

Towards Machine Learning Enabled Future-Generation Wireless Network Optimization

by

Peizhi Yan

B.Sc. Algoma University, 2018

B.Eng. University of Jinan, 2019

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE FACULTY OF GRADUATE STUDIES
OF LAKEHEAD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

© Copyright 2020 by Peizhi Yan
Lakehead University
Thunder Bay, Ontario, Canada

Towards Machine Learning Enabled Future-Generation Wireless Network Optimization

by

Peizhi Yan

Supervisory Committee

Dr. Salimur Choudhury,

Supervisor

*(Department of Computer Science, Lakehead University, Thunder Bay,
Ontario, Canada)*

Dr. Shan Du,

Co-supervisor

*(Department of Computer Science, Lakehead University, Thunder Bay,
Ontario, Canada)*

Dr. Dariush Ebrahimi,

Internal Examiner

*(Department of Computer Science, Lakehead University, Thunder Bay,
Ontario, Canada)*

Dr. Abdallah Shami,

External Examiner

*(Department of Electrical and Computer Engineering, Western University, London,
Ontario, Canada)*

ABSTRACT

We anticipate that there will be an enormous amount of wireless devices connected to the Internet through the future-generation wireless networks. Those wireless devices vary from self-driving vehicles to smart wearable devices and intelligent household electrical appliances. Under such circumstances, the network resource optimization faces the challenge of the requirement of both flexibility and performance. Current wireless communication still relies on one-size-fits-all optimization algorithms, which require meticulous design and elaborate maintenance, thus not flexible and cannot meet the growing requirements well. The future-generation wireless networks should be “smarter”, which means that the artificial intelligence-driven software-level design will play a more significant role in network optimization.

In this thesis, we present three different ways of leveraging artificial intelligence (AI) and machine learning (ML) to design network optimization algorithms for three wireless Internet of things network optimization problems. Our ML-based approaches cover the use of multi-layer feed-forward artificial neural network and the graph convolutional network as the core of our AI decision-makers. The learning methods are supervised learning (for static decision-making) and reinforcement learning (for dynamic decision-making). We demonstrate the viability of applying ML in future-generation wireless network optimizations through extensive simulations. We summarize our discovery on the advantage of using ML in wireless network optimizations as the following three aspects:

1. Enabling the distributed decision-making to achieve the performance that near a centralized solution, without the requirement of multi-hop information;
2. Tackling with dynamic optimization through distributed self-learning decision-making agents, instead of designing a sophisticated optimization algorithm;
3. Reducing the time used in optimizing the solution of a combinatorial optimization problem.

We envision that in the foreseeable future, AI and ML could help network service designers and operators to improve the network quality of experience swiftly and less expensively.

ACKNOWLEDGEMENTS

“The present is theirs; the future, for which I really worked, is mine.”¹

– Nikola Tesla

I would like to give thanks to the following funding sources, for their financial support to my research:

- **Lakehead University Faculty of Graduate Studies;**
- **Lakehead University Faculty of Science and Environmental Studies;**
- **Dr. Salimur Choudhury (Discovery Grant);**
- **Toronto Vector Institute.**

I would like to thank my supervisors, Dr. Salimur Choudhury and Dr. Shan Du, for giving me excellent guidance and support not only to my research but also to my past two years’ academic life. I appreciate Dr. Ruizhong Wei for his constructive suggestions for my work in Chapter 3. Thanks, Dr. Fadi Al-Turjman, and Dr. Ibrahim Al-Oqily, for introducing the topology control project (in Chapter 5) to us. Thanks to Dr. Dariush Ebrahimi and Dr. Abdallah Shami for their constructive comments. I also want to give thanks to my teachers, friends, and everyone who helped me in my life. Last but not least, I gratitude to my family for their support, encouragement, and love.

¹In Margaret Cheney’s book, *Tesla: Man Out of Time* [15], “During the lawsuit which I’ve instituted against Mr. Marconi for stealing my apparatus and drawings from the Patent Office,’ Tesla allegedly continued, ‘Mr. Pupin, called to testify on my behalf as a countryman, went on the side of Mr. Marconi, who, after three years of legal battle was forced to admit under oath that the transmission of power to long distances is my invention.’ Tesla paused, then added, ‘Let the future tell the truth and evaluate each one according to his work and accomplishments. The present is theirs, the future, for which I really worked, is mine.’”

PUBLICATIONS

Parts of this thesis have been submitted for peer-review, published or accepted for publication:

- The paper **A Distributed Graph-Based Dense RFID Readers Arrangement Algorithm** [138] was published in the proceedings of the *IEEE International Conference on Communications 2019*. (part of Chapter 3)
- The article **A Machine Learning Auxiliary Approach for the Distributed Dense RFID Readers Arrangement Algorithm** [139] has been accepted by the *IEEE Access on Intelligent and Cognitive Techniques for Internet of Things*. (part of Chapter 3)
- The paper **Optimizing Mobile Edge Computing Multi-Level Task Offloading via Deep Reinforcement Learning** [137] has been accepted by the *IEEE International Conference on Communications 2020*. (part of Chapter 4)
- The article **An Energy-Efficient Topology Control Algorithm for Optimizing the Lifetime of Wireless Ad-hoc IoT Networks in 5G and B5G** [136] has been submitted to *Elsevier Computer Communications*. (part of Chapter 5)

Contents

Supervisory Committee	ii
Abstract	iii
Acknowledgements	iv
Publications	v
Table of Contents	vi
List of Tables	ix
List of Figures	x
1 Introduction	1
2 Background	6
2.1 Radio-Frequency Identification Networks	6
2.2 Mobile Edge Computing	7
2.3 Wireless Ad-hoc Networks	8
2.4 Graph Theory Basics	9
2.5 Artificial Neural Networks	10
2.5.1 Multi-Layer Feed-Forward Network	11
2.5.2 Graph Convolutional Network	13
2.5.3 Learning the Weights: Gradient Descent with Backpropagation	16
2.6 Deep Reinforcement Learning	18
3 Dense RFID Network Collision Avoidance	20
3.1 Introduction	21
3.2 Related Works	24

