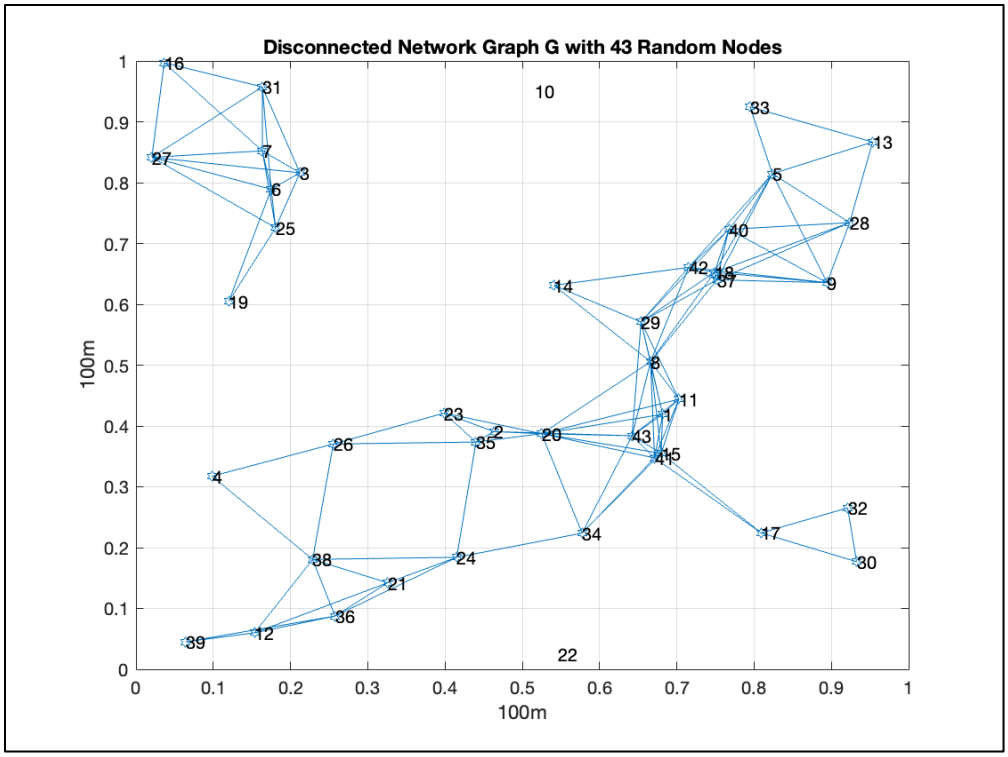

Figure 2.5: Disconnected Wireless Network as Graph G

When it comes to graph coloring algorithms in wireless communications, their typical applications are graph coloring-based spectrum sharing [31, 32] and resource allocation based on graph coloring [33, 34, 35]. More discussion of graph coloring will be provided in Chapter 3, and our simulations for channel allocation using the graph coloring algorithm will be in Chapter 5.

2.3 SOLUTION APPROACHES

We have discussed the constraints of IoT networks, as well as some existing technical solutions to solve these problems. IoT networks may contain thousands of nodes, which can transmit between the source and the destination using other nodes as relays. Since this concept is a particular situation of wireless multi-hop networks, in this thesis project, we will focus on a multi-hop network. In general, a multi-hop network uses a single spectrum frequency for all nodes and enables spectrum sharing in case of a multiple access approach. Nonetheless, this approach leads to interference across the network. Thus, the network connectivity can be degraded, and the network cannot provide a high level of resilience. Figure 2.6 illustrates an example of a multi-hop IoT network. In this project, we have attempted to answer the problems of channel allocation to avoid channel interference. Hence, some questions have arisen: How do we ensure that the channel allocation over the network is optimal? How do we ensure that any solutions deal with a dynamic and heterogeneous wireless network like an IoT network? How do we ensure that interference over the network minimizes?

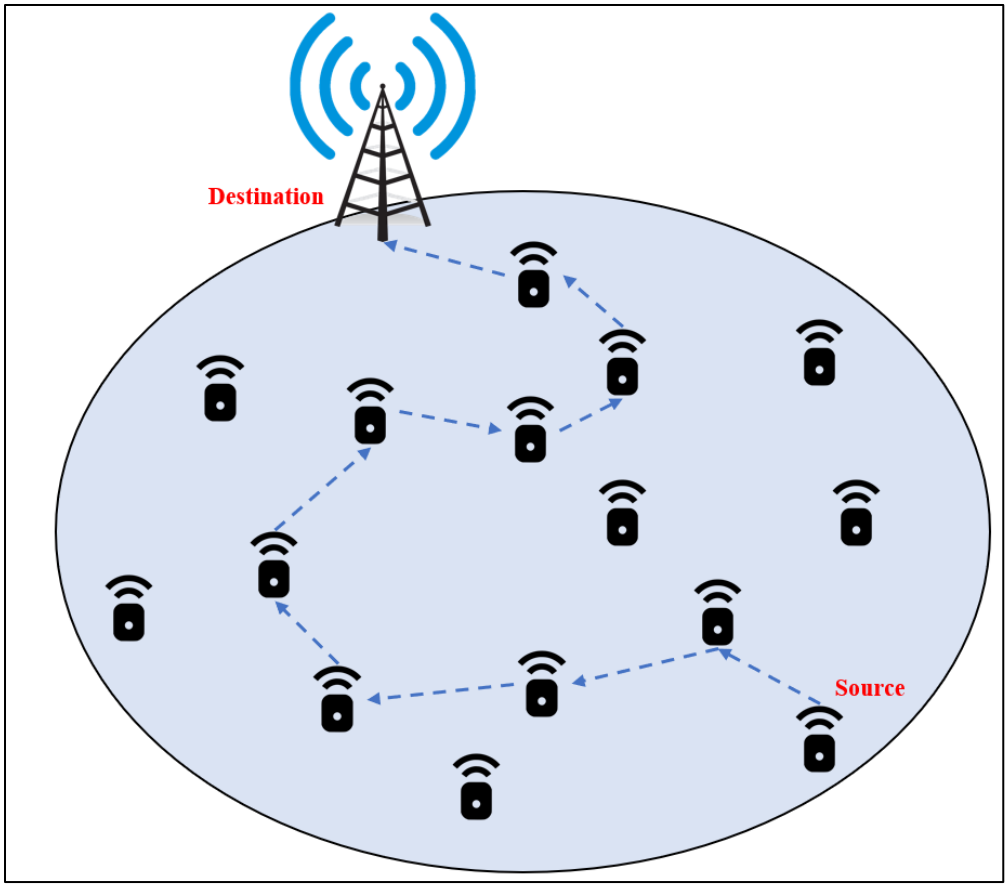

Figure 2.6: Multi-hop Routing

Our proposed solutions require that each node sends and receives on multiple channels (i.e., spectrum frequencies), and no two adjacent nodes use the same channel. We propose two different approaches based on graph coloring. The first approach is to implement the greedy graph coloring to determine the number of channels assigned to each node. We then decrease the number of channels (i.e., minimize interference) in the network by reducing the least important nodes. The second approach we propose is combining deep

deterministic policy gradient and graph coloring to efficiently allocate available channels in the network. Relied on the power of ML, the number of channels can be efficiently allocated to each node in order to avoid channel interference.

CHAPTER 3: GRAPH COLORING-BASED CHANNEL ALLOCATION

3.1 INTRODUCTION

In order to address our issue of enhancing the network connectivity of an IoT network, we concentrate on frequency diversity, which allows the same message signal to be transmitted using multiple carrier frequencies [36], by channel allocation. By varying the frequencies that the network uses, this strategy improves network connectivity while reducing interference risk.

In wireless networks, optimal resource allocation is important for effective wireless spectrum use. This is still the case for either homogeneous or heterogenous wireless networks. Network resource refers to the bandwidth and the power that is assigned to users to transmit their data between the source and destination [37]. Therefore, resource allocation aims to distribute radio frequencies to specific individuals in a way that maximizes system capacity while maintaining the quality of service.

There are many different techniques to allocate network resources. One common resource allocation approach is a wireless network using only a single frequency spectrum to distribute to all nodes in the network. Nonetheless, sharing a frequency spectrum in the network can increase the risk of interference, as well as multiple access problems. Another straightforward approach is to share the total bandwidth W that is available equally across N nodes in the network. As a consequence, each node has $\frac{W}{N}$ Hz of bandwidth. However, since the spectrum resource is scarce, this approach may encounter a problem regarding

scalability in a large wireless network. Practically, the number of available channels may not have enough to allocate to all nodes. Therefore, we assume that the number of nodes in the network is much more than the number of channels available. Channel allocation now can turn into an optimization problem in mathematics. In this chapter, we describe various aspects of our system model in more detail, including node distribution, graphs as wireless network models, modeling interference in IoT networks, node criticality, and elasticity theory.

3.2 SYSTEM MODEL

In the previous chapters, we have described our system model at a high level, which focuses on improving the high level of network resilience for IoT networks, as well as maintaining the QoS. Therefore, in order to achieve this goal, we need to avoid channel interference between IoT devices of the network. To put it differently, effectively allocating channels in the network can minimize network interference. In this section, we next describe various aspects of our system model in more detail, including node distribution, graphs as wireless network models, and node criticality.

3.2.1 Node Distribution

In general, wireless networks are dynamic and unpredictable because the locations of the nodes keep changing over time. Therefore, we have decided to use the Poisson Point Process (PPP) to assign the distributed locations of the nodes in the wireless network. We

assume that in a given area $A \in R^2$ the N nodes are distributed following the PPP process. The probability $P(N)$ that N nodes are allocated in A is expressed as follows:

$$P(N) = \frac{(\lambda A)^N e^{-\lambda A}}{N!} \tag{3.1}$$

From Equation (3.1), λ is the node density in a unit area of the network. The node density represents the relationship between the location and activity of the nodes in the network [38]. Accordingly, the node density is directly proportional to the number of nodes in a given area. It is also clear that the initial value of the node density that we have assigned plays an important role in analytical and simulated calculations. Figure 3.1 illustrates an example of a wireless network with 45 nodes distributed using PPP in a given area of 100,000 m^2.

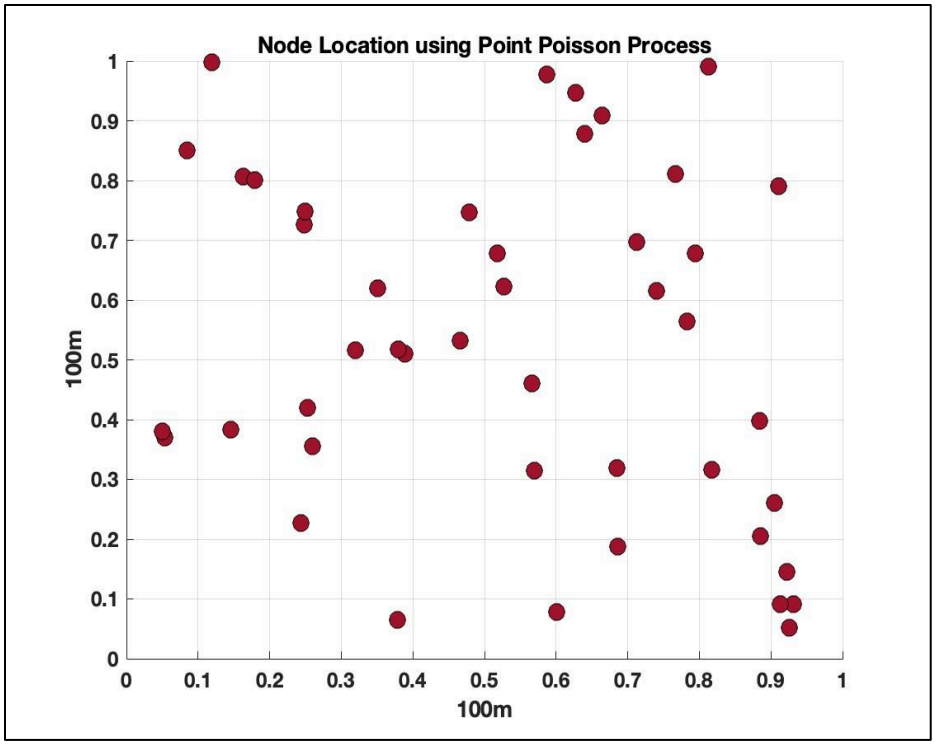

Figure 3.1: Node Distribution using Point Poisson Process

Each wireless network has a specific range of operation, within which the antennas can have enough transmission power to transmit the data from the source to the destination in the network. For example, routes in a Wi-Fi network normally operate for point-to-point communications in a maximum range of 50 meters. Another example is an Ad Hoc network, which is a type of LAN. In the Ad Hoc network, the operational range is approximately 100 meters, which is capable to transmit data longer than the Wi-Fi network. An IoT device can have a link to another device only if they are in the operational range of each other. On the other hand, the distance between every two nodes (i.e., IoT devices) can

be used to compute the power consumption and the signal-to-interference-plus-noise ratio (SINR).

3.2.2 Graphs as Wireless Network Models

In communications, mathematical graphs are used to represent wireless communication networks with a set of vertices and a set of edges. In the graph, a set of vertices V represents a set of nodes (i.e., IoT devices), while communication links between each node are represented by a set of edges E. We assume that undirected graphs, in which all the edges are bidirectional, are used in this project. An example of a wireless network graph is demonstrated in the following figure:

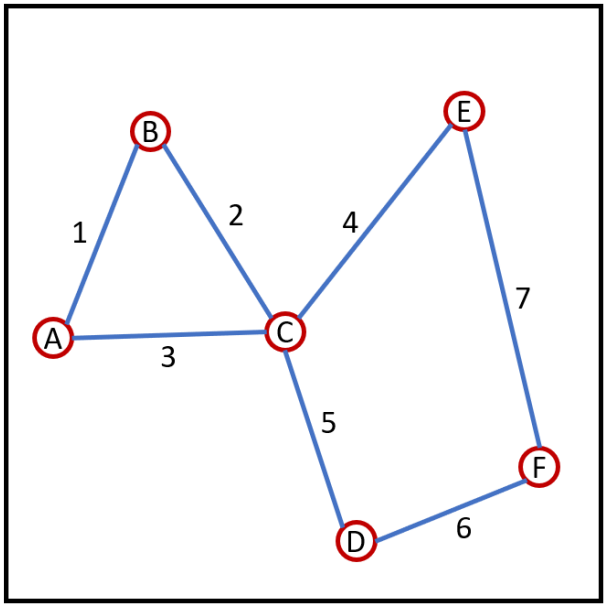

Figure 3.2: A Wireless Network Graph

According to Figure 3.2, the network graph consists of a set of vertices is $V = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$ and a set of edges is $E = \{1, 2, 3, 4, 5, 6, 7\}$.

In graph theory, there are a variety of ways to critically analyze graphs, such as degree matrix, transition matrix, adjacency matrix, and incidence matrix. An adjacency matrix is a squared matrix, which is used to demonstrate the relationship between nodes and how the communication links are used to connect them together. With the network consisting of *N* nodes, we can have a list of nodes { node 1, node 2, node 3, … node N}. Additionally, we can have its adjacency matrix $A_{N\times N}$ as follows:

$$A_{N\times N} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NN} \end{bmatrix}$$

To simplify, the adjacency matrix can be expressed as the following equation:

$$A_{ij} = \begin{cases} 1, & if\ ij \in E\ (i\ and\ j\ are\ adjacent) \\ 0, & otherwise \end{cases} \tag{3.2}$$

where $1 \leq i \leq N$ and $1 \leq j \leq N$. For example, the adjacency matrix of the wireless network in Figure 3.2 is the squared matrix as follows:

$$A_{6\times 6} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The diagonal of this adjacency matrix has all zeros. It means that none of the nodes do have a loopback connection. In addition, the adjacency matrix is symmetric.