3.3	Problem Statement	26
3.4	MWIS Algorithms and MWISBAII	26
3.4.1	GWMIN2 Algorithm	27
3.4.2	MWISBAII	28
3.5	The Distributed MWISBAII	29
3.6	Machine Learning Auxiliary Approach	33
3.6.1	Machine Learning Stage	34
3.6.2	Application Stage	36
3.7	Experimental Results	38
3.8	Summary	42
4	Mobile Edge Computing Task Offloading Optimization	44
4.1	Introduction	45
4.2	Related Work	46
4.3	Network Model	47
4.4	Problem Statement	48
4.4.1	Local Computing Mode Cost	49
4.4.2	Task Offloading Modes Cost	50
4.4.3	Optimization Objective	51
4.5	The Proposed Optimization Approach	51
4.6	Simulation and Experimental Results	54
4.7	Summary	64
5	Wireless Ad-hoc Network Topology Control	65
5.1	Introduction	66
5.2	Existing Topology Control Algorithms	68
5.3	System Model and Problem Statement	72
5.4	The Proposed Approach	75
5.4.1	Energy Distribution Index	75
5.4.2	Energy-Degree Topology Control Algorithm	78
5.4.3	Graph Convolutional Network Based Energy-Degree Topology Control Algorithm	80
5.5	Experiments	83
5.6	Summary	95
6	Conclusion	97

A List of Symbols and Notations	101
B List of Abbreviations	108
Bibliography	112

List of Tables

Table 1.1	A Brief Summary of 3G, 4G, 5G, and B5G	2
Table 2.1	Signal Propagation in an MLF Network	13
Table 3.1	Simulation Settings for Training	39
Table 4.1	Experimental Parameters [2, 52, 92]	56
Table 5.1	The Comparison of Different Topology Control Algorithms	68
Table 5.2	Some Mathematical Notations	81
Table 5.3	Goodness-of-Fit Test Results	86
Table 5.4	Experimental Parameters [4], [135]	88
Table 5.5	Area Under the Curve (The Second Experiment)	89
Table 6.1	Summary of the Proposed Machine Learning-Based Algorithms	98

List of Figures

Figure 1.1	An Outlook of the Future-Generation Wireless Networks	3
Figure 2.1	The MEC Conceptual Architecture	8
Figure 2.2	Example Un-directed Graph	10
Figure 2.3	Biological Neuron and Artificial Neuron	12
Figure 2.4	Example Multi-Layer Feed-Forward Neural Network .	13
Figure 3.1	Some Types of Collisions in RFID	22
Figure 3.2	Example Dense RFID System Anti-collision Solution .	23
Figure 3.3	Example Maximum Weight Independent Set Problem	28
Figure 3.4	Example Dense RFID System	29
Figure 3.5	Demonstration of the Distributed MWISBAII Algorithm	32
Figure 3.6	Simulated Dense RFID System	34
Figure 3.7	Machine Learning Stage Flowchart	35
Figure 3.8	Application Stage Flowchart	37
Figure 3.9	Testing ROC Curves	38
Figure 3.10	K-Means Clustering on RFID Readers	39
Figure 3.11	Performance Comparison (1)	40
Figure 3.12	Performance Comparison (2)	41
Figure 3.13	Performance Comparison (3)	42
Figure 4.1	The Proposed Wireless Edge Computing Network Scheme	46
Figure 4.2	The Proposed Multi-Level Task Offloading Pipeline .	54
Figure 4.3	Deep Q-Learning Loss Curves	57
Figure 4.4	Average Cumulative Delay (Experiment 1)	60
Figure 4.5	Average Cumulative Energy Consumption (Experiment 1)	61
Figure 4.6	Average Cumulative Delay (Experiment 2)	62

Figure 4.7	Average Cumulative Energy Consumption (Experiment 2)	63
Figure 5.1	Example LTCA Solution	71
Figure 5.2	A Demonstration of Unsatisfactory Network Topologies	74
Figure 5.3	The Demonstration of Value Range Normalization Process	77
Figure 5.4	Maximum Spanning Tree Solution Demonstration . .	78
Figure 5.5	The Visualization of Some Matrices and Graphs . . .	84
Figure 5.6	Comparison of the original topology G , G'_{max} , and G'_{min}	85
Figure 5.7	The Distribution of ED Index	85
Figure 5.8	Topologies Derived by Different Topology Control Algorithms	87
Figure 5.9	Number of Alive Nodes (1)	90
Figure 5.10	Number of Alive Nodes (2)	91
Figure 5.11	The Distribution of ED Indices of Different Topologies	92
Figure 5.12	Average Node Degree and Average Topology Construction Time	92
Figure 5.13	Network Lifetime (Small Instances)	94
Figure 5.14	Network Lifetime (Large Instances)	95

Chapter 1

Introduction

The past decade has witnessed two significant evolutions in commercial telecommunication, from 3G to 4G, then to 5G. To most customers, the direct experience of each upgrade is the faster data rate and better reliability (lower latency, less packet loss). For instance, the maximum data rate of 3G is 2 Mbps (allows video streaming), whereas the data rate of 4G is up to 1 Gbps (provides broadband Internet experience) [28]. More and more mobile services emerge in the past decade, such as video chat, mobile gaming, and video streaming. One objective of the future-generation telecommunication is to connect the items in our daily life to the Internet, which is known as the Internet of Things (IoT) [6]. Some examples of IoT are smart wearable devices (such as smartwatches), vehicular networks, and smart home [72]. The telecommunication plays a significant role in the realization of IoT because most IoT devices require a wireless connection. Although the 3G and 4G technologies are widely used in today's IoT devices, they are not fully optimized for IoT applications [3]. For example, notwithstanding the 4G network supports connecting many devices to the Internet within a region, it can not meet the requirement of connecting a massive amount of devices. Therefore, many IoT devices such as smartwatches still need to access the Internet through some host devices such as mobile phones. Besides, the connections between those IoT devices and the host devices are often established via WiFi and Bluetooth technologies, which leverage unlicensed wireless channels and cannot guarantee a high quality of service (QoS). To support a ubiquitous IoT or even Internet of Everything (IoE) [87], new telecommunication technologies (both hardware and software) are necessary. In the IEEE 5G and Beyond Technology Roadmap White Paper [39], the better implementation of the IoT is listed as an objective of 5G and beyond 5G (B5G). Some key technologies of 5G not only support a much

higher data rate and extremely low latency but also pave the road for connecting a massive amount of IoT devices. We briefly summarize the different generations of telecommunication technologies (from 3G to B5G) in Table 1.1, to present a better comparison.

	3G	4G	5G	B5G
Introduced in	1998	2001 to 2002	2000s	-
Commercialized in	2001	2009	-	-
Frequency	up to 2.5 GHz	2 to 8 GHz	sub-6 GHz; 24 to 100 GHz	sub-6 GHz; 24 to 100 GHz; >100 GHz
Rate (up to)	2 Mbps	2 Mbps to 1 Gbps	1 Gbps	100 Gbps
Latency	100 to 500 ms	< 100 ms	8 to 12 ms	1 ms
Key technologies	CDMA; Beam-forming; Link adaptation	OFDM; MIMO; IP telephony	Beam-forming; mmWave; D2D communication; Massive MIMO; Full duplex	Software-centric design; Artificial intelligence; Blockchain
Applications	Video streaming	Mobile gaming; Cloud computing; IoT	Vehicular network; UAV network; Video game streaming; Smart grid; Edge computing; Network slicing; Remote surgery	VR/AR application; IoE; ICN

References: [19, 28, 39, 93, 104, 111].

“-” denotes unclear or has not happened. Some abbreviations see Appendix B.

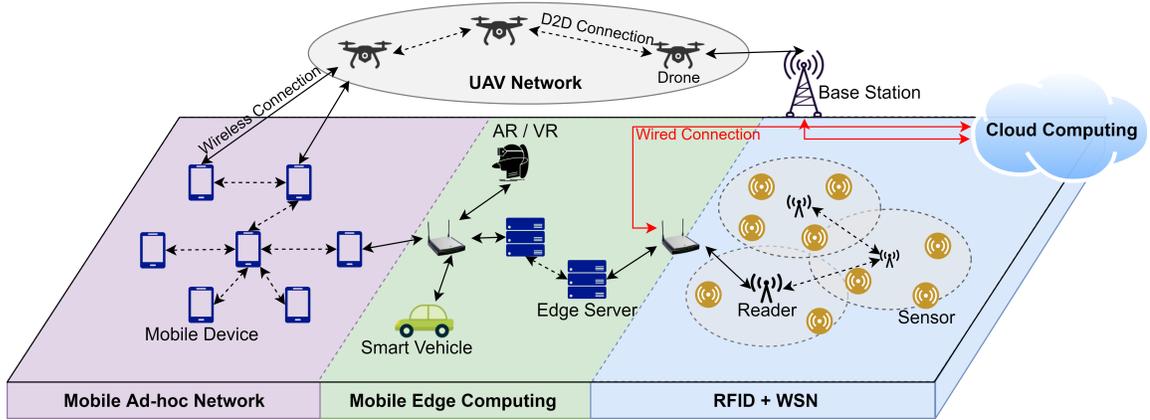
Table 1.1: A Brief Summary of 3G, 4G, 5G, and B5G

Apart from IoT, artificial intelligence (AI), especially machine learning (ML), was also thriving in the past decade. One reason is that the computing power of today’s computing devices satisfies most AI and ML algorithms; the other reason is that enabling the machine to learn and act as human intelligence is a trend in the development of modern technologies. Sun et al. pointed out that the 5G networks will be more complex, which in turn leads to more complex mathematical formulations in developing conventional algorithms to optimize the network [117]. Moreover, the traditional algorithms can be infeasible due to a large number of network nodes in the 5G networks. Therefore, exploring the application of AI and ML in future-generation wireless network optimization problems is promising. We list some potential benefits of integrating ML with the future-generation wireless network optimization in below:

1. Achieving better performance than the conventional algorithms;
2. Achieving similar performance but with much lower complexity than the con-

ventional algorithms;

3. Saving the cost of designing a complex mathematical model;
4. Improving the flexibility and adaptability of the algorithm;
5. Helping realize the self-optimization of distributed network nodes;
6. Making better use of the hidden patterns of the network.



© Peizhi Yan

Figure 1.1: An outlook of some technologies and applications in the future-generation (5G and B5G) wireless networks.

In this thesis, we explore and demonstrate the potential applications of ML approaches/algorithms in three particular future-generation IoT network optimization problems. Figure 1.1 provides an outlook of the radio-frequency identification (RFID) networks, mobile edge computing (MEC) networks, and wireless ad-hoc IoT (WAIoT) networks in the future-generation wireless communication. We envision that the application of artificial intelligence and machine learning in wireless IoT optimization will be ubiquitous due to the increase of device computing power and the improvement in network performance. The rest of this thesis is organized as follows. Chapter 2 gives a review of the fundamental concepts, theories, and methods used in this thesis. Chapter 3 presents the first problem, dense RFID network collision avoidance [138]. RFID is an important component of IoT, which allows quick and automatic identification of the objects with RFID tags [99]. RFID is widespread in industries such as supply-chain management and logistics due to its low-cost feature. One often needs to leverage a considerable amount of RFID readers to cover a large area. Due to

the overlapping coverage of different RFID readers, the collision problem is likely to happen and need to be handled. We first introduce a distributed collision avoidance algorithm, which only leverages one-hop information. In our experiment, there is a performance gap between the distributed algorithm and its centralized counterpart. To reduce this gap, we utilize the centralized algorithm to generate training data and train (supervised learning) a fully-connected artificial neural network (ANN) to help the distributed algorithm at the early decision-making stage. Experimental results show that the distributed algorithm with ML auxiliary can get almost the same performance as its centralized counterpart.

Chapter 4 presents the second problem, task-offloading optimization in MEC. We consider the scenario that in a large crowded area (such as a stadium, a concert hall, or a shopping mall), multiple wireless edge gateways (each edge gateway has an edge server) are deployed to provide MEC services. The concept of this type of MEC network is similar to the idea of the personal cloud network [108]. We do not require the direct-wired connection between each pair of edge gateways. Instead, edge gateways can communicate with each other wirelessly. The reason is that this way allows the deployment of the edge gateways to be more flexible. For example, the organizers can rent the edge gateways only when needed, to save the cost. The user device could offload tasks to its nearby edge server to achieve a better QoS. However, when the edge server is overload, offloading a task to the edge server might reduce the QoS. Thus, the task-offloading scheduler needs to make task offloading decisions in terms of the current system (both the device itself and the network system) status. We also consider the situation when the user devices are not evenly distributed. For instance, people might gather around some places. In this case, some edge servers might be overload, while other edge servers are relatively idle. Therefore, we enable the edge servers to offload tasks to their neighbor edge servers further. We model this task-offloading optimization problem as a multi-level (device-level and edge-level) joint optimization problem and apply deep Q-learning (a reinforcement learning method) for each level of optimization [137]. Simulation results demonstrate the prominent trade-off (delay and energy consumption) performance of our proposed approach.