Based on the adjacency matrix which represents the link connections between nodes, we also define the maximum operational distance which is the maximum distance that a node can transmit or receive the signal to/from any node. Therefore, the definition of a maximum operational distance matrix *D* can be derived as follows:

$$D_{ij} = \begin{cases} d_{ij}, & ij \in E \ (i \ and \ j \ are \ adjacent) \\ 0, & otherwise \end{cases} \tag{3.3}$$

Similar to the adjacency matrix, the maximum operational distance matrix has the same properties. However, the only difference between the two matrices is that the maximum operational distance matrix determines not only the connections between nodes but also the distance of these connections.

3.2.3 Modeling Interference in IoT Networks

Next, we will demonstrate the general model of interference on IoT networks. This model will greatly contribute to channel allocation. Also, this model is one of the parts that are used to evaluate the performance of our system later.

Since a multi-hop network is one of the most common wireless networks that are used in IoT networks, we have applied the characteristic of the multi-hop network in our system model. In the multi-hop network, any destination node can be reached from a source

node by using other nodes as relays. Our system model is an IoT network, which consists of a total of *N* nodes. We assume this network has a total available bandwidth of *W* Hz. Also, other similar networks use the same spectrum, which results in a higher probability of interference occurring. We assume there are *M* available channels, which will be allocated to *N* nodes of the IoT network. However, since the channels are limited, we can have a relation between *M* available channels and *N* nodes of the network:

$$M < N, \forall N \tag{3.4}$$

This expression means some nodes of the network must use the same channel as each other because there are not enough available channels covering all nodes of the network. However, using the same frequency spectrum is also a major reason causing the interference problem. Therefore, the available channels must be allocated properly to minimize channel interference.

In the modeling of interference, we consider both external and internal interference. We denote a node, which belongs to our IoT network, by A_i, where $1 < i < N$. Meanwhile, we denote a node, which belongs to other interfering networks, by B_j, where $1 < j < K$. Since we have no a priori knowledge of the nodes from other networks, we only pay attention to the nodes causing the interference to our IoT network, and we assume *K* is the total number of interfering nodes from other networks.

In our IoT network, we assume that each node has equal transmitted power $P_t(A)$. We also assume that a node in the interfering network has a transmitted power. From the

wireless communication perspective, there is a direct relation between the transmitted power and the distance. This relationship is expressed by a Free-Space Path Loss (FSPL), which is used to calculate the loss of a waveform strength when a transmitted antenna transmits the waveform to a received antenna through free space. The free-space path loss derives from the Fiirs transmission formula [39] as follows:

$$FSPL = \left(\frac{4\pi d}{\lambda}\right)^2 = \left(\frac{4\pi df}{c}\right)^2 \tag{3.5}$$

where

d is the distance between two nodes (m)

λ is the signal wavelength (m)

f is the signal frequency (Hz)

c is the speed of light (m/s)

Equation (3.5) is also written in terms of unit dB:

$$FSPL_{dB} = 20 \times \log_{10}(d) + 20 \times \log_{10}(f) + 92.45 \tag{3.6}$$

The transmission loss between node i and node j is denoted by the free-space loss channel gain $\delta_{i,j}$. Meanwhile, the FSPL channel gain of other interfering nodes is γ. Therefore, the following is a summary of all parameters we assume:

$P_t(A_i)$ Transmitted power of node i in our IoT network A

$P_t(B_k)$ Transmitted power of node k in other interfering networks B

$\delta_{i,j}$ Free-space path loss channel gain between node i and j in network A

γ Free-space path loss channel gain between the receiver and an interfering node

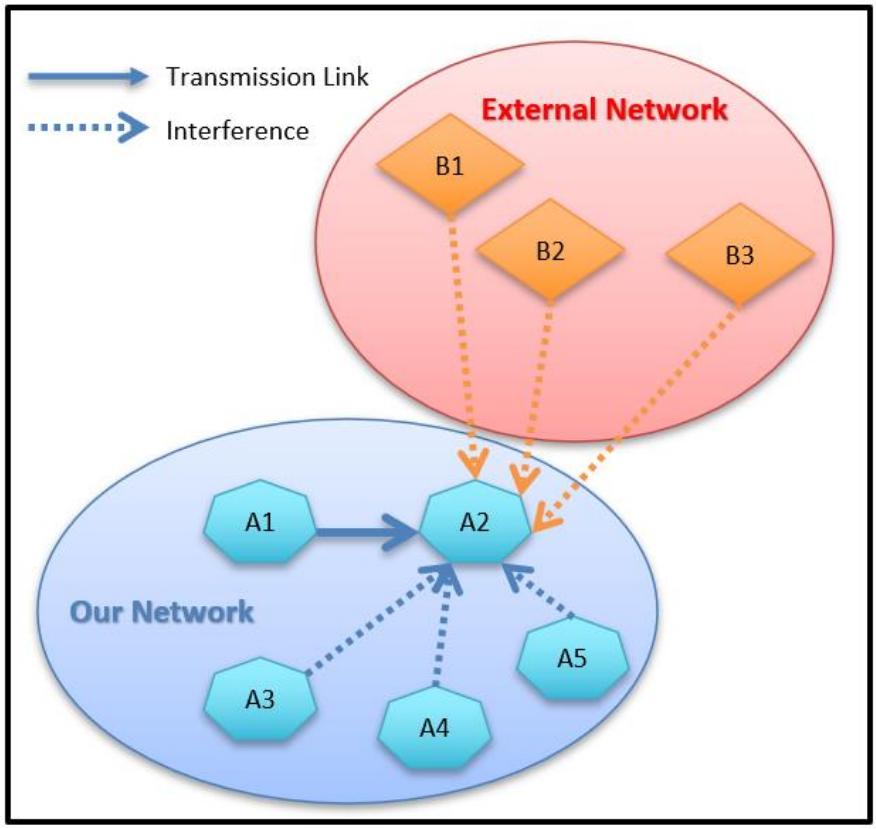

Figure 3.3: Interference Model

The concept of external and internal interference is illustrated in Figure 3.3. Accordingly, the solid line represents a signal transmission link between the nodes in our network. Meanwhile, the dashed lines represent signal transmission links, which are interfering links occurred by interfering nodes from other networks impacting the received node of our network. In our network, the signal received by transmitting from a node i to a node j is denoted by $P_t(A_i)\ \delta_{i,j}$. In addition to other networks, the signal received by transmitting from an interfering node k to a received node j in our network is denoted by $P_t(B_k)\ \gamma_{k,j}$. Aside from Additive White Gaussian Noise (AWGN), interference from both internal network nodes and external network nodes can deteriorate the signal. We demonstrate that the interference caused by the nodes of network A as $[A_1, A_3, A_4, A_5, \dots, A_n]$ can directly impact node A_2. On the other hand, the nodes of outside network A (i.e., network B) as $[B_1, B_2, B_3, B_4, \dots, B_k\ ]$ cause to the interference at node A_2.

It is clear that channel interference can degrade the quality of the received signal, as well as the performance of communication links in the network. Therefore, we need to have a measurement to assess whether the signal is accurate after traveling from transmitter to receiver. According to the wireless communication perspective, the Signal-to-Interference-plus-Noise ratio (SINR) is used to measure the strength of the desired signal to the interference and unwanted signal (i.e., noise). Since the network is complicated with various variables that need to be considered, the SINR equation is as follows for node j of network A receiving a transmission link from node i:

$$SINR_j = \frac{P_t(A)\delta_{i,j}}{\sigma^2 + \sum_{p\in A, p\neq i} P_t(A)\delta_{p,j} + \sum_{k\in B} P_t(B)\gamma_{k,i}} \quad (3.7)$$

Every wireless network (including IoT networks) must deal with interference, one of the most challenging issues. By effectively avoiding the assignment of conflicting channels, called channel allocation, the interference between the nodes in network *A* can be solved. Since these nodes belong to the same channel, which is controlled by IoT management, it is feasible to allocate the channels to avoid interference. Regarding interference from outside the IoT network, spectrum sensing techniques must be applied to identify which channels cause the interference to our IoT network. In the next section, we will present a solution strategy to minimize the interference experienced at each node.

3.2.4 Measuring Node Criticality

Some nodes have more influence on the performance of the network than others. Therefore, nodes in the network vary in their level of importance. Measuring the node criticality (i.e., node importance) can be a way to evaluate how a node contributes to wireless network connectivity. The importance of the nodes is determined by the nodes' locations and the connections with their neighboring nodes. According to the graph theory perspective, the node importance is usually indicated by degree centrality, closeness centrality, betweenness centrality, and the number of spanning trees reduced [40].

In some situations, we cannot guarantee that all nodes operate effectively in the network. Several nodes may be denied access due to channel interference that leads to the

resilience reduction of link connectivity. Therefore, to maintain a high level of resilience in the network, it is necessary to measure the node criticality to identify which node needs to be protected. On the other hand, some nodes, which are not worthwhile, may be removed to avoid channel interference. In this thesis, we measure the node criticality by using the eigenvector centrality of the network.

Unlike degree centrality, which simply determines the number of edges a node has in terms of ranking [41], eigenvector centrality can indicate the importance of these edges. To put it in another way, degree centrality only considers the number of communication links a node has, but eigenvector centrality considers both the number of neighbors and the neighbors' links. According to the principle of eigenvector centrality, a node has a high score of eigenvector centrality if it has many communication links to other important nodes. On the other hand, a node that has many connected links to its neighbors does not necessarily have a high score of eigenvector centrality. Next, we assume that a wireless network graph $G := (V, E)$ contains N nodes and the adjacency matrix $A(G)$. The formula to calculate the eigenvector centrality x_i of node i is represented as

$$x_i = \frac{1}{\lambda}\sum_{j=1}^{N} A_{ij}x_j \tag{3.8}$$

where A_{ij} is the elements of the adjacency matrix $A(G)$ and $\frac{1}{\lambda}$ is proportionality factor with λ a constant. Accordingly, the eigenvector centrality x_i is dependent on the sum of the

centrality scores of all nodes communicated with node i. The equation of the eigenvector can be rewritten in matrix notation as

$$\boldsymbol{Ax} = \boldsymbol{\lambda x} \tag{3.9}$$

where $\boldsymbol{A}$ is the adjacency matrix, $\boldsymbol{\lambda}$ is a vector of eigenvalue calculated from the adjacency matrix $\boldsymbol{A}$, and $\boldsymbol{x}$ is the eigenvector corresponding to the largest eigenvalue calculated obtained from $\boldsymbol{\lambda}$. The eigenvector centrality of a wireless network graph with 79 total nodes randomly connected is illustrated in Figure 3.4. The importance of the node is indicated from the lowest (blue) to the highest (red) in the color bar of the figure.

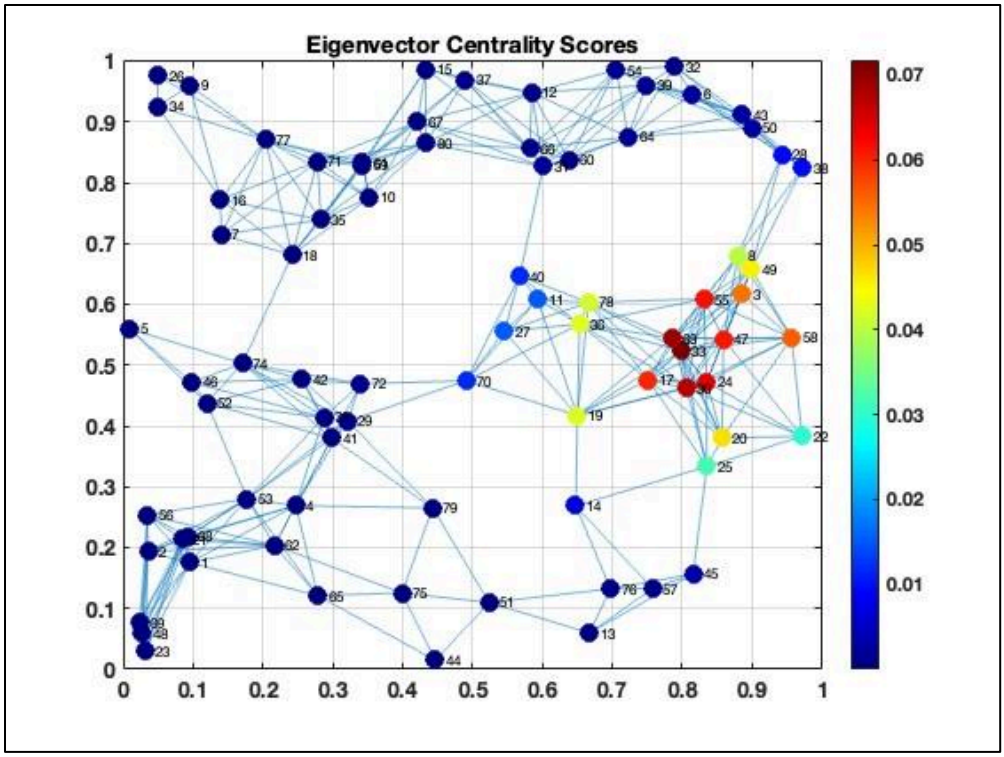

Figure 3.4: Eigenvector Centrality of a Random Wireless Network Graph

3.2.5 Elasticity Theory

According to our proposed solution, we desire to eliminate several less important nodes, which are based on eigenvector centrality, so that the total number of channels allocated to each node can decrease. In other words, minimizing the number of channels can avoid interference, but the network connectivity can also be affected in terms of the total number of nodes, the number of communication links between nodes, and the average number of nodes connected to each node in the network. Therefore, we introduce elasticity theory to present the relationship between node elimination and network connectivity.

In business and economics, price elasticity of demand refers to the percentage change in demand divided by the percentage change in price [42]. The following equation expresses the calculation of the price elasticity coefficient:

$$E = \frac{\% \triangle Q}{\% \triangle P} \tag{3.10}$$

The amount demanded and the price change proportionally when the price elasticity coefficient equals one. As a result, while setting prices, the firms frequently adopt the going rate. When the price elasticity coefficient is smaller than one, it demonstrates that if the company adopts a low-price strategy for their products, the demand to purchase the products will not increase significantly, but profits would decline. In contrast, when the price elasticity coefficient is greater than one, it means that the marginal change in price can produce a larger change in demand. This shows that the overall revenue may increase.

The price elasticity of demand is applied to our simulations in order to provide analytical insight into the overall profitability in case of node elimination and required channel reduction.

3.3 GRAPH COLORING THEORY

As we have described, wireless networks can be modeled as graphs. In the next section, we describe our proposed channel allocation based on graph coloring theory to avoid interference. Before we propose our technical solution, we will quickly introduce graph coloring.

Considering that we have a graph $G := (V, E)$ and a set of available colors $\{1, 2, 3, 4, \ldots M\}$, there are two approaches to color graph G. The first graph coloring approach is labeling the vertices of the graph using one of the colors belonging to the set of available colors, which is called vertex coloring. The vertices are colored following the rule that no two adjacent vertices of the graph are allocated the same color. The second approach is called edge coloring. It has a similar concept to vertex coloring. All edges of a given vertex must have distinct colors. The examples of vertex coloring and edge coloring are demonstrated in Figures 3.5 and 3.6, respectively.