Chapter 5 presents the third problem, energy-efficient topology control in WAIoT networks [97]. Conventional commercial telecommunications either not support device-to-device (D2D) communications or do not make the best of D2D technology. Whereas, in 5G, D2D communication is encouraged in helping with the massive connectivity

[39]. The WAIoT network is formed by IoT devices that can communicate with their neighbor devices through D2D connections. Such network paradigm could also benefit the information-centric network (ICN) or content-centric network (CCN) [77]. However, most wireless IoT devices have a limited power supply, such as a battery. Thus, when some devices are run out of energy, the WAIoT network will be disconnected. We assume that by reducing the number of connections of the devices with less energy and increasing the number of connections of the devices with more energy could improve the overall network lifetime. Based on this assumption, we propose a centralized topology control algorithm, named EDTC [136]. The proposed algorithm achieves better performance than the state-of-the-art in terms of network lifetime. To reduce the topology optimization time, we further train a graph convolutional network (GCN) to imitate the proposed algorithm. Experimental results show that the GCN-based approach can achieve similar performance to EDTC while significantly reduce the running time.

Finally, we conclude our work and put forward some future research directions in Chapter 6.

Chapter 2

Background

This chapter gives a review of three types of wireless network technologies, as well as the existing theories and methods that are fundamental for the rest of this thesis.

2.1	Radio-Frequency Identification Networks	6
2.2	Mobile Edge Computing	7
2.3	Wireless Ad-hoc Networks	8
2.4	Graph Theory Basics	9
2.5	Artificial Neural Networks	10
2.5.1	Multi-Layer Feed-Forward Network	11
2.5.2	Graph Convolutional Network	13
2.5.3	Learning the Weights: Gradient Descent with Backpropagation	16
2.6	Deep Reinforcement Learning	18

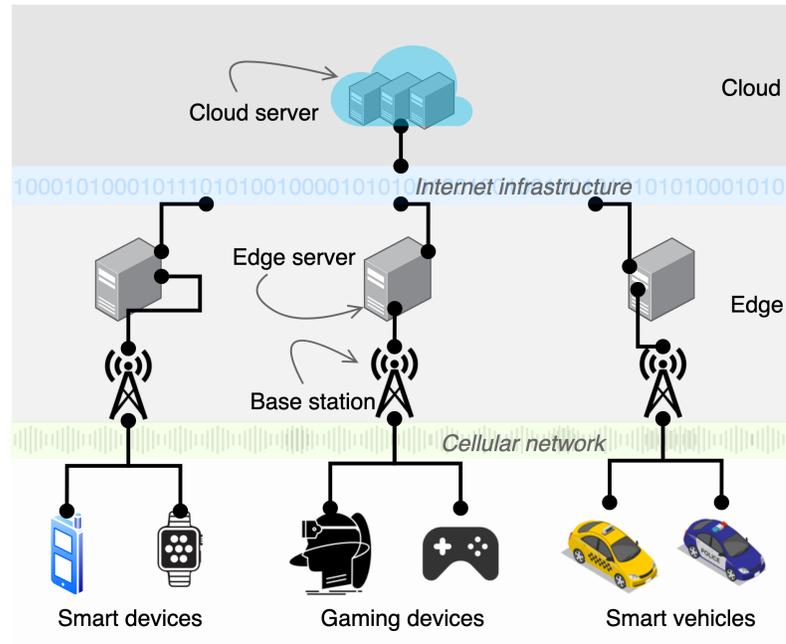
2.1 Radio-Frequency Identification Networks

Radio-frequency identification (RFID) is an automatic identification and data capture technology, using radio-frequency electromagnetic waves to transmit signals [122]. A normal RFID system has three major types of components: RFID reader(s), RFID tag(s), and the host system (or central computer) [99]. The RFID reader can read (collect) the information stored on the RFID tag. The collected information will be sent to the host system for further processing (such as saving into the database).

Because RFID tags are inexpensive and extremely portable, RFID technology is an essential part of the modern IoT world [18]. Some common usages of RFID systems are product identification (to replace the traditional bar code), theft-detection, and contact-less payment [55, 110]. In addition, RFID can also be applied in positioning [30, 126, 128]. Both RFID readers and tags have antenna for communication. Based on the type of power source, RFID tags can be categorized into active RFID tags and passive RFID tags. The antenna of a passive RFID tag serves as both the power receiver and the signal transmitter. To read the information recorded on a passive RFID tag, the RFID reader needs to emit the electromagnetic wave to power the passive RFID tag. Whereas, an active RFID tag should have an internal power source (usually a battery) to power the antenna and the microchip. Due to this reason, the transmission range of an active RFID tag is usually much more extensive than the transmission range of a passive RFID tag. However, active RFID tags are more expensive and larger than passive RFID tags [99]. In contrary, the passive RFID tags are much smaller (often as thin as paper), so they can be installed in a passport, a luggage tag, or even a book [7, 8].

2.2 Mobile Edge Computing

Mobile edge computing (MEC) is one of the promising technologies in future IoT networks [116]. The context of the application of MEC varies from virtual reality [141] to smart vehicular network [89]. MEC aims to reduce latency, ensure network efficiency, and improve user experience. To implement MEC, we need the virtualized platform, which is supported by 5G [48]. Moreover, it is anticipated that the MEC will become an essential component in the 5G network, which supports diverse novel applications and services [1]. MEC reduces the reliance of smart mobile end devices (such as cellphones) to the cloud service and allows many functions to be performed “offline” (on the edge server). In MEC, edge servers could collect and process the data locally. Thereby, some sensitive information does not need to go through the cloud server. The cloud servers are usually remote, which is inefficient to some time-sensitive tasks [109]. MEC could reduce the overhead of the communication with the remote cloud server. In summary, some advantages of MEC include low latency, end device battery conserving, privacy protection, robustness, and bandwidth saving. Figure 2.1 depicts the concept of an MEC network.



© Peizhi Yan

Figure 2.1: The MEC conceptual architecture.

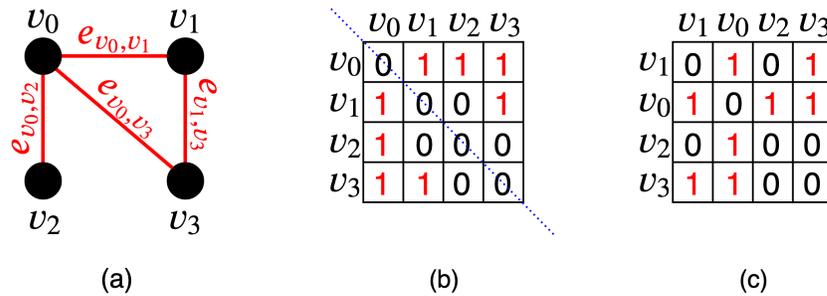
2.3 Wireless Ad-hoc Networks

Many IoT networks are also wireless ad-hoc networks (WANET) because such networks have the advantages of flexibility, better fault tolerance, and the support of rapid deployment. Some examples of wireless ad-hoc IoT (WAIoT) networks are wireless sensor networks (WSN) [5], unmanned aerial vehicle networks [41, 100], and smart vehicular networks [43]. Conventionally, one has to leverage technologies such as WiFi and Bluetooth to form WAIoT networks over unlicensed channels. However, the quality of service is not guaranteed for those approaches. One objective of 5G and beyond 5G (B5G) is to connect almost everything ubiquitously and also provide a high data rate. Thus, it is promising to take advantage of the WAIoT paradigm to meet the growing demand for next-generation cellular networks. [120] provided an outlook of the D2D use cases in 5G cellular networks. One scenario is to allow the WAIoT network to provide Internet services in congested areas such as a stadium and a shopping mall. Unmanned aerial vehicles (UAVs) can also form an ad-hoc network and act as flying base stations to assist communications for the 5G devices [73]. In [58] and [147], the authors envisioned the scenarios of implementing the content-centric paradigm over WAIoT networks in the future-generation cellular networks.

2.4 Graph Theory Basics

This section refers to Reinhard Diestel's book *Graph Theory* [21]. Graph theory is a sub-field of mathematics (more specifically, discrete mathematics), which studies the structure (named graph) of modeling objects and the relationships among those objects. We can represent a graph G (denote as $G = (V, E)$) as the combination of a set of vertices (nodes) V and a set of edges (links) E . Denote $v_i \in V$ as the i^{th} vertex in V . An edge $e_{v_i, v_j} \in E$ represents that there is direct relationship between the v_i and v_j in G . If there is an edge e_{v_i, v_j} in G , then we say v_i and v_j are neighbors, or v_i and v_j are adjacent. We denote $neighbors(G, v_i)$ as the set of all the neighbors of v_i in G . In terms of whether the order of two vertices appears in e_{v_i, v_j} has meaning or not, we can classify a graph as either un-directed graph or directed graph (also known as digraph). In an un-directed graph, $e_{v_i, v_j} = e_{v_j, v_i}$. Whereas, in a digraph, e_{v_i, v_j} means that there is an edge start from v_i and end at v_j ; thus, $e_{v_i, v_j} \neq e_{v_j, v_i}$. The degree of a vertex v_i (denoted as $deg(v_i)$) is the number of edges with that vertex as an end-point (e.g., in e_{v_i, v_j} , v_j is the end-point). We can also assign weights to vertices and/or edges, to represent some quantitative attributes. In this thesis, we use $w(v_i)$ to represent the weight of v_i ; use $w(e_{v_i, v_j})$ to represent the weight of an edge e_{v_i, v_j} . Figure 2.2a shows an example of un-directed graph with four vertices and four edges in form of a diagram. We often use the adjacency matrix to represent a graph. If the edges have no weights, we can use a binary adjacency matrix $\dot{A} \in \{0, 1\}^{|V| \times |V|}$ to represent the graph. $\dot{A}_{ij} = 1$ represents that there is an edge between v_i and v_j . Figure 2.2b depicts the adjacency matrix of the graph in Figure 2.2a. For un-directed graph, $\dot{A}_{ij} = \dot{A}_{ji}$, therefore, the \dot{A} is symmetric along the diagonal (see the dashed line in Figure 2.2b). For the same graph, there could have different adjacency matrix representations of it. For example, the adjacency matrix shown in Figure 2.2c also represents the graph in Figure 2.2a. If the edges have weights, we can use the weighted adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$ to represent the graph, where $A_{ij} = w(e_{v_i, v_j})$.

A path represents a sequence of distinct vertices (except for the initial vertex could be the final vertex), where each pair of adjacent vertices is connected by an edge, and the edges are also distinct. The number of vertices in a path minus 1 represents the length of the path. For instance, $v_0 \rightarrow v_1 \rightarrow v_3$ represents the path (length is 2) starting from v_0 (initial vertex) and ending at v_3 (final vertex). We say a graph is connected if there exists a path between each pair of vertices. A bipartite graph is a special type of graph if the vertex set can be split into two disjoint sets so that each



© Peizhi Yan

Figure 2.2: An example un-directed graph (a) and its two possible adjacency matrix representations (b, c).

edge of G connects a vertex from one set to a vertex from the other set. We denote a bipartite graph as $G = (V_a, V_b, E)$, where $V_a \cap V_b = \emptyset$; $V_a \cup V_b = V$; and $e_{v_i, v_j} \in E$ if and only if $v_i \in V_a, v_j \in V_b$. If the initial vertex is also the final vertex, we say the path is a cycle. The graph contains no cycle is known as the acyclic graph. Acyclic graphs are bipartite graphs. The connected acyclic graph is also known as a tree.