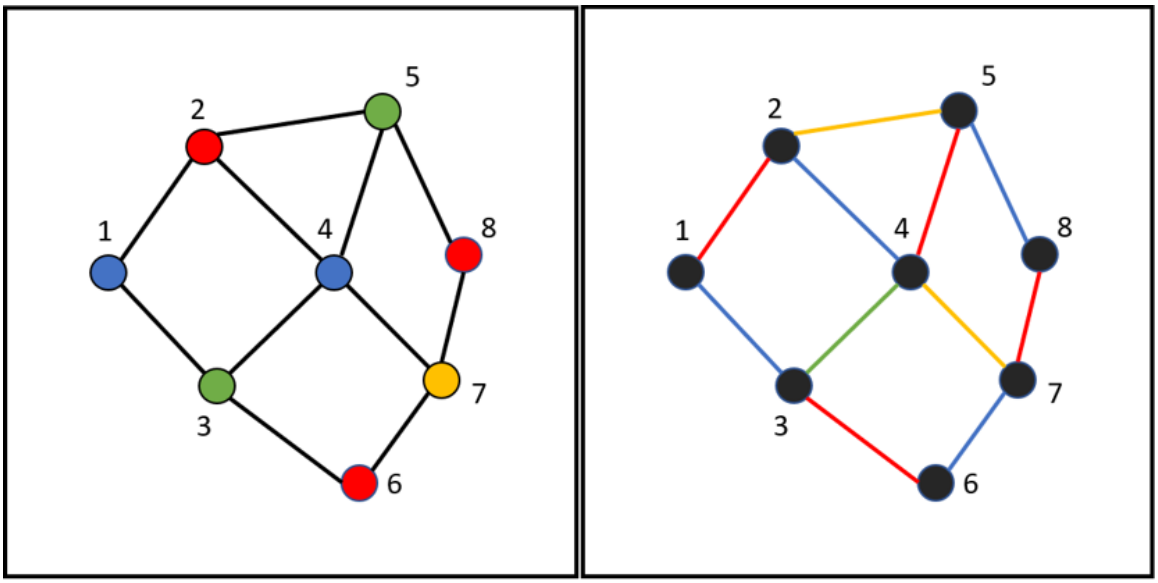

Figure 3.5: Vertex Coloring (left) and Edge Coloring (right)

In graph theory, the minimum number of colors used to label the vertices or edges of a graph is called the chromatic number, denoted by $\chi(G)$. In several particular cases of a large wireless network graph, which contains thousands of vertices and edges with a complex structure, determining the exact value of chromatic number $\chi(G)$ can be difficult. From the wireless network perspective, the chromatic number $\chi(G)$ represents the number of channels that are used to transmit or receive the data in the network. Therefore, determining the number of colors used is important to exactly identify the number of channels. Even though it is difficult to determine exactly the chromatic number, this number can be limited by an upper bound. Brooks's theorem states that the maximum degree of the network graph G, denoted by $d_{max}(G)$, is equal to or greater than the chromatic number $\chi(G)$ [43]. It can also be written in form of inequality:

$$\chi(G) \leq d_{max}(G) \tag{3.11}$$

The maximum degree of the network graph G is equivalent to the maximum number of colors that graph G needs. Besides that, determining $d_{max}(G)$ plays an important role in designing the training model in our ML section later. There are several algorithms to color a graph, as we have mentioned in Section 2.3.2 of Chapter 2. In this project, we implement a greedy coloring algorithm for graph coloring.

3.3.1 Greedy Graph Coloring for Channel Allocation

As we have mentioned previously, the minimum number of colors used to label the vertices or edges of a graph represents the number of channels. Therefore, for the channel

allocation problem, the desired goal is to minimize the number of colors required. In other words, the smaller number of channels used, the less interference affects a given node of the network.

Firstly, we implement the greedy coloring algorithm for determining the minimum number of colors required (i.e., chromatic number $\chi(G)$). There are two steps for graph coloring.

1. Graph Generation: This step is to generate the graph G with a set of vertices, which relate to each other by a set of edges. The graph represents an IoT network, which consists of nodes and communication links. The locations of the nodes are distributed based on the Poisson Point Process (as defined in Section 3.2.1). We assume that each node in the network uses a fixed transmitted power to send the data. Therefore, the communication links between the nodes are generated based on the maximum operational distance (as defined in Section 3.2.2) and the measurement of SINR (see Section 3.2.3). In other words, with the fixed transmitted power, we can figure out the maximum operational distance and acceptable SINR for a received signal.

2. Graph Coloring: This step is to properly color the vertices of the graph generated in the previous step.

Now, we specifically describe how to implement the greedy coloring algorithm in our graph. After the graph is generated, we have a set of nodes, an adjacency matrix, and a maximum operational distance matrix. The greedy coloring algorithm starts with a given

set of nodes and a set of available colors. Also, the relationship between a given node and its neighboring nodes is represented by the adjacency matrix. The greedy coloring initially assigns color 1 to node 1 of the network. Next, the adjacent nodes of node 1 are identified to assign the different colors such that no adjacent nodes of node 1 have the same color as node 1. Then, we continue to assign the remaining nodes in a similar way until all nodes have their colors. The total number of colors used is the chromatic number $\chi(G)$. Also, the number of available channels depends on the chromatic number $\chi(G)$. The overall algorithm is presented as follows:

Algorithm 1: Graph Coloring

Input: Graph G with the Adjacency Matrix *A(G)*

STEP 1: Find number of nodes in graph G

Define number of nodes in graph G: **numberOfNodes**

STEP 2: Initialization

Initialize zero vector of length **numberOfNodes**, called **color**

Initialize **totalColor** = 1, defined as total number of colors used for graph G

STEP 3: Use a loop to assign the color for all nodes of graph G

for *k = 1 to* ***numberOfNodes*** **do**

Create a vector **colorOfNeighbors**, defined as the color of the neighbors

Assign the smallest value of **color** that is not used by graph G, called **currentColor**

if ***currentColor*** *is empty* **then**

Increment **totalColor** by 1

currentColor = **totalColor**

else

Do nothing

end

Set the k^{th} value of color equal to **currentColor**

end

Result: The total number of colors used in graph G

In order to achieve the desired goal of minimizing the number of colors used, we next consider removing several nodes, which are less important, based on the measurement of node criticality (as discussed in Section 3.2.4). In other ways, removing several nodes, which have low influence on the network, can avoid interference, but also maintain a high level of network connectivity.

Our proposed solution for the channel allocation problem is to combine the greedy graph coloring algorithm with the node importance to minimize the number of colors used to label the nodes.

3.4 Node Elimination-Based Graph Coloring for Channel Allocation

After implementing the greedy coloring algorithm to determine the chromatic number of the network graph, we aim to reduce this chromatic number to suppress channel interference. To achieve this, we will remove several nodes from the network graph. However, we need to carefully consider removing the nodes from the network because it can be related to network connectivity (i.e., some nodes might be unreachable from the remaining nodes). Hence, some questions have arisen: How should the nodes be removed from the network so that the maximum number of colors can be reduced? Which nodes should be eliminated to achieve the maximum gain from this technique? and How many nodes should be removed?

The mandatory requirement is that the wireless network must be fully connected. In other words, all the remaining nodes must be reachable after several nodes are removed

from the network. If the network is disconnected or partly disconnected, removing the nodes to reduce the number of used channels is meaningless. Therefore, to answer the questions that have arisen, we propose to eliminate several less important nodes so that it does not significantly affect the network connectivity. The node importance is measured by using the eigenvector centrality technique we have introduced in Section 3.2.4. The gain of this solution is defined as the percentage of color reduction (i.e., channel reduction).

3.5 CONCLUSION

As described earlier, IoT networks must maintain a high level of resilient communication. IoT networks use a single frequency for all nodes that enable spectrum sharing. Spectrum sharing is unquestionably the best option for the IoT due to the dramatic growth of IoT devices, but the scarcity of spectrum resources makes this an unsustainable solution. Besides that, along with the advantages of the spectrum-sharing strategy, the interference that adversely affects the network's transmission link is a major concern, which we would like to solve in this thesis.

We have thus developed a channel allocation algorithm using a combination of the greedy coloring algorithm and the node removal, related to the node criticality technique. In addition, we have also demonstrated the modeling wireless network graph, along with our system models, including node distribution using PPP, node importance using eigenvector centrality, modeling interference, and the price elasticity of demand. The greedy algorithm is implemented first to determine the number of colors used, then the least important nodes are removed from the network to decrease the number of colors. The

price elasticity of demand is used later to evaluate the simulation results and the gain of our proposed solution.

CHAPTER 4: DEEP LEARNING-BASED CHANNEL ALLOCATION

4.1 INTRODUCTION

In Chapter 3, we have proposed the technical solution to suppress the channel interference by determining the minimum number of colors used in the network, then reducing this number using node removal. In this chapter, we continue to propose another approach to efficiently allocate channels in IoT networks using deep reinforcement learning.

Although the greedy coloring algorithm is fast to implement in many cases with few available colors, this algorithm is not an optimal solution for channel allocation in large IoT networks (i.e., these networks require many available colors assigned to all nodes, and IoT devices may be mobile). As described earlier, in graph theory, determining exactly the chromatic number is not easy due to the dynamic topology of wireless networks. Therefore, it is more possible to specify the chromatic number's upper and lower bounds. The upper bound of the chromatic number is the maximum degree d_{max} of the network graph based on Brooks's theorem (as stated in Equation 3.11). On the other hand, the lower bound is not necessarily specified because our goal is to determine the minimum number of colors allocated to the nodes. Besides that, our training models require the upper bound (i.e., the maximum degree) as the input data. Therefore, we propose a deep reinforcement learning-based channel allocation as an alternative solution, which can work well in a large IoT network containing thousands of nodes.

4.2 Background of Deep Reinforcement Learning

One of the most active areas of artificial intelligence research is reinforcement learning. In computer science, it has been considered a useful tool to solve problems regarding wireless communications. Therefore, we choose to develop and implement the deep reinforcement learning algorithm for channel allocation. In contrast to other ML techniques (e.g., Naïve Bayes, Support Vector Machine, and Nearest Neighbor), reinforcement learning involves learning from actions mapping onto the environment. The behavior of an agent, which interacts with the environment, obtains the reward. In other words, the actions directly affect the performance of the learning machine. The goal of reinforcement learning is to maximize the reward value obtained. From the perspective of reinforcement learning, at least five essential components need to be defined. These are policy, reward, state, agent, and environment. Designing reinforcement learning algorithms must be dependent on these elements. Figure 5.1 demonstrates the deep reinforcement learning process. Accordingly, an agent is an object that can learn to behave from the environment. The agent will continuously take the action at each state. Each action of the agent to the environment will fetch the reward. Throughout the learning process, the model tries to achieve the maximum expected reward so that the agent can behave optimally at each state [45].

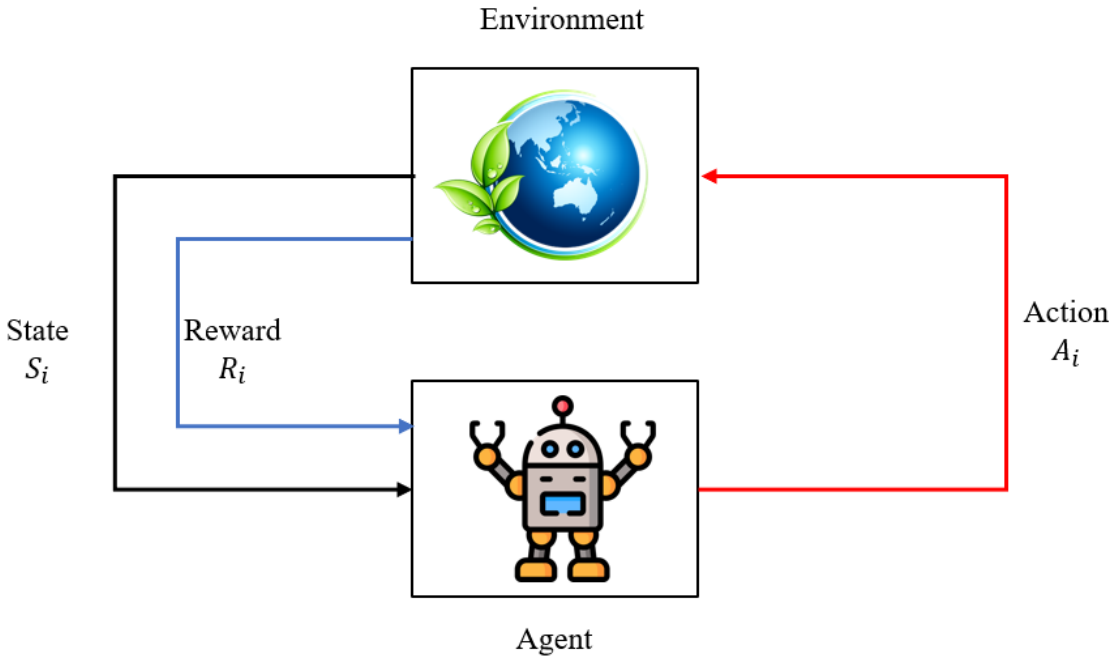

Figure 4.1: Deep Reinforcement Learning Process

Deep reinforcement learning, in which machines can generalize the representation of data by autonomously learning from their features, is a combination of deep learning and reinforcement learning. With the increase in computational powers these days, deep reinforcement learning can speed up the process of inputting data in multiple layers.

Several deep learning architectures are designed for various purposes of applications. For example, to deal with supervised learning problems, convolutional networks, and recurrent neutral networks can be applied to train the model. In contrast, self-organizing maps and autoencoders can be used to solve unsupervised learning problems. Regardless of the architecture of deep learning, three essential components are

required to design a deep learning architecture. These are an input layer, a set of hidden layers, and an output layer. The sampled diagram of deep learning network architecture is illustrated in Figure 4.2 below.