2.5 Artificial Neural Networks

Artificial neural networks (ANN) are mathematical models designed to solve a variety of problems such as pattern recognition, autonomous control, and optimization, through learning [54]. A biological neural system contains an enormous amount of information/signal processing units called biological neurons. Compared with the Von Neumann architecture computer [125], which relies on a centralized controller to execute manually defined sequential procedures, the biological neural system has a significant advantage due to its parallel processing nature and the massive amount of connections between neurons [54]. Biological neural networks inspired the invention of ANN, and the study on ANNs can also help us understand how the biological brain works [83]. The fundamental work on the birth and development of ANN may date back to Frank Rosenblatt's perceptron machine [101]. In 1957, Frank Rosenblatt proposed the perceptron machine to imitate the biological nerve net in learning to perform binary classification on linearly separable signals. The perceptron can be either implemented on a hardware-level or simulated through computer programs [102]. Figure 2.3 depicts two connected biological neurons. The connections between two biological neurons are established through synapses and dendrites. The synapses

of the first neuron (the left neuron) can release chemicals called neurotransmitters to pass a signal to the second neuron (the right neuron). Similar to the biological neural network, the basic building block of an ANN is called an artificial neuron. We show an example architecture of the artificial neuron with two inputs (x_0 and x_1) in Figure 2.3. Computation inside the artificial neuron is represented through the computational graph. We can use an equation to summarize the computation:

$$o = \textit{activation}\left(\sum_{i=0}^1 \mathbf{a}_i x_i + \mathbf{b}\right), \quad (2.1)$$

where o is the output of the neuron; \mathbf{a}_i is the weight for input x_i ; \mathbf{b} is bias; $\textit{activation}(\cdot)$ is the activation function. The main idea of an artificial neuron is to compute a weighted sum of the inputs, add a bias value to the weighted sum, and pass the result to a non-linear activation function to get the final output of this neuron. Weights and biases in an ANN can be trained to enable the ANN to perform some tasks. The application of an activation function is to introduce non-linearity to the ANN, because the activation of a biological neuron is non-linear, and most of the real-world patterns/signals are also non-linear. Some frequently-used activation functions are sigmoid (see Equation 2.2, e is the Euler’s number), hyperbolic tangent (or \tanh , see Equation 2.3), and rectified linear unit function (or ReLU [91], see Equation 2.4).

$$\textit{sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (2.2)$$

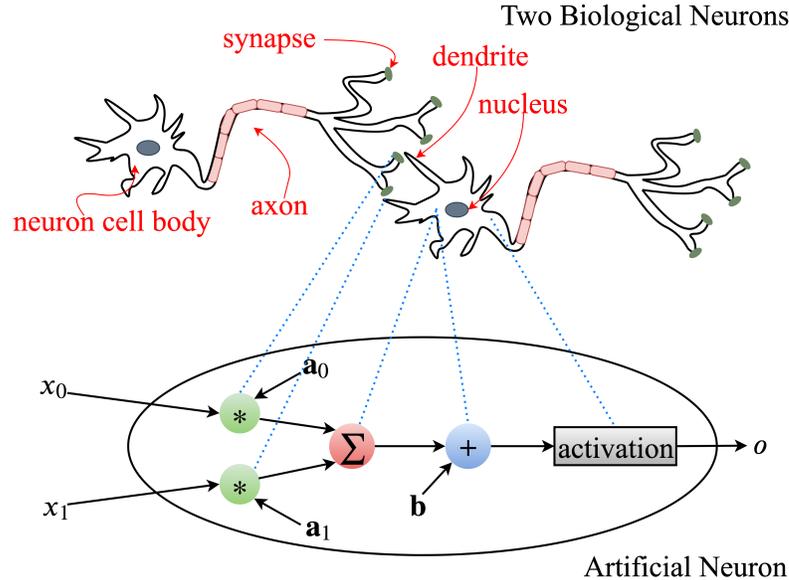
$$\textit{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.3)$$

$$\textit{ReLU}(x) = \max(0, x) \quad (2.4)$$

In the rest of this section, we review two types of ANNs and some algorithms for training ANNs.

2.5.1 Multi-Layer Feed-Forward Network

The multi-layer feed-forward (MLF) networks are universal function approximators with one or more hidden layers [47]. The term “layer” here refers to a group of artificial neurons, where the neurons in the same layer have no direct connections with each other. The architecture of an MLF network can be represented by a digraph,

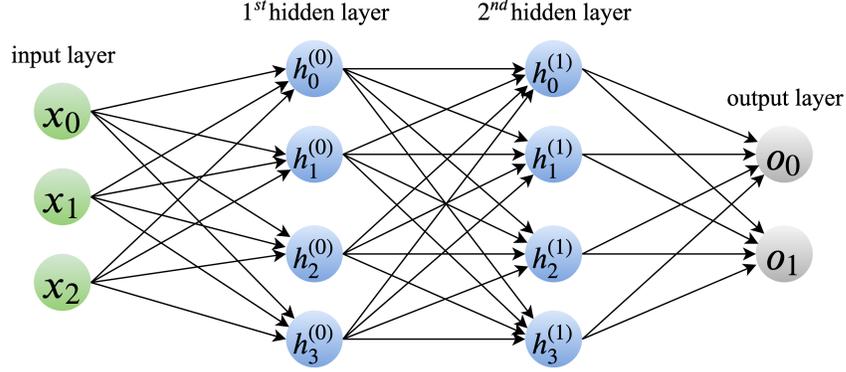


© Peizhi Yan

Figure 2.3: Biological neurons and artificial neuron. Dashed lines represent a conceptual match between the biological neuron and artificial neuron but not a reflection of how a biological neuron works.

where the vertices represent neurons, and the edges represent the pass of signal. Different from recurrent/feedback neural networks, an MLF network graph has no loop. Besides the hidden layer(s), an MLF network has one input layer (the first layer) and one output layer (the last layer). The input layer has no weights, and it does no computation. It is like a placeholder to pass the input signal to the next layer (the first hidden layer). The purpose of using multiple layers (with non-linear activation) in an MLF network is to enable the MLF network to learn non-linear features/patterns. Figure 2.4 depicts an MLF network with two hidden layers; each hidden layer has four neurons; the input and output layers have three and two neurons, respectively.

Each layer (except the output layer) of an MLF network can only pass the signal to the next layer, which is known as feed-forward. Two nearest layers are fully connected, which means that each neuron in the previous layer passes the signal to every neuron in the next layer. We use vector $\vec{x} = \{x_0, x_1, \dots, x_p\} \in \mathbb{R}^{1 \times p}$ to represent the input p -dimensional signal, use vector $\vec{o} = \{o_0, o_1, \dots, o_q\} \in \mathbb{R}^{1 \times q}$ to represent the output q -dimensional signal. Suppose the MLF network has L hidden layers. The vector $\vec{h}^{(l-1)} = \{h_0^{(l-1)}, h_1^{(l-1)}, \dots, h_{nn^{(l-1)}}^{(l-1)}\} \in \mathbb{R}^{1 \times nn^{(l-1)}}$ represents the output of the



© Peizhi Yan

Figure 2.4: The digraph-representation of an example MLF network.