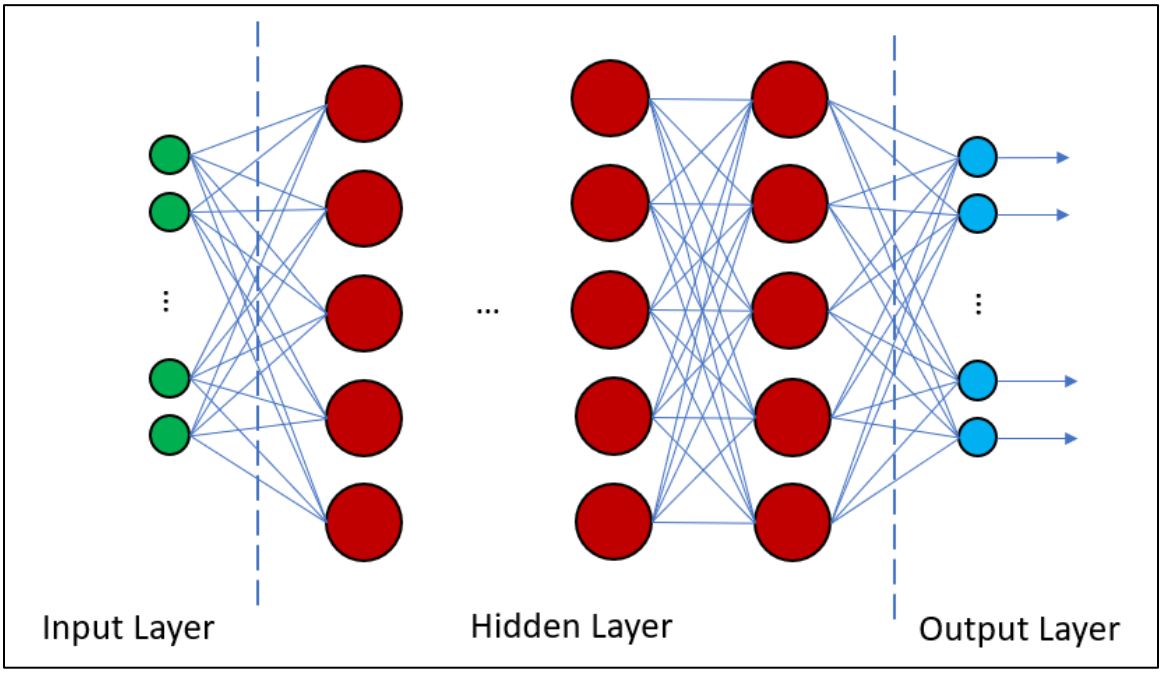

Figure 4.2: Deep Learning Network Architecture

Figure 4.2 is a generalized deep-learning network. Accordingly, all layers of the network can consist of multiple neurons. To summarize, the input layer is the first layer where initial data is taken into the deep learning network and passed to the next layer (i.e., the hidden layer). The second part is the hidden layer, where all the computations of the model are

operated. Lastly, the output layer is where the results of given input data are produced. The design of input and output layers is generally straightforward. However, the hidden layer is much more complicated than the two other layers because it contains multiple neurons in multiple layers of the hidden layer. We need to seriously consider the complexity of the hidden layer because the performance of the model mostly depends on the complex structure of the hidden layer. However, if the hidden layer is too complicated, the time to train the network can be extremely high [44].

4.3 SYSTEM MODEL

In this thesis, we choose deep deterministic policy gradient (DDPG), which is one of the types of deep reinforcement learning, to determine the minimum number of colors used to label a network graph [45]. The principle of the DDPG is based on the combination of the actor-critic method and the deep Q-learning architecture. The DDPG algorithm uses four neutral networks (actor-network, target actor network, critic network, and target critic network) to search for an optimal policy so that the expected result can be maximized. In each training step, the actor-network is employed to generate the action for the environment. Then, the critic network is utilized to evaluate the performance of the action network by calculating the Q-value. In addition, the target action network and the target critic network are used to calculate the loss function so that the stability of DDPG can be improved. The architecture of the DDPG algorithm is illustrated in Figure 4.3 below:

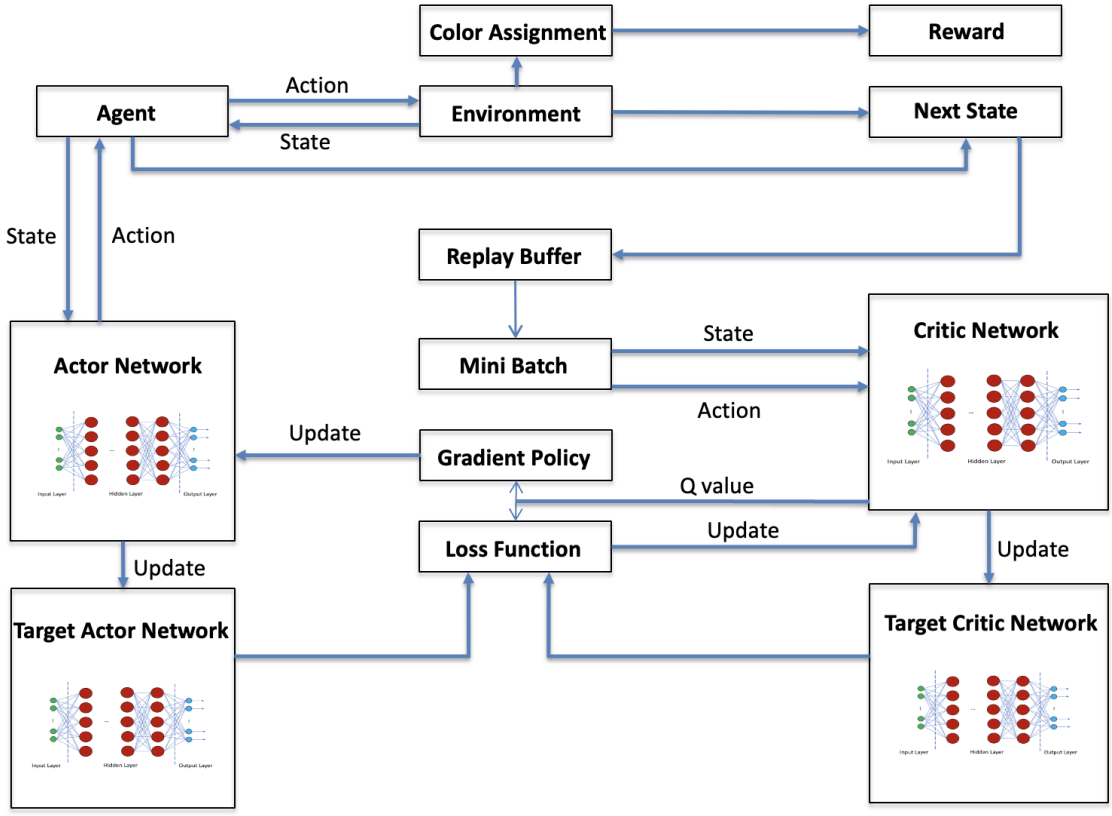

Figure 4.3: The Architecture of the DDPG Algorithm

As we have described in the previous section, a fully connected neutral network is formed by three layers as follows: the input layer, the hidden layer, and the output layer. Indeed, all four neural networks in Figure 4.3 must be followed have three layers. Each layer consists of neurons, which are connected by weighted links. The weight of each link is randomly assigned in the first state, then it is adjusted later based on the environment.

Now, the state, the action, the reward, and the next state of DDPQ need to be defined below. Assuming that the network graph G has N nodes, we have the adjacency matrix $A(G)$ and the location of the nodes following PPP. The system model is similar to the graph coloring-based channel allocation described in the previous chapter.

1. State: There are three different components used to represent the current status of the given environment after each action is done. These are the location of the node, the adjacent links, and the associated color of the node where the agent is staying.

2. Action: The agent will release the color results of each node in the network graph G. For each node, the color will be chosen from the list of available colors $C = \{1, 2, 3, \ldots d_{max}\}$, where d_{max} is the maximum degree, which is defined as the upper bound of the chromatic number in graph coloring.

3. Reward: Depending on the action at each state, the reward is the summation of the total number of colors used and invalid/disconnected links. The invalid/disconnected links between the nodes are identified when two adjacent nodes have the same color. The expected reward needs to be maximized, but we aim to minimize the total number of colors and the invalid/disconnected links between the nodes. Therefore, in our situation, the final results are multiplied by (-1) to maximize the reward. The calculation of the reward can be expressed as follows:

$$R = M + \frac{I}{\gamma} \tag{4.1}$$

where R is defined as the reward, M is the number of colors required, I is the total invalid/disconnected links of each two nodes having the same colors, and γ is the weight. Here, the weight (γ) which is the parameter of a neural network is used to transform input data within the hidden layer. The inverse of the weight (γ) is indicated in Equation 4.2 as follows:

$$\gamma = \frac{1}{F * \frac{E}{d_{max}}} \tag{4.2}$$

where F is a factor, E represents the total number of communication links, and d_{max} is the maximum degree of the network graph.

4. Next State: Similar to the definition of the state above, the next state represents the next node in terms of the location, the adjacent links, and the used colors of previous nodes. All of these components are updated by using action.

4.4 SUMMARY

In order to suppress channel interference in IoT networks, an effective technique for channel allocation needs to be implemented. The goal is to minimize the number of colors labeled in the network graph. Although the greedy coloring algorithm can determine

the chromatic number (i.e., the minimum number of colors), this algorithm is sub-optimal because it may not work well in a large network like an IoT network. Therefore, we propose a more powerful technique to specify the total number of colors. This is a DDPG algorithm based on the graph coloring theory.

In various types of ML technology, deep reinforcement learning is considered the best possible choice to develop and implement for our wireless networks' problem. There are two important things that all deep reinforcement learning needs to take into account. The first important thing is to define state, action, reward, policy, and agent of learning models. With each particular case, these elements need to be correctly identified. The second thing is to design a good neutral network, including an input layer, hidden layer, and output layer, to train the data so that the maximum reward can be achieved as much as possible.

In addition, we follow the requirements of a typical DDPG to describe the state, the action, and the reward corresponding to our system model. This will be used to run the simulation in the next chapter.

CHAPTER 5: SIMULATION

5.1 INTRODUCTION

In the previous chapters, we have introduced the theoretical concepts and our system models, including the wireless network graph and deep reinforcement learning. We now run our simulations for channel allocation using two proposed solutions: node elimination and DDPG. To run the simulation, we use two different programming languages (MATLAB and Python) corresponding to the following software applications: MATLAB and PyCharm. Channel allocation using node elimination-based graph coloring is simulated in MATLAB because this program provides the package containing all materials of mathematical graph theory. Also, the network graphs can be easily plotted using MATLAB. On the other hand, deep reinforcement learning-based channel allocation is simulated in Python language because Python is the best option for ML and Artificial Intelligence projects. Python provides the access to great libraries and frameworks for ML, and many ML projects are available in Python due to the immense community contributing to Python.

5.2 CHANNEL ALLOCATION USING NODE ELIMINATION-BASED GRAPH COLORING

We now focus on the first proposed solution of channel allocation based on graph coloring. We first introduce the simulation model, then obtain the results from the simulation to have a critical discussion in the later section.

5.2.1 Simulation Model

For all simulations of our network model as a random geometric graph, we assume that the total number of nodes N is distributed in a $100{,}000\ m^2$ area A. The number of nodes in a given area A follows the Point Poisson Process with the node density λ. All nodes of our network model have a fixed transmit power P_t of $10\ mW$ (or $10\ dBm$) and a frequency f of $2.0\ GHz$. Based on the fixed transmit power P_t and the frequency f, the maximum operational distance d_t is $20\ m$ that follows a free-space path loss model. The adjacency matrix of the network graph can be easily calculated when the maximum operation distance matrix is defined. Once the adjacency matrix is identified, the degree matrix and node criticality of the network graph can be derived.

5.2.2 Simulation Results

We consider that the network consists of 83 IoT devices (or 83 nodes), with other parameters (including given area, transmit power, frequency, etc.) defined above. Figure 5.1 indicates the locations of the nodes and communication links between nodes.

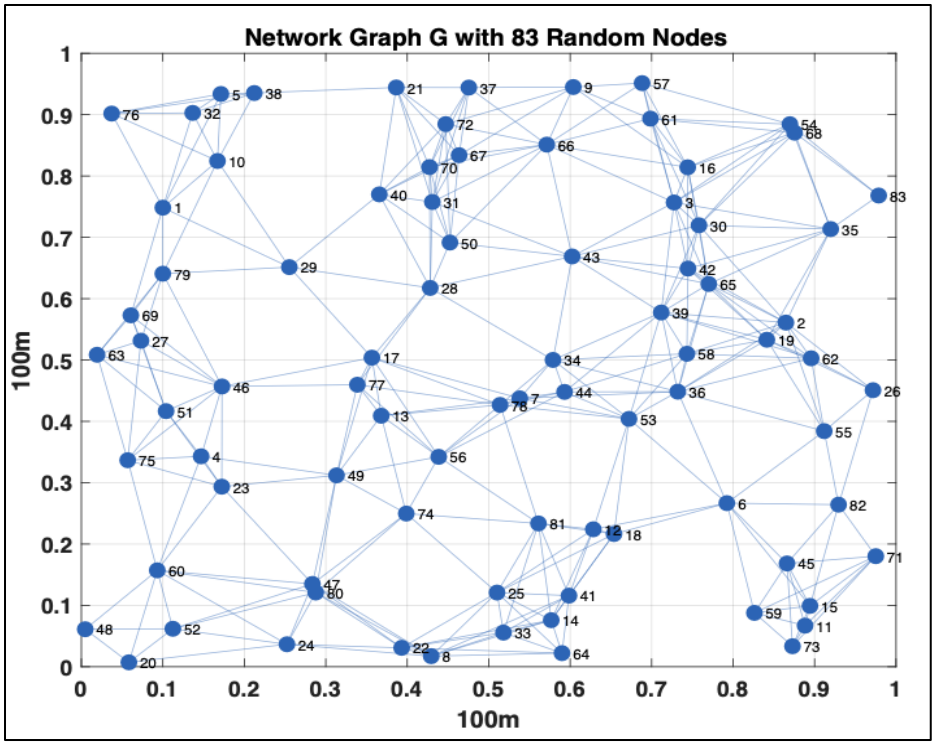

Figure 5.1: Network Graph G with 83 Random Nodes

With node distribution and connections between nodes indication, it does not provide much information on network graph G. Hence, we need more insight into network graph G to evaluate the performance, in order words, network connectivity. The eigenvector centrality is measured to provide more insight regarding the importance of each node based on their neighboring nodes and communication links. Results for an 83-node eigenvector centrality of network graph G simulation are provided in Figure 5.2. According to the eigenvector centrality, we can identify that the most important node is node 39 (indicated as red), while the least important node is node 48 (indicated as blue). It is clear that the neighboring nodes

of node 39 also have high eigenvector centrality scores. Therefore, we conclude that the important nodes are always located in a cluster or a group. On the other hand, the least important nodes are located in the boundary of the network, and they do not have many connected links with their neighboring nodes. Next, the greedy coloring algorithm was implemented in this network graph to calculate the chromatic number, in other words, the number of channels needed to efficiently allocate to each node. The result of the chromatic number obtained from the simulation is 8. Equivalently, the channel allocation requires at least 8 different colors so that no adjacent nodes have the same colors.

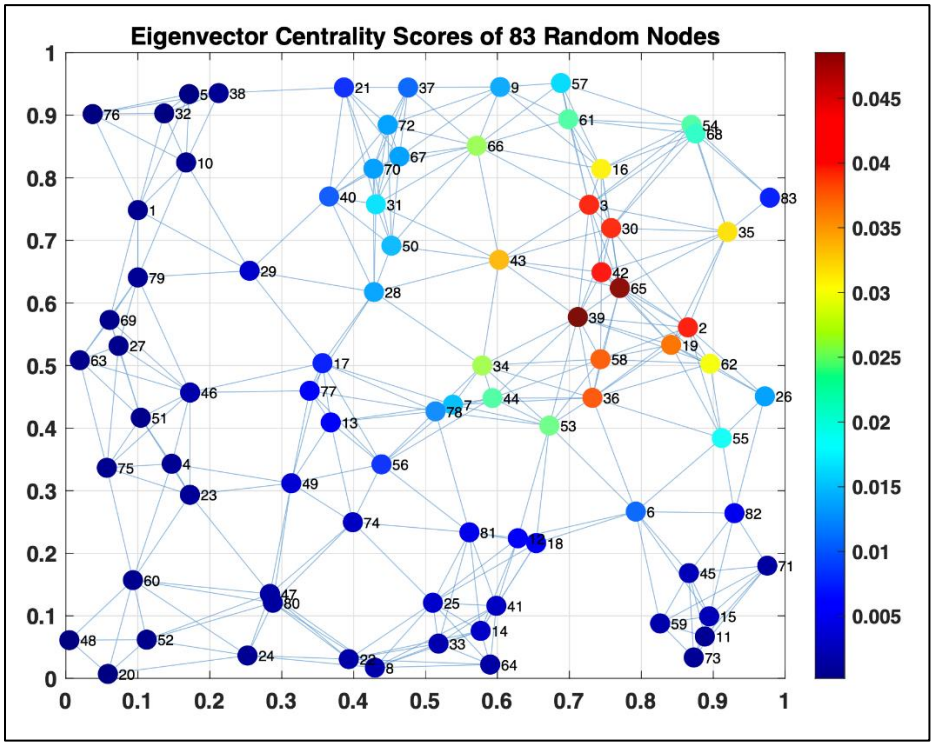

Figure 5.2: Eigenvector Centrality Scores of 83 Random Nodes

The following figure demonstrates the network graph G applying the greedy coloring algorithm to assign the color to the nodes. It is clear that none of the adjacent nodes have the same colors.

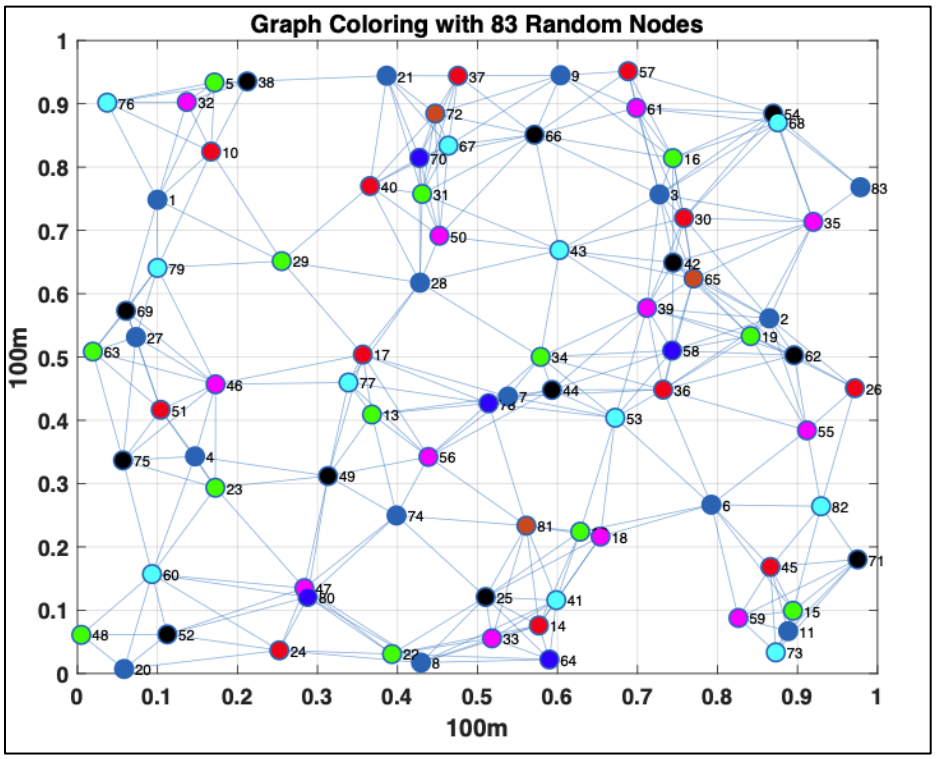

Figure 5.3: Graph Coloring with 83 Random Nodes

In addition to the graph coloring, Table 5.1 shows the node numbers corresponding to the colors. The nodes of the network graph G are denoted by a list of $\{1, 2, 3, \ldots 83\}$ and the

colors used to label each node is denoted by a list of $\{1, 2, 3, \ldots 8\}$. Therefore, we can know exactly which color is associated with which node of the network.

Table 5.1: Greedy Graph Coloring of an 83-Node Wireless Network G

Node Number	Assigned Color	Node Number	Assigned Color	Node Number	Assigned Color	Node Number	Assigned Color
1	1	22	2	43	6	64	7
2	1	23	2	44	5	65	8
3	1	24	3	45	3	66	5
4	1	25	5	46	4	67	6
5	2	26	3	47	4	68	6
6	1	27	1	48	2	69	5
7	1	28	1	49	5	70	7
8	1	29	2	50	4	71	5
9	1	30	3	51	3	72	8
10	3	31	2	52	5	73	6
11	1	32	4	53	6	74	1
12	2	33	4	54	5	75	5
13	2	34	2	55	4	76	6
14	3	35	4	56	4	77	6
15	2	36	3	57	3	78	7
16	2	37	3	58	7	79	6
17	3	38	5	59	4	80	7
18	4	39	4	60	6	81	8
19	2	40	3	61	4	82	6
20	1	41	6	62	5	83	1
21	1	42	5	63	2		

Next, we start to simulate our wireless network models with the different values of node density. Other parameters of the system models follow the assumption above. To analyze the networks, we take into account the following elements: node density, the total number of nodes, the average number of nodes connected for each node, the total number of edges, total transmit power in W, the maximum number of colors, and the number of colors used when applying the greedy coloring approach. As the node density increases, the chromatic number increases as well (see Figure 5.4).

Table 5.2: Simulation Results of Node Density Varying from 50 to 250

Node Density	50	100	150	200	250
Total Nodes	54	83	131	228	248
Average Nodes Connected to Each Node	5.81	7.81	12.9	22.83	25.07
Total Communication Links	157	324	845	2603	3109
Total Transmit Power (W)	15.7	32.4	84.5	260.3	310.9
Maximum Needed Colors	11	13	25	36	39
Number of Colors Used to Label	7	8	13	19	21

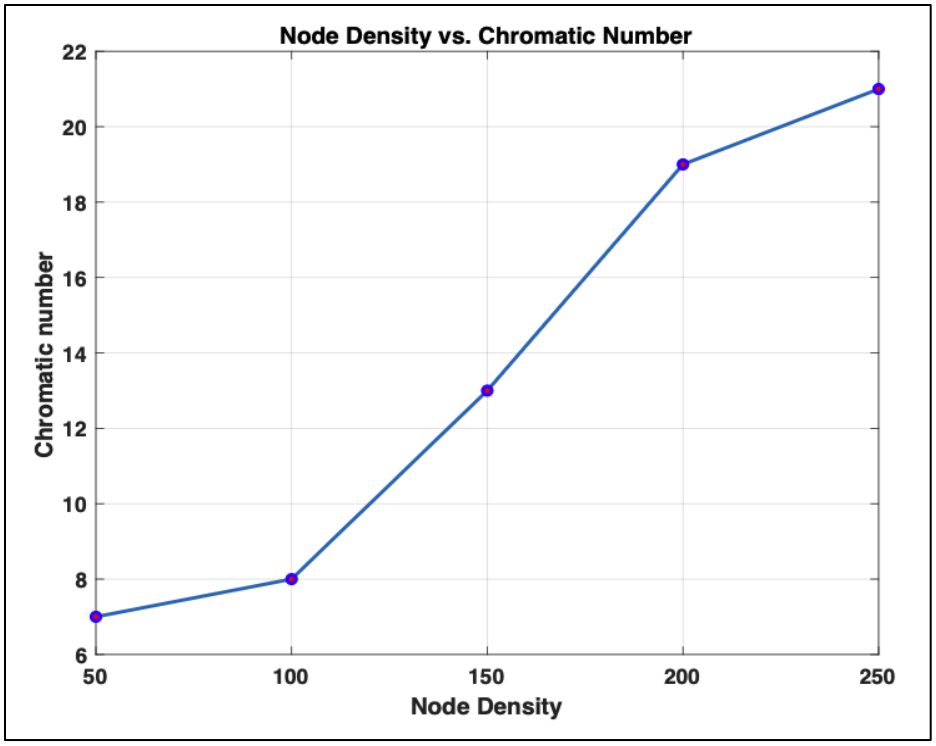

Figure 5.4: Node Density vs. Chromatic Number using Greedy Coloring Algorithm

Again, the goal is to minimize the number of colors used to avoid channel interference and reduce channel usage. According to our simulations, we observe that eliminating the least important nodes does not significantly affect a small network, but it does in a large network (e.g., a wireless network consists of more than 200 nodes). Therefore, we run our simulation with node density λ of 250. Table 5.3 demonstrates the simulation results with the following variables: percentage node removal, number of nodes, the average number of nodes connected to each node, number of connected links, and

number of colors used. In addition, Figure 5.5 illustrates the relationship between the percentage of node removal and the number of colors using the greedy coloring algorithm.

Table 5.3: Simulation Results with Node Density λ of 250

Percentage Node of Removal (%)	Number of Nodes	Average Number of Nodes Connected	Number of Connected Links	Number of Colors Used
0	248	25.07	3109	21
10	224	22.56	2527	19
20	199	19.85	1975	17
30	174	17.44	1517	15
40	149	15.05	1136	14
50	124	12.15	753	12

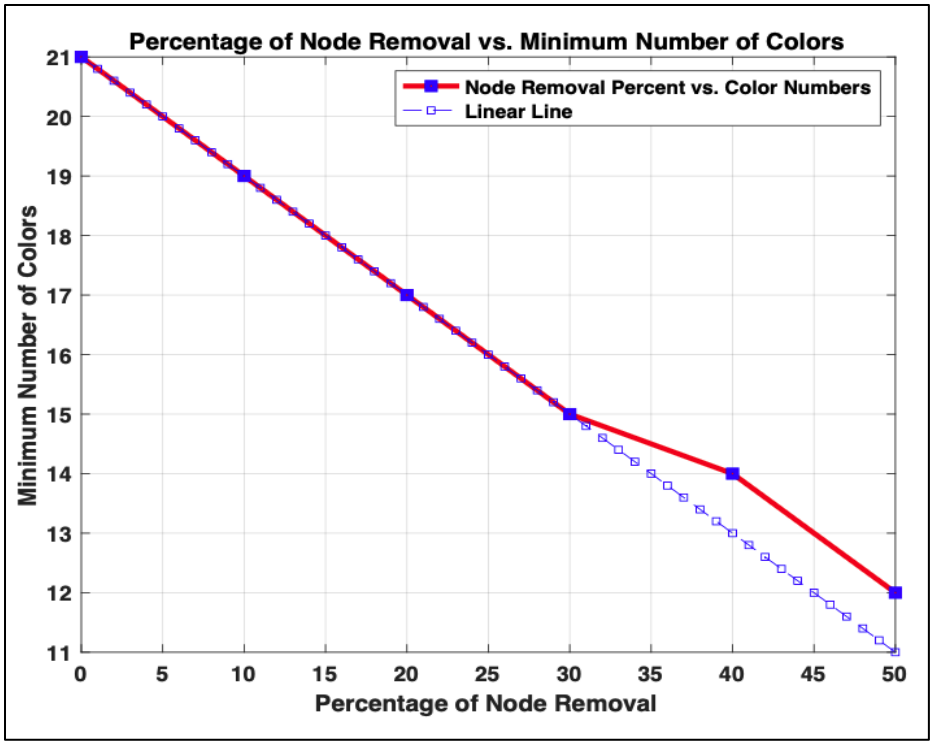

Figure 5.5: Number of Colors vs. Percentage of Node Removal

According to the simulation, the number of colors linearly decreases when the percentage of node removal increases. Also, the percentage of node removal is inversely proportional to the average number of nodes connected to each node and the number of connected links. It is difficult to conclude that this approach is good if we just evaluate based on the collected data in Table 5.3 and Figure 5.5. Therefore, we need to calculate the price elasticity coefficient using the elasticity theory. From the data collected in Table 5.3, the price elasticity coefficient is 0.964 when the percentage of node removal changes from 0 percent to 20 percent. This demonstrates that the value of the elasticity coefficient is

approximately one. It means that the overall profitability does not always decrease or increase if the less important nodes are eliminated to suppress the number of colors. Therefore, we conclude that this proposed approach can be applied to wireless networks, but it depends on each scenario and the certain time when this approach is implemented. In other words, the performance can achieve high revenue if the proposed approach is applied to a suitable wireless network. The factors to define this network can be listed as follows: the location of the nodes, network topology, number of nodes, and maximum operational distance. However, the gain of the network (as defined in Section 3.4) is 0.2 when the number of colors reduces from 21 distinct colors to 17 distinct colors. That means our proposed solution can bring a certain benefit to suppress the interference impacting each node of the network.

5.3 Channel Allocation Using Deep Reinforcement Learning

In the previous section, we reduced the chromatic number determined by the greedy coloring algorithm by eliminating the least important nodes. We now run the simulations to determine the chromatic number by deep reinforcement learning based on the principled ideas of graph coloring theory.

5.3.1 Simulation Model

For wireless network graph generation, we also use the same assumption as Section 5.2.1. In particular, we assume that there are N nodes distributed in a given area of $100{,}000\ m^2$ following PPP. Also, the maximum operational distance d_t is $20\ m$, which is

calculated based on the free-space path loss In the ML part, we define two parameters to evaluate our simulation results. These are the number of colors used to allocate the nodes (as defined in Chapter 4) and the invalid/disconnected links.

5.3.2 Simulation Results

The performance of the DDPG model is highly dependent on choosing the value of the weight to train the model. Therefore, we want to demonstrate the relationship between the value of the weight and two elements of the reward (i.e., the number of colors required and total invalid/disconnected links. In Figure 5.6, we can see that the DDGP model can achieve the maximum reward (a low number of colors required and a low number of disconnected links) when the weights are in the range of 11 – 17 in this case.

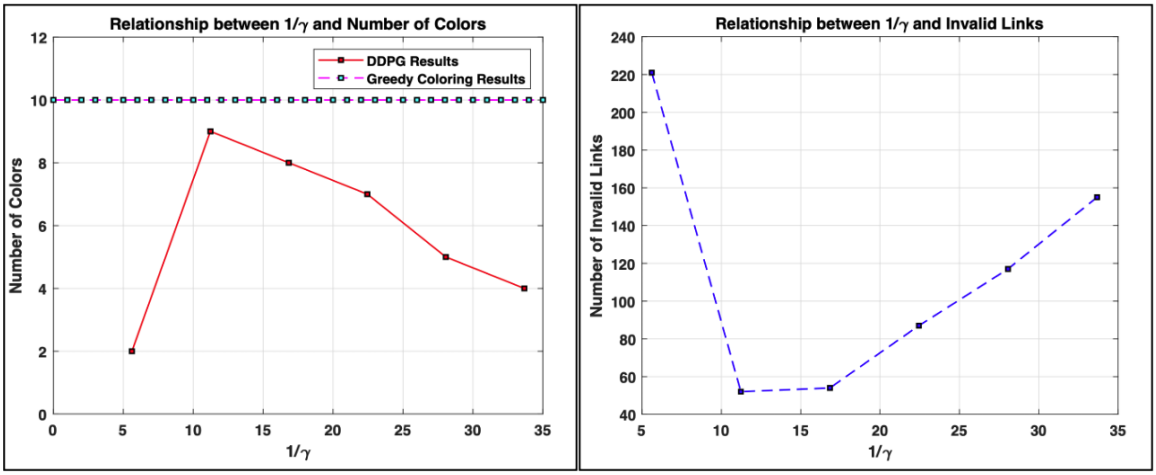

Figure 5.6: $\frac{1}{\gamma}$ vs. Number Of Colors (left) and vs. $\frac{1}{\gamma}$ Number of Invalid Links (right)

Results of different simulations using the DDPG algorithm are indicated in Table 5.4 and Figure 5.7 below. To evaluate the performance of the DDPG algorithm, we compare it with a baseline algorithm (i.e., greedy coloring algorithm).