Layer	Output of this layer
Input layer	$\vec{x} = \{x_0, x_1, \dots, x_p\}$
l^{st} hidden layer ($l = 1$)	$\vec{h}^{(0)} = \text{activation}(\vec{x}\mathcal{W}^{(0)} + \vec{b}^{(0)})$
l^{st} hidden layer ($l > 1$)	$\vec{h}^{(l-1)} = \text{activation}(\vec{h}^{(l-2)}\mathcal{W}^{(l-1)} + \vec{b}^{(l-1)})$
Output layer	$\vec{o} = \text{activation}(\vec{h}^{(L-1)}\mathcal{W}^{(L)} + \vec{b}^{(L)})$

The choice of activation function for each layer could be different.

Table 2.1: Signal Propagation in an MLF Network

l^{th} hidden layer ($1 \leq l \leq L, l \in \mathbb{Z}_+$), where $nn(l-1)$ is the number of neurons in the l^{th} layer. We use two-dimensional matrix \mathcal{W} to represent the layer weights. The weights of the l^{th} hidden layer is $\mathcal{W}^{(l-1)} \in \mathbb{R}^{nn(l-2) \times nn(l-1)}$, except for the first hidden layer $\mathcal{W}^{(0)} \in \mathbb{R}^{p \times nn(0)}$. The output layer weights is $\mathcal{W}^{(L)} \in \mathbb{R}^{nn(L-1) \times q}$. Similarly, we define the bias of the l^{th} hidden layer as a vector $\vec{b}^{(l-1)} \in \mathbb{R}^{1 \times nn(l-1)}$, and the output layer biases as a vector $\vec{b}^{(L)} \in \mathbb{R}^{1 \times q}$. Based on the definitions, we summarize the feed-forward propagation of each layer in an MLF network in Table 2.1. For convenience, we represent the MLF network as a function with input \vec{x} and the set of parameters θ (the collection of weights and biases): $F_{\theta}(\vec{x}) = \vec{o}$.

2.5.2 Graph Convolutional Network

Convolutional neural networks (CNNs) [70] are powerful tools in learning on the spatially structured data such as sound wave signals (1D data), pixel images (2D data),

and voxel representation of 3D models (3D data). Different from the densely connecting each input value to a neuron (usually seen in the MLF networks), the CNN utilizes the trainable kernels (filters) and the convolution operation to detect similar patterns on the signals. In summary, the primary features of a CNN are local connection, shared filter weights, and multi-layer deep feature extraction [71]. These features significantly reduce the number of neurons and dramatically improve the convergence speed of a CNN. Intuitively, we want to apply the same idea on graph structures, because a graph structure could contain repetitive and similar patterns. However, the graph structure belongs to the non-Euclidean data structure, and the traditional CNN does not work in the non-Euclidean domain [149]. To address this issue, Kipf et al. proposed the graph convolutional network (GCN), which is based on the idea of graph Fourier transform [67]. In this section, we summarize the GCN algorithm and review some applications of GCN in combinatorial optimization problems.

Mathematical Representation

Denote the undirected graph $G = (V, E)$. We can represent the graph G as an adjacency matrix (binary or weighted) $A \in \mathbb{R}^{|V| \times |V|}$. If A is a binary matrix, A_{ij} indicates whether there is an edge between vertex i and vertex j . For weighted matrix A , A_{ij} is the weight of the edge between vertex i and vertex j . The degree of each vertex is recorded in matrix $D \in \mathbb{R}^{|V| \times |V|}$ ($D_{ij} = 0$ if $i \neq j$), where $D_{ii} = \sum_j A_{ij}$ is the degree of vertex i . For graph G , the order of vertices could be different; A and D are dependent on the order of vertices. Therefore, there could be different adjacency matrices represent the same graph, which makes directly learning on A infeasible. Based on the theory that the convolution in one domain is equivalent to the point-wise production in the other domain, GCN leverages graph Fourier transform to enable the spectral graph convolution.

Define $\tilde{A} = A + I_{|V|}$, where $I_{|V|}$ is the identity matrix of shape $|V| \times |V|$. Correspondingly, define $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. Assume $X \in \mathbb{R}^{|V| \times C}$ is a signal. Each row in X represents a C -dimensional feature vector of the corresponding vertex. Denote $W \in \mathbb{R}^{C \times F}$ as the matrix of filter parameters (F represents the number of filters), the spectral graph convolution operation is defined in Equation 2.5:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X W, \quad (2.5)$$

where $Z \in \mathbb{R}^{|V| \times F}$ is the derived new signal; $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ is the normalized graph

Laplacian. We use $\tilde{S} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ to abbreviate the normalized graph Laplacian. We call \tilde{S} the support matrix.

In a GCN, there could be more than one graph convolution layers. Equation 2.6 represents the layer-wise propagation:

$$H^{(l+1)} = \text{activation}(\tilde{S}H^{(l)}W^{(l)}), \quad (2.6)$$

where l represents the l^{th} layer; $\text{activation}(\cdot)$ is the general representation of element-wise activation function, which could be $\text{ReLU}(\cdot)$ (see Equation 2.4), $\text{sigmoid}(\cdot)$ (see Equation 2.2), etc. In the first layer, $H^{(0)} = X$. The time complexity of GCN is $O(|E|)$.

Applications in Combinatorial Problems

One can modify the original GCN architecture to achieve different objectives. Some applications of GCN include graph-level classification, vertex-level classification or clustering, and link prediction [146]. Li et al. [78] propose a GCN-based heuristic function that treats the GCN output as a likelihood map over vertices and leverages the greedy tree search algorithm to derive the final solution. This approach increases the speed of tree search and achieves satisfactory performance on the maximum independent set problem (MWIS) [119]. However, there are some limitations to this approach. For instance, this approach cannot solve the maximal clique problem on large and dense network graphs. A similar approach was introduced in [56], which uses the GCN to predict the link likelihood map, and leverages the heuristic-based beam search to find the optimal route for traveling salesman problem. In [37], the researchers formulated the branch-and-bound scheme, which is a method for solving NP-hard mixed-integer linear programming problems, as a Markov decision process. They trained the GCN to imitate the branching policies. The experiment shows that the GCN trained on relatively small instances could generalize to larger instances.

In summary, the merit of applying GCN in combinatorial optimization problems is either reducing the optimization time or improving the local-optimal solution, or both. A typical way of using GCN in solving combinatorial optimization problems is to train the GCN to imitate the conventional algorithms. Because GCN adopts the idea of the convolution operation, theoretically, the input graph could be any size. This feature enables one to train the GCN on small instances and generalize it on large instances (computationally prohibitive to conventional algorithms) later.