Table 5.4: Simulation Results to Determine $\chi(G)$ using the DDPG Algorithm

Node Density	$\chi(G)$ Greedy Coloring Algorithm	$\chi(G)$ DDPQ Algorithm	Disconnected Links	Total Number of Links	Disconnected Links Percent (%)
50	9	7	24	165	14.55
75	10	7	45	328	13.72
100	10	8	54	449	12.03
125	15	12	130	1210	10.74

We observe that the greedy coloring algorithm uses a possible number of colors to allocate channels, and it can guarantee that no adjacent nodes have the same colors. Therefore, the greedy algorithm does not have any disconnected links. Meanwhile, even though the DDPQ algorithm does not guarantee that all adjacent nodes have distinct colors, allocating the channels using the DDPG can optimize the effectiveness with the minimum number of colors. However, several links between two adjacent nodes having the same colors will be vulnerable that causing them to be disconnected. we only take into account the outcomes of invalid/disconnected links that fall within an acceptable range from 10 percent to 15 percent of the total number of communication links in the wireless network. According to Figure 5.7, to allocate the channels in wireless networks with node density varying from 50 to 125, the DDPG algorithm always requires a smaller number of colors than the greedy algorithm.

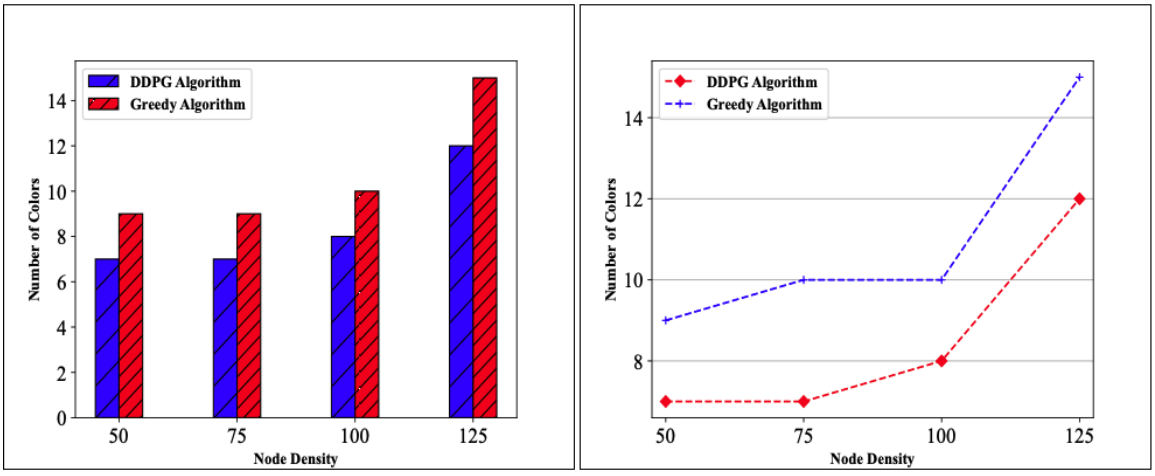

Figure 5.7: Node Density λ vs. Number Of Color $\chi(G)$

In addition, the plots of the wireless networks applied DDPG-based channel allocation are demonstrated in Figures 5.8 and 5.9.

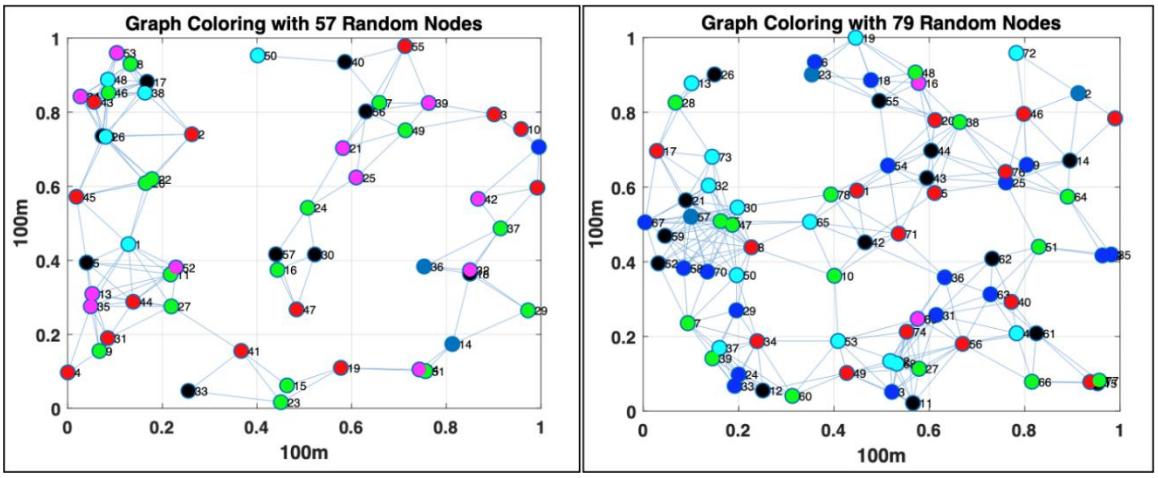

Figure 5.8: Graph Coloring using DDPG with λ of 50 (left) and 75 (right)

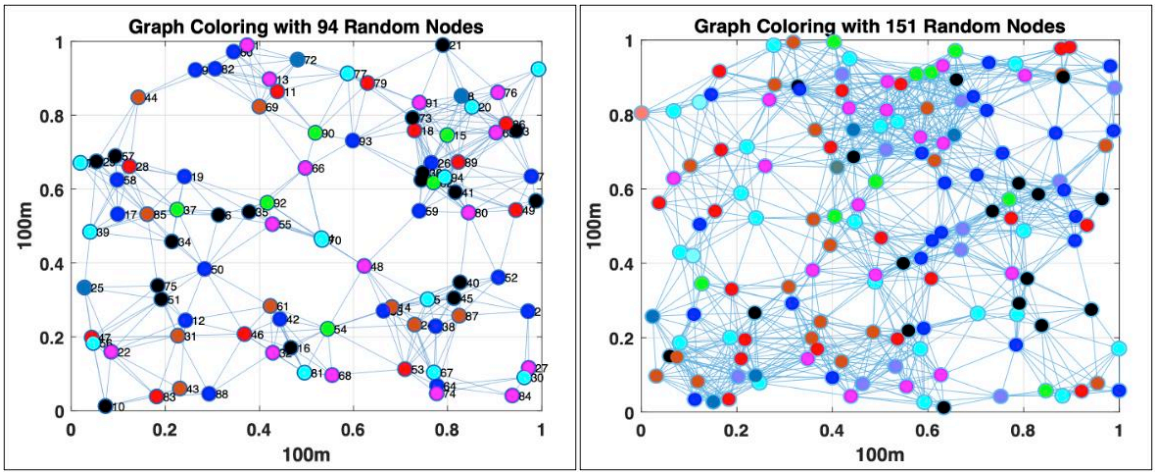

Figure 5.9: Graph Coloring using DDPG with λ of 100 (left) and 125 (right)

5.4 CONCLUSION

In terms of the wireless network, not all nodes are equally important. It depends on not only the position of the node but also the involvement of its surrounding nodes. Therefore, we also present the method for measuring the node criticality using eigenvector centrality. We can determine which node is the most significant in the network based on the effects of its importance. With the less significant nodes, we can consider removing these nodes to lessen the interference.

We calculated the price elasticity of demand to determine that the overall profitability of our proposed approach depends on the topology of wireless networks. In some particular cases, the benefits can be achieved high when the least important nodes are eliminated from the network graph. In addition, the gain we obtain is the number of colors reduced compared to the results of the greedy coloring algorithm. The gain can increase if the percentage of node removal increases as well.

We also propose deep reinforcement learning-based channel allocation to determine the number of colors used to allocate the nodes. The results show that the number of colors using the DDPG algorithm is always smaller than the greedy coloring algorithm. Therefore, we conclude that deep reinforcement learning-based channel allocation outperforms graph coloring-based channel allocation. However, the training time for the learning model is a challenge. In the real world, since wireless networks are dynamic, as well as IoT devices have the ability to move around, the updated locations of the IoT

devices and the optimal routing path need to propose in the shortest time (e.g., milliseconds).

CHAPTER 6: SUMMARY AND FUTURE WORK

6.1 SUMMARY

IoT networks are heterogenous and dynamic because they are flexible and dependent on IoT applications and these applications' ability to communicate with other components. In this thesis, we first introduced other research works related to route optimization, channel allocation, and graph coloring. Next, we discussed the concept of IoT networks in terms of applications, operation, and constraints (including resiliency, latency, and efficiency). We also introduced an interference model with IoT network access to determine the negative effects of channel interference on the performance of network connectivity and quality of service. We then implemented the greedy graph coloring algorithm based on the concept of graph theory to allocate the channels in the wireless network. From the perspective of our system models, we developed and defined the following concepts: node distribution using Point Poisson Process, free-space path loss formula, Signal-to-Interference-plus-Noise ratio (considering external and internal interference), and node centrality. All components of the system model above are utilized to do analytical simulations.

We also proposed two different approaches for improving the resilience of connections. The first approach is reducing the number of colors labeled by eliminating several less important nodes. By observing the results of the simulations applied to this approach, we conclude that it is a trade-off between link connectivity (i.e., link density) and channel interference. The second approach we propose is determining the number of

channels used by using deep reinforcement learning (i.e., DDPG) based on the principled idea of graph coloring. According to the simulation results, it is evident that the DDPG algorithm outperforms the greedy coloring algorithm in effectively allocating the channels. However, to apply the DDPG algorithm for channel allocation in the real world, we also need to minimize the training time of the learning model to obtain the final results as soon as possible (in milliseconds).

6.2 Future Work

In this thesis, we discussed the solution to avoid channel interference by eliminating several nodes, which have the lowest score for node centrality. This work can be extended by deeply analyzing to determine which nodes should be removed so that the number of colors used can be minimum, but the links between nodes can still have a high level of connectivity. On the other hand, we used node centrality to measure node importance. Future work can focus on developing a measurement to provide a better performance of evaluation for node criticality.

Another aspect, which can be extended for future work, is to develop a better performance of DDPG to determine the number of colors labeled for channel allocation. This could reduce the computational cost to obtain the outcomes, as well as decrease the time that DDPG needs to run the simulation for reaching the highest reward.

APPENDIX A: MATLAB

```
%{
Written by Hieu Nguyen
Date: Oct 28, 2022
Description: This program demonstrates the wireless network with the
following components:
   1) Node Distribution using Point Poisson Process (Node Location)
   2) Maximum Operational Distance Matrix
   3) Adjacency Matrix
   4) Degree of Each Node in Graph G
   5) Average Number of Nodes Connected to a Given Node
   6) Total Number of Communication Links
   7) Total Number of Transmit Power/ Average Transmit Power
   8) Node Centrality
   9) Greedy Coloring Algorithm (Number of Colors)
   10) Plot Figures
%}

clc;
clear;
close all;

%% 1) Node Distribution using Point Poisson Process (Node Location)

%Simulation window parameters
xMin=0;xMax=1;
yMin=0;yMax=1;
xDelta=xMax-xMin;yDelta=yMax-yMin; %rectangle dimensions
areaTotal=xDelta*yDelta;

%Point process parameters
lambda=120; %intensity (ie mean density) of the Poisson process
fprintf('Node density: %d \n', lambda);

%Simulate Poisson point process
numberOfNodes=poissrnd(areaTotal*lambda);%Poisson number of points
xx=xDelta*(rand(numberOfNodes,1))+xMin;%x coordinates of Poisson points
yy=yDelta*(rand(numberOfNodes,1))+yMin;%y coordinates of Poisson points

%Node Location
nodeLocation = zeros(numberOfNodes,2);
```

```
for i = 1:numberOfNodes
      nodeLocation(i,:) = [xx(i) yy(i)];
end

fprintf('The total number of nodes: %d nodes\n', numberOfNodes);

%% 2) Maximum Operational Distance Matrix

%Calculate distance between each node
distanceMatrix = zeros(numberOfNodes,numberOfNodes);
for i = 1:numberOfNodes
   for j = 1:numberOfNodes
      distanceMatrix(i,j) = norm(nodeLocation(i,:) - nodeLocation(j,:));
   end
end

%Maximum Opeartional Distance Matrix
maxOperationDistance = zeros(numberOfNodes,numberOfNodes);
for i = 1:numberOfNodes
   for j = 1:numberOfNodes
      if distanceMatrix(i,j) > 0.20
         maxOperationDistance(i,j) = 0;
      else
         maxOperationDistance(i,j) = distanceMatrix(i,j);
      end
   end
end

%% 3) Adjacency Matrix

%Adjacency Matrix of Graph G
adjacencyMatrix = maxOperationDistance;

for i = 1:numberOfNodes
   for j = 1:numberOfNodes
      if adjacencyMatrix(i,j) > 0
         adjacencyMatrix(i,j) = 1;
      end
      adjacencyMatrix(j,i) = adjacencyMatrix(i,j); %copy the values of upper matrix to
lower matrix
   end
end

%Network Graph G
```

```
G = graph(adjacencyMatrix);

%% 4) Degree of Each Node in Graph G

degreeVector = degree(G);

%The maximum value of degree (dmax - upper bound)
maxDegree = max(degreeVector);
fprintf('The maximum value of degree: %d \n', maxDegree);
%Check whether or not the graph is fully connected
minDegree = min(degreeVector);
if min(degreeVector) == 0
   fprintf('Graph G is disconnected\n');
else
   fprintf('Graph G is fully connected\n');
end

%% 5) Average Number of Nodes Connected to a Given Node

nodeConnected = sum(adjacencyMatrix(:,:)==1);
nodeConnected = nodeConnected';
expectedNodes = mean(nodeConnected);
fprintf('The expected number of nodes connected for each node: %.2f nodes\n',
expectedNodes);

%% 6) Total Number of Communication Links

numberOfEdges = numedges(G);
fprintf('The total number of edges: %d links\n', numberOfEdges);

%% 7) Total Number of Transmit Power/ Average Transmit Power

%Assume that the transmit power of each node (Pt) = 10 mW (10 dBm)
totalPower = 10*numberOfEdges*10E-3;
fprintf('The total transmit power: %.2f W\n', totalPower);

%Average Transmit Power
averagePower = totalPower/numberOfNodes;
fprintf('The average transmit power of each node: %.2f W\n', averagePower);

%% 8) Node Centrality

nodeCentrality = centrality(G,"eigenvector");
eigenvalue = eig(adjacencyMatrix);
```

```
fprintf('The total number of eigenvalues: %d \n', length(eigenvalue));
maxEigenvalue = max(eigenvalue);
fprintf('The largest eigenvalue: %.2f \n', maxEigenvalue);

%% 9) Greedy Coloring Algorithm (Number of Colors)

%n = length(xx);
color = zeros(numberOfNodes,1); % Initialize array to hold color numbering
numcolors = 1; % Number of colors needed (so far)
for k = 1:numberOfNodes
   idx = neighbors(G,k); % Get neighbors of the kth node
   idx = idx(idx < k);   % But just those that have an assigned color
   neighborcolors = unique(color(idx)); % Get colors used by neighbors
   % Assign the smallest color value not used by the neighboring nodes
   thiscolor = min(setdiff(1:numcolors,neighborcolors));
   % If there isn't one, add another color to the map
   if isempty(thiscolor)
      numcolors = numcolors + 1;
      thiscolor = numcolors;
   end
   color(k) = thiscolor;
end
disp([num2str(numcolors),' colors needed'])

%% 10) Plot Figures

%Plotting node location map
figure(1);
sz = 100;
p1 = scatter(xx,yy,sz,'filled','o');
p1.MarkerFaceColor = [0.6350 0.0780 0.1804];
p1.MarkerEdgeColor = 'k';
xlabel('100m');ylabel('100m');title('Node Distribution using Point Poisson Process');
xlim([xMin xMax]); ylim([yMin yMax]);
set(gca,'fontsize',12,'FontWeight','bold');
grid on;

%Plotting network graph G
figure(2);
p2 = plot(G, 'XData', nodeLocation(:,1), 'YData', nodeLocation(:,2),'MarkerSize',8);
xlabel('100m');ylabel('100m'); title(['Network Graph G with ',num2str(numberOfNodes),'
Random Nodes']);
xlim([xMin xMax]); ylim([yMin yMax]);
set(gca,'fontsize',12,'FontWeight','bold');
```

```
grid on;
hold on;

%Plotting eigenvector centrality scores
figure(3);
p3 = plot(G, 'XData', nodeLocation(:,1), 'YData', nodeLocation(:,2),'MarkerSize',10);
p3.NodeCData = nodeCentrality;
colormap jet;
colorbar
xlim([xMin xMax]); ylim([yMin yMax]); title(['Eigenvector Centrality Scores of
',num2str(numberOfNodes),' Random Nodes']);
set(gca,'fontsize',12,'FontWeight','bold');
grid on;

%Plotting the graph coloring

figure(6);
p6 = plot(G, 'XData', nodeLocation(:,1), 'YData', nodeLocation(:,2),'MarkerSize',10);
xlabel('100m');ylabel('100m'); title(['Graph Coloring with ',num2str(numberOfNodes),'
Random Nodes']);
xlim([xMin xMax]); ylim([yMin yMax]);
set(gca,'fontsize',12,'FontWeight','bold');
grid on;
hold on;

nodeColor = [nodeLocation color];
for i = 1:numberOfNodes
   if nodeColor(i,3) == 1
      s1 = scatter(nodeColor(i,1),nodeColor(i,2), 80,'filled');
      s1.MarkerFaceColor = [0 0.4470 0.7410];
   elseif nodeColor(i,3) == 2
      s2 = scatter(nodeColor(i,1),nodeColor(i,2), 80,'filled');
      s2.MarkerFaceColor = [0 1 0];
   elseif nodeColor(i,3) == 3
      s3 = scatter(nodeColor(i,1),nodeColor(i,2), 80,'filled');
      s3.MarkerFaceColor = [1 0 0];
   elseif nodeColor(i,3) == 4
      s4 = scatter(nodeColor(i,1),nodeColor(i,2), 80,'filled');
      s4.MarkerFaceColor = [1 0 1];
   end
end
```

APPENDIX B: PYTHON

B.1 NETWORK GENERATION

```
# Import packages
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import dill
import class_set as CL
matplotlib.rcParams['pdf.fonttype'] = 42
matplotlib.rcParams['ps.fonttype'] = 42
plt.close('all')
# Simulation window parameters
xMin = 0
xMax = 1
yMin = 0
yMax = 1
xDelta = xMax - xMin
yDelta = yMax - yMin  # rectangle dimensions
areaTotal = xDelta * yDelta
"(1) Node Distribution using Point Poisson Process"
# Point process parameters
lambda0 = 125 # intensity (ie mean density) of the Poisson process
# Simulate Poisson point process
numberOfNodes = np.random.poisson(lambda0 * areaTotal)  # Poisson number of points
# numberOfNodes =100
xx = xDelta * np.random.uniform(0, 1, numberOfNodes) + xMin  # x coordinates of
Poisson points
yy = yDelta * np.random.uniform(0, 1, numberOfNodes) + yMin  # y coordinates of
Poisson points
# Node Locations
xx1 = [xx]
yy1 = [yy]
nodeLocation = np.concatenate((xx1, yy1))
nodeLocation = np.transpose(nodeLocation)
"(2) Maximum Operational Distance"
# Calculate the distance between nodes
xx = np.transpose(xx)
yy = np.transpose(yy)
distance = np.zeros(shape=(numberOfNodes, numberOfNodes))
for i in range(0, numberOfNodes):
```

```
    for j in range(0, numberOfNodes):
        distance[i][j] = ((xx[i] - xx[j])**2 + (yy[i] - yy[j])**2)**0.5
# Maximum Operational Distance Matrix
maxOperDistance = np.zeros(shape=(numberOfNodes, numberOfNodes))
for i in range(0, numberOfNodes):
    for j in range(0, numberOfNodes):
        if distance[i][j] > 0.2:
            maxOperDistance[i][j] = 0
        else:
            maxOperDistance[i][j] = distance[i][j]
"(3) Adjacency Matrix"
# Calculate Adjacency Matrix
adjacencyMatrix = np.zeros(shape=(numberOfNodes, numberOfNodes))
for i in range(0, numberOfNodes):
    for j in range(0, numberOfNodes):
        if maxOperDistance[i][j] > 0:
            adjacencyMatrix[i][j] = 1
        else:
            adjacencyMatrix[i][j] = 0

"(4) Average Number of Nodes Connected"
# Calculate number of nodes connected to each node
nodeConnected = sum(adjacencyMatrix[:][:] == 1)
# Mean of nodes connected number
averNodeConnected = np.mean(nodeConnected)
"(5) Degree of the Graph"
# Degree Values of each Node
degreeVector = nodeConnected
# The maximum value of degree vector (upper bound)
maxDegree = np.max(degreeVector)
# Check the network G is connected or not
minDegree = np.min(degreeVector)
if minDegree == 0:
    print("The network graph G is disconnected")
else:
    print("The network graph G is connected")

"(6) Total Number of Edges"
numberOfEdges = sum(nodeConnected)/2

"(7) Plot Figures"

# Plotting Network Graph G
plt.figure(1)
```

```
plt.scatter(xx, yy, edgecolor='black', facecolor='r', linewidths=1)
for i in range(numberOfNodes):
    for j in range(numberOfNodes):
        if adjacencyMatrix[i][j] == 1:
            node1 = nodeLocation[i]
            node2 = nodeLocation[j]
            x_value = [node1[0], node2[0]]
            y_value = [node1[1], node2[1]]
            plt.plot(x_value, y_value, color='b', linewidth=0.5, markersize=12)
for i in range(0, numberOfNodes):
    plt.text(xx[i], yy[i]+0.015, i+1, fontsize=8)
plt.xlabel("100")
plt.ylabel("100m")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.title("Network Graph G with " + str(numberOfNodes) + " Random Nodes")
plt.grid()
filename = ('position' + '.pdf')
plt.savefig(filename, format='pdf')
plt.ion()  # plt.show()
plt.pause(4)
plt.close()
"(11) Print out the results"
print("Number of Nodes: " + str(numberOfNodes))
print("Number of Edges: " + str(numberOfEdges))
print("Upper Bound for Number of Colors Used: " + str(maxDegree))
print("Average Number of Nodes Connected to Each Node: " + str(averNodeConnected))
#####         save variable into load.pkl           #####
load             = CL.Load()
load.N_ue        = numberOfNodes
load.nodloc      = nodeLocation
load.ad          = adjacencyMatrix
load.maxDe       = maxDegree
filename = ('load4'+'.pkl')
# filename = os.path.join(dir_path, 'bs'+ '.pkl')
bsfile = open(filename, 'wb')
dill.dump(load, bsfile)
bsfile.close()
```

B.2 MAIN

```
import dill
import numpy as np
import class_set as CL
```

```
from RL_ddpg import RL_ddpg
import ddpg_agent as agent_ddpg
from collections import Counter

load =[]
filename='load4'+'.pkl'
load = dill.load(open(filename, "rb" ))
N_ue   = load.N_ue
Adm    = load.ad
maxDe  = load.maxDe
nodloc = load.nodloc
num_epoch      = 1              ###  number of training epochs
num_slot       = 7000           ###  number of training steps

action_size    = load.maxDe
load.num_epoch = num_epoch
load.num_slot  = num_slot

######### Initialize the machine learning agent  #########
params = {"num_pos": 3,
        "num_UE":  N_ue,
        "appnum":  1,
        "num_intf_bs": 6,
        "num_epoch" : num_epoch,
        "tti_sec" : 1e-3,
        "num_slot": num_slot,
        "pf_alpha": 0.97,
        "alpha" : 5e-5,
        "beta" : 5e-4,
        "gamma": 0.99,
        "tau": 0.001,
        "replay_buffer_size" : 1000000,
        "batch_size" :  64,
        "layer1_size" : 800,
        "layer2_size" : 600,
        "layer3_size":  600,
        "action_space_high" : 1,
        "action_space_low" : -1
         }
agent = {}
agent = agent_ddpg.Agent(alpha=params["alpha"], beta=params["beta"],
gamma=params["gamma"], tau=params["tau"],
                    input_dims=(params["num_UE"]*params["num_UE"]) +
params["num_UE"],
```

```
                    n_actions=params["num_UE"],
                    replay_buffer_size=params["replay_buffer_size"],
batch_size=params["batch_size"],
                    layer1_size=params["layer1_size"], layer2_size=params["layer2_size"],
                    layer3_size=params["layer3_size"],
                    action_space_high=params["action_space_high"],
action_space_low=params["action_space_low"])
######### Initialize actions #########
RLL = {}
RLL = CL.Rll(N_ue)
QoE_all = {}
load.weight = np.sum(Adm) * 0.5 * 0.5/ maxDe

for tt in range(num_epoch):
   print("This is the %sth slot." % tt)
   load.color = np.random.randint(0, load.maxDe, N_ue)
   ###       start machine learning      ###
   RLL.print = 20
   RLL.epoch = tt
   RLL.current_state = np.zeros(N_ue*N_ue+N_ue).astype(int)
   RLL.action = np.zeros(N_ue).astype(int)

   RLL, agent, load = RL_ddpg(RLL, agent, load, tt)
   QoE_all[tt] = []
   QoE_all[tt] = np.array(RLL.QoE_state_vect)
   alloc = []
   alloc = RLL.allocation

   final_color = np.unique(alloc[np.where(alloc >0)[0]])
   print('alloc = \n%s' % alloc)
   print('final_color = \n%s\n' % final_color )
color, colorResult  = [], []
color   = alloc
invalid_links = 0
for i in range(N_ue):
   for j in range(N_ue):
      if Adm[i][j] == 1:
         if color[i] == color[j]:
            invalid_links = invalid_links + 1
invalid_links = invalid_links / 2
colorResult  = len(Counter(color))

bs = []
bs = CL.Bs()
```

```
bs.outcome      = [-RLL.reward, colorResult, invalid_links]
print('The final result is:\n%s\n' %bs.outcome)
```

B.3 DDPG AGENT

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers

class Agent(object):
    def __init__(self, alpha, beta, gamma, tau, input_dims, n_actions,
                 replay_buffer_size, batch_size, layer1_size, layer2_size,layer3_size,
                 action_space_high, action_space_low):
        self.gamma = gamma
        self.tau = tau
        self.input_dims = input_dims
        self.n_actions = n_actions
        self.replay_buffer_size = replay_buffer_size
        self.batch_size = batch_size
        self.layer1_size = layer1_size
        self.layer2_size = layer2_size
        self.layer3_size = layer3_size
        self.action_space_high = action_space_high
        self.action_space_low = action_space_low
        self.mem_cntr = 0
        self.state_buffer = np.zeros((self.replay_buffer_size, self.input_dims ))
        self.action_buffer = np.zeros((self.replay_buffer_size, self.n_actions))
        self.reward_buffer = np.zeros((self.replay_buffer_size, 1))
        self.next_state_buffer = np.zeros((self.replay_buffer_size, self.input_dims))

        # Initialize actor and critic models
        self.actor_model = self.get_actor()
        self.target_actor = self.get_actor()
        self.critic_model = self.get_critic()
        self.target_critic = self.get_critic()

        # Set target weights equal to initial actor/critic weights
        self.target_actor.set_weights(self.actor_model.get_weights())
        self.target_critic.set_weights(self.critic_model.get_weights())

        self.actor_optimizer = tf.keras.optimizers.Adam(alpha)
        self.critic_optimizer = tf.keras.optimizers.Adam(beta)
        # self.noise = OUNoise(mu=np.zeros(n_actions))
```

```
    def record(self, state, action, reward, new_state):
        # Set index to zero if buffer_capacity is exceeded,
        # replacing old records
        index = self.mem_cntr % self.replay_buffer_size

        self.state_buffer[index] = state
        self.action_buffer[index] = action
        self.reward_buffer[index] = reward
        self.next_state_buffer[index] = new_state
        self.mem_cntr += 1

    def update_target(self):

        new_weights = []
        target_variables = self.target_actor.weights
        for i, variable in enumerate(self.actor_model.weights):
            new_weights.append(variable * self.tau + target_variables[i] * (1 - self.tau))

        self.target_actor.set_weights(new_weights)

        new_weights = []
        target_variables = self.target_critic.weights
        for i, variable in enumerate(self.critic_model.weights):
            new_weights.append(variable * self.tau + target_variables[i] * (1 - self.tau))

        self.target_critic.set_weights(new_weights)

    def choose_action(self, state, noise_pre):
        state = state[np.newaxis, :]
        mu = self.actor_model(state)
        noise =self.noise_update(mu, noise_pre)
        mu_prime = mu + noise
        mu_prime=mu
        mu_legal = np.clip(mu_prime, self.action_space_low, self.action_space_high)

        return mu_legal[0], noise

    def noise_update(self, mu, noise_pre):
        is_empty = tf.equal(tf.size(noise_pre), 0)
        if is_empty:
            noise_pre = np.zeros_like(mu)
        sigma, theta, dt = 0.15, .2, 1e-2
        noise = noise_pre + theta * (mu - noise_pre) * dt + sigma * np.sqrt(dt) *
np.random.normal(size=mu.shape)
```

```
        return noise

    def learn(self):
        if self.mem_cntr < self.batch_size:
            return
        # reward_index=np.argmax(self.reward_buffer)  #### Liang 2021.6.9
        batch_indices = np.random.choice(self.mem_cntr, self.batch_size)

        # Convert to tensors
        state_batch = tf.convert_to_tensor(self.state_buffer[batch_indices])
        action_batch = tf.convert_to_tensor(self.action_buffer[batch_indices])
        reward_batch = tf.convert_to_tensor(self.reward_buffer[batch_indices])
        reward_batch = tf.cast(reward_batch, dtype=tf.float32)
        next_state_batch = tf.convert_to_tensor(self.next_state_buffer[batch_indices])

        # Training and updating Actor & Critic networks.
        # See Pseudo Code.
        with tf.GradientTape() as tape:
            target_actions = self.target_actor(next_state_batch)
            y = reward_batch + self.gamma * self.target_critic([next_state_batch,
target_actions])
            critic_value = self.critic_model([state_batch, action_batch])
            critic_loss = tf.math.reduce_mean(tf.math.square(y - critic_value))

        critic_grad = tape.gradient(critic_loss, self.critic_model.trainable_variables)
        self.critic_optimizer.apply_gradients(
            zip(critic_grad, self.critic_model.trainable_variables)
        )

        with tf.GradientTape() as tape:
            actions = self.actor_model(state_batch)
            critic_value = self.critic_model([state_batch, actions])
            # Used `-value` as we want to maximize the value given
            # by the critic for our actions
            actor_loss = -tf.math.reduce_mean(critic_value)

        actor_grad = tape.gradient(actor_loss, self.actor_model.trainable_variables)
        self.actor_optimizer.apply_gradients(
            zip(actor_grad, self.actor_model.trainable_variables)
        )

    def get_actor(self):
        # Initialize weights between -3e-3 and 3-e3
        last_init = tf.random_uniform_initializer(minval=-0.003, maxval=0.003)
```

```
        inputs = layers.Input(shape=(self.input_dims))
        out = layers.Dense(self.layer1_size, activation="relu")(inputs)
        out = layers.BatchNormalization()(out)
        out = layers.Dense(self.layer2_size, activation="relu")(out)
        out = layers.BatchNormalization()(out)
        # out = layers.Dense(self.layer3_size, activation="relu")(out)
        # out = layers.BatchNormalization()(out)
        outputs = layers.Dense(self.n_actions, activation="tanh",
kernel_initializer=last_init)(out)

        model = tf.keras.Model(inputs, outputs)
        return model

    def get_critic(self):
        # State as input
        state_input = layers.Input(shape=(self.input_dims))
        state_out = layers.Dense(128, activation="relu")(state_input)
        state_out = layers.BatchNormalization()(state_out)
        state_out = layers.Dense(256, activation="relu")(state_out)
        state_out = layers.BatchNormalization()(state_out)

        # Action as input
        action_input = layers.Input(shape=(self.n_actions))
        action_out = layers.Dense(256, activation="relu")(action_input)
        action_out = layers.BatchNormalization()(action_out)

        # Both are passed through seperate layer before concatenating
        concat = layers.Concatenate()([state_out, action_out])

        out = layers.Dense(self.layer1_size, activation="relu")(concat)
        out = layers.BatchNormalization()(out)
        out = layers.Dense(self.layer2_size, activation="relu")(out)
        out = layers.BatchNormalization()(out)
        # out = layers.Dense(self.layer3_size, activation="relu")(out)
        # out = layers.BatchNormalization()(out)
        outputs = layers.Dense(1)(out)

        # Outputs single value for give state-action
        model = tf.keras.Model([state_input, action_input], outputs)

        return model
```

B.4 REINFORCEMENT LEARNING DDPG

```
import os
import time
import random
import math
import dill
import numpy as np
from random import randint
import class_set as CL
from collections import Counter

def RL_ddpg(RLL, agent, load, tt):

    print_interval = RLL.print
    for slot in range(load.num_slot):
        # print("This is the %sth slot."%slot)
        run_ddpg(RLL,agent,load, tt)

        if slot % print_interval == 0 and slot != 0:
            print(f"# of num_slots :{slot}, reward = : {RLL.reward}, next_reward = :
{RLL.QoE_state_vect[slot + 1]}")
    return RLL, agent, load

def run_ddpg(RLL, agent, load, tt):
    norm_ue_assoc = normalize_assoc(load.ad)
    used_color = load.color
    norm_used_color = normalize_rate(used_color)
    RLL.next_state = np.hstack((norm_ue_assoc.flatten(),norm_used_color.flatten()))

    get_reward(RLL)
    RLL.score += RLL.reward
    agent.record(RLL.current_state, RLL.action, RLL.reward, RLL.next_state)
    agent.learn()
    agent.update_target()
    RLL.current_state = RLL.next_state
    action=[]
    RLL.action, RLL.noise = agent.choose_action(RLL.current_state, RLL.noise)
    allocation = denormalize_action(RLL.action,load)
    load.color = allocation
    RLL.allocation=allocation
    net_update(RLL, load, tt)
    return RLL, load

def normalize_assoc(cqi):
    min_cqi = 0
```

```
    max_cqi = 1
    lo_cqi = (max_cqi - min_cqi) / 2
    hi_cqi = (max_cqi + min_cqi) / 2
    normalized_assoc = (cqi - hi_cqi) / lo_cqi
    return normalized_assoc

def normalize_rate(rate):
    min_rate = min(rate)
    temp     = max(rate)
    max_rate = max(temp,1)
    lo_rate = (max_rate - min_rate) / 2
    hi_rate = (max_rate + min_rate) / 2
    if lo_rate ==0:
        lo_rate = 0.00001
    normalized_rate = (rate - hi_rate) / lo_rate
    return normalized_rate

def denormalize_action(action,load):
    min_action = 0
    max_action = load.maxDe
    lo_action = (max_action - min_action) / 2
    hi_action = (max_action + min_action) / 2
    denormalized_action = np.rint(lo_action * action + hi_action)
    return np.int_(denormalized_action)

def get_reward(RLL):
    if RLL.reward == True:
        RLL.reward = int(RLL.reward)
        RLL.reward = 0
    else:
        RLL.reward=RLL.QoE
        return RLL

def net_update(RLL, load, tt):
    allocation       = RLL.allocation
    load.allocation = allocation
    #########        calculate the reward          ##########
    N_ue    = load.N_ue
    Adm     = load.ad
    maxDe   = load.maxDe
    nodloc  = load.nodloc
    color   = load.color
    weight  = load.weight
    invalid_links = 0
```

```
for i in range(N_ue):
    for j in range(N_ue):
        if Adm[i][j] == 1:
            if color[i] == color[j]:
                invalid_links = invalid_links + 1
invalid_links = invalid_links / 2
colorResult  = len(Counter(color))
reward_temp  = colorResult + invalid_links /weight
reward_temp  = np.round(100*reward_temp)/100
#########         calculate the reward          ##########
RLL.QoE      =  -reward_temp
RLL.QoE_state_vect.append(-reward_temp)
return RLL,load
```

REFERENCES

[1] Gillis A.S. “What is the internet of things (IoT)?”. TechTarget. [Online]. Available: https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, 2015.

[3] B. Sellami, A. Hakiri, and S. B. Yahia, “Deep Reinforcement Learning for energy-aware task offloading in join SDN-Blockchain 5G massive IoT edge network,” *Future Generation Computer Systems*, 2022.

[4] K. Ergun, R. Ayoub, P. Mercati, and T. Rosing, “Reinforcement learning based reliability-aware routing in IoT networks,” *Ad Hoc Networks*, vol. 132, 2022.

[5] E. Bellini, F. Bagnoli, A. A. Ganin and I. Linkov, "Cyber Resilience in IoT Network: Methodology and Example of Assessment through Epidemic Spreading Approach," *IEEE World Congress on Services (SERVICES)*, pp. 72-77, 2019.

[6] P. Smith et al., "Network resilience: a systematic approach," *in IEEE Communications Magazine*, vol. 49, no. 7, pp. 88-97, July 2011.

[7] J. P. G. Sterbenz, "Smart city and IoT resilience, survivability, and disruption tolerance: Challenges, modeling, and a survey of research opportunities," *9th International Workshop on Resilient Networks Design and Modeling (RNDM*), pp. 1-6, 2017.

[8] Z. Ma, M. Xiao, Y. Xiao, Z. Pang, H. V. Poor, and B. Vucetic, "High-Reliability and Low-Latency Wireless Communication for Internet of Things: Challenges, Fundamentals, and Enabling Technologies," *in IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7946-7970, Oct. 2019.

[9] 3GPP TR 36.881 v0.6.0, “Evolved Universal Terrestrial Radio Access, Study on Latency Reduction Techniques for LTE”.

[10] M. Bennis, M. Debbah, H. V. Poor, “Ultrareliable and Low-Latency Wireless Communication: Tail, Risk and Scale”, *IEEE Proceedings*, vol. 106, no. 10, 2018.

[11] K. A. Nsiah, Z. Amjad, A. Sikora, B. Hilt and J. -P. Lauffenburger, "Latency Reduction Techniques for NB-IoT Networks," *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, pp. 478-482, 2019.

[12] W. U. Khan, J. Liu, F. Jameel, V. Sharma, R. Jäntti and Z. Han, "Spectral Efficiency Optimization for Next Generation NOMA-Enabled IoT Networks," *in IEEE Transactions on Vehicular Technology,* vol. 69, no. 12, pp. 15284-15297, Dec. 2020.

[13] J. G. Hong, S. J. Lee, J. Lim, W. S. Yoon, and S. M. Han, "RF spectrum sensing receiver system with improved frequency channel selectivity for cognitive IoT sensor network applications," *IEEE MTT-S International Microwave Symposium (IMS)*, pp. 1-3, 2016.

[14] E. Axell, G. Leus, E. Larsson, and H. Poor, "Spectrum sensing for cognitive radio: State-of-the-art and recent advances*," Signal Processing Magazine*, IEEE, vol. 29, no. 3, pp. 101–116, May 2012.

[15] A. Bujunuru and T. Srinivasulu, "A Survey on Spectrum Sensing Techniques and Energy Harvesting," *International Conference on Recent Innovations in Electrical, Electronics & Communication Engineering (ICRIEECE),* 2018, pp. 751-755, 2018.

[16] S. Ziafat, W. Ejaz and H. Jamal, "Spectrum Sensing Techniques for Cognitive Radio Networks: Performance Analysis," *IEEE MTT-S International Microwave Workshop Series on Intelligent Radio for Future Personal Terminals*, pp. 1-4, 2011.

[17] M. P. Mishra and P. C. Saxena, "Survey of Channel Allocation Algorithms Research for Cellular Systems," *International Journal of Networks and Communications*, pp. 75-104, 2012.

[18] U. Sharma, P. Mittal, and C. K. Nagpal, "Implementing Game Theory in Cognitive Radio Network for Channel Allocation: An Overview," *International Journal of Energy, Information and Communications*, vol. 6, no. 2, pp. 17-22, 2015.

[19] P. M. Papazoglou, D. A. Karras, and R. C. Papademetriou, "A Critical Overview on the Recent Advances in Channel Allocation Strategies for voice and Multimedia Services in Wireless Communication Systems and the Applicability of Computational Intelligence Techniques," *Mathematical Methods, Computational Techniques, Non-linear Systems, Intelligent Systems,* 2008.

[20] L. Le, W. Zhang, A. Festag, and R. Baldessari, "Analysis of Approaches for Channel Allocation in Car-to-Car Communication," *1st International Workshop on Interoperable Vehicles (IOV 2008)*, pp. 33-38, 2008.

[21] Z. Al Aghbari, A. M. Khedr, W. Osamy, et al, "Routing in Wireless Sensor Network Using Optimization Techniques: A Survey," *Wireless Personal Communication*, vol. 111, pp. 2407-2434, 2020.

[22] R. Eberhar and J. Kennedy, “A New Optimizer using Particle Swarm Theory,” *Proceedings of the sixth international symposium on micro machine and human science*, pp. 39-43, 1995.

[23] E. Karami and S. Glisic, "Optimization of routing, network coding and scheduling in wireless multicast ad-hoc networks with topology compression," *IEEE 20th International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 410-414, 2009.

[24] M. Kumar, K. K. Pattanaik, B. Yadav and R. K. Verma, "Optimization of Wireless Sensor Networks inspired by Small World Phenomenon," *IEEE 10th International Conference on Industrial and Information Systems (ICIIS)*, pp. 66-70, 2015.

[25] A. Mittal, P. Jain, S. Mathur, and P. Bhatt, "Graph Coloring with Minimum Colors: An Easy Approach," *International Conference on Communication Systems and Network Technologies*, pp. 638-641, 2011.

[26] G. L. Prajapati, A. Mittal, and N. Bhardwaj, "An efficient coloring of graphs using less number of colors," *World Congress on Information and Communication Technologies*, pp. 666-669, 2012.

[27] D. Brelaz, “New Methods to Color Vertices of a Graph,” *Communications of the ACM 22*, pp. 251-256, 1979.

[28] A. Hertz and D. de Werra, “Using Tabu Search Techniques for Graph Coloring,” *Computing 39(4)*, pp. 345-351, 1987.

[29] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, “Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and number Partitioning,” *Operational Research 39(3)*, pp. 378-406, 1991.

[30] J. S. Turner, “Almost All k-colorable Graphs are Easy to Color,” *Journal of Algorithms 9(1),* pp. 63-82, 1988.

[31] S. Xiao, W. Li, L. Yang and Z. Wen, "Graph-Coloring Based Spectrum Sharing for V2V communication," *International Conference on UK-China Emerging Technologies (UCET)*, pp. 1-4, 2020.

[32] D. Orden, I. M. Maestre, J. M. G. Guzman, E. Hoz, and A. A. Suarez, “Spectrum Graph Coloring to Improve Wi-Fi Channel Assignment in a Real-World Scenario via Edge Contraction,” *Discrete Applied Mathematics*, vol. 263, pp. 234-243, 2019.

[33] F. Yunlong, L. Jianhui, L. Qing, and Z. Xiaoyi, “Downlink Channel Allocation of Visible Light Communication Network Based on Graph Coloring and Traffic Fairness,” *Procedia Computer Science*, vol. 107, pp. 667-673, 2017.

[34] A. Reichman, S. Wayer, and M. P. Moreno, "Resource Allocation in Wireless Mesh Networks," *IEEE International Conference on the Science of Electrical Engineering in Israel (ICSEE)*, pp. 1-5, 2018.

[35] B. Yao, J. Yin, H. Li, H. Zhou, and W. Wu, "Channel resource allocation based on graph theory and coloring principle in cellular networks," *IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA),* pp. 439-445, 2018.

[36] A. Grami, *Introduction to Digital Communications*. Academic Press, 2016.

[37] M. Abaii, Y. Liu and R. Tafazolli, "An Efficient Resource Allocation Strategy for Future Wireless Cellular Systems," *in IEEE Transactions on Wireless Communications*, vol. 7, no. 8, pp. 2940-2949, August 2008.

[38] X. Liu, X. Shang, X. Dong, and G.Xiong, “Chapter 9 - Behavior modeling and its application in an emergency management parallel system for chemical plants,” *Big Data and Smart Service Systems,* pp. 139-150, 2017.

[39] H. T. Friis, "A Note on a Simple Transmission Formula," *in Proceedings of the IRE*, vol. 34, no. 5, pp. 254-256, May 1946.

[40] Q. Qiong and W. Dongxia, "Evaluation method for node importance in complex networks based on eccentricity of node," *2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 2499-2502, 2016.

[41] M. Newman, Networks, 2nd ed. Oxford University Press, 2018.

[42] Tan Huazhen, "The application of price demand elasticity theory in commodity Price Adjustment [J]", *Journal of Jiaozuo Institute of Technology (Social Science Edition)*, no. 3, pp. 29-30, 2001.

[43] C. Godsil and G. Royle, Algebraic Graph Theory. Springer, 2001.

[44] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]: http://neuralnetworksanddeeplearning.com/

[46] L. Zhang, B. Jabbari and N. Ansari, "Deep Reinforcement Learning Driven UAV-assisted Edge Computing," *in IEEE Internet of Things Journal*, 2022.

BIOGRAPHY

Hieu Thanh Nguyen received the Bachelor of Science in Electrical Engineering in 2021, then has pursued a Master of Science in Electrical Engineering concentration in Networks and Communication at George Mason University since 2022. He has worked as a Graduate Research Assistant in the Communications and Networking Lab and contributed to the BRIDGES research project funded by NSF for one year and a half.