2.5.3 Learning the Weights: Gradient Descent with Backpropagation

Different ANN architectures might require different weights learning (or training) algorithms. Some famous learning algorithms are perceptron learning algorithm [112] (for training a perceptron), support vector machine algorithm [107] (for training a support vector machine), extreme learning machine algorithm [49] (for training an extreme learning machine), and gradient descent-based algorithms (capable of training a wide variety of ANNs). In this subsection, we focus on the gradient descent-based ANN optimization scheme with the error backpropagation technique.

The foundation of backpropagation (BP) could date back to the Henry J. Kelley’s gradient-based optimization technique (proposed in 1960) in the domain of control theory [61]. In 1974, Paul Werbos first proposed the BP algorithm [132, 133]. In 1986, David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams defined the term “backpropagation” and demonstrated its promising application in training ANNs [103]. In the recent decade, the studies on deep learning further proved the effectiveness and robustness of BP-based ANN optimization algorithms in training deep ANNs [71]. The main idea of gradient descent with BP is to reduce the ANN prediction error (or loss) through back-propagating the error to tune the ANN weights layer-by-layer. Therefore, to train an ANN in this way, one needs to define a loss function to evaluate the difference between the ANN prediction and the desired output (also known as the target). We use the MLF network as an example to briefly explain the process of BP and gradient descent.

We represent the training data as a pair of input vector and the corresponding target vector (\vec{x}, \vec{y}) , where $\vec{y} \in \mathbb{R}^{1 \times q}$. According to $F_{\theta}(\vec{x}) = \vec{o}$, the objective is to tune the weights and biases in θ to let \vec{o} as close to \vec{y} as possible. We define the loss function $\mathcal{L}(\vec{o}, \vec{y})$ as a metric of the distance between \vec{o} and \vec{y} . For different problems, we often need to design different loss function to better guide the optimization algorithm. Some commonly used loss functions are squared error ($\mathcal{L}_{SE}(\vec{o}, \vec{y}) = \|\vec{o} - \vec{y}\|^2$) and absolute error ($\mathcal{L}_{AE}(\vec{o}, \vec{y}) = \|\vec{o} - \vec{y}\|$). For classification problems, especially multi-class classification, cross-entropy loss function is highly recommended.

Given the randomly initialized network parameters θ , the gradient descent approach first calculate the partial derivatives of the loss function with respect to each parameter

$$\delta = \frac{\partial \mathcal{L}(F_{\theta}(\vec{x}), \vec{y})}{\partial \theta} = \frac{\partial \mathcal{L}(\vec{o}, \vec{y})}{\partial \theta}, \quad (2.7)$$

where δ is called the gradient. Then, based on the gradient, update the network parameters through

$$\theta_{new} = \theta - \alpha\delta, \quad (2.8)$$

where θ_{new} represents the updated parameters; α is called learning rate, to control the speed of training. We can repeat the training multiple times on the training dataset to ensure the loss converges to an acceptable local optimal.

However, Equation 2.7 and Equation 2.8 are generalized representations. In practice, we need to compute the gradient for each layer, and update the parameters respectively. We start the BP by computing

$$\frac{\partial \mathcal{L}(\vec{\sigma}, \vec{y})}{\partial \vec{\sigma}}. \quad (2.9)$$

The gradient of the output layer weights is derived through the chain rule

$$\delta_{\mathcal{W}}^{(L)} = \frac{\partial \mathcal{L}(\vec{\sigma}, \vec{y})}{\partial \mathcal{W}^{(L)}} = \underbrace{\frac{\partial \mathcal{L}(\vec{\sigma}, \vec{y})}{\partial \vec{\sigma}}}_{\text{term 1}} \cdot \frac{\partial \vec{\sigma}}{\partial \mathcal{W}^{(L)}}. \quad (2.10)$$

Similarly, for the previous layer, we can compute the gradient of its weights through

$$\delta_{\mathcal{W}}^{(L-1)} = \frac{\partial \mathcal{L}(\vec{\sigma}, \vec{y})}{\partial \mathcal{W}^{(L-1)}} = \underbrace{\frac{\partial \mathcal{L}(\vec{\sigma}, \vec{y})}{\partial \vec{\sigma}}}_{\text{term 1}} \cdot \underbrace{\frac{\partial \vec{\sigma}}{\partial h^{(L-1)}}}_{\text{term 2}} \cdot \frac{\partial h^{(L-1)}}{\partial \mathcal{W}^{(L-1)}}. \quad (2.11)$$

By calculating the gradient of layer weights in a backward order, we can reuse the previous calculation. In Equation 2.10, the “term 1” was derived from Equation 2.9. We can further use the previously calculated “term 1” in Equation 2.11. If there is the $(L - 1)^{th}$ hidden layer, we can reuse the “term 2” derived in Equation 2.11 in computing the $\delta_{\mathcal{W}}^{(L-2)}$. We can repeat this procedure to derive the gradient for the weights of all the layers. Similarly, we can also derive the gradient for the biases of all the layers. Based on the derived gradients, we can update the corresponding parameter through Equation 2.8.

2.6 Deep Reinforcement Learning

The idea of reinforcement learning came from the way an individual with intelligence interacts with the environment and learns from the feedback of the environment to achieve some objective [59]. In other words, reinforcement learning deals with learning sequential decision-making. In a typical reinforcement learning process, the agent (a computer program) repeatedly performs the series of steps: observing the environment, making the decision of action, completing the action, learning from the feedback of the environment. To simplify the learning process, one typically model the dynamic process of the agent interacts with the environment as a Markov decision process (MDP) [123]. The MDP is usually defined by a set of states \mathcal{S} ($s \in \mathcal{S}$); a set of actions \mathcal{A} ($a \in \mathcal{A}$); a probabilistic transition function $\mathbb{P}(s'|s, a)$, where s is the current state, a is the action regarding s , and s' is the expected next state; a reward function of a given state $R(s)$; an initial state s_0 , and a terminal state s_{end} if possible. The MDP assumes that the future state is independent of the past states given the present state (Markov property).

A model-free reinforcement learning algorithm Q-learning was proposed in [131] to find the optimal policy which maximizes the total reward for any finite state MDP. The Q-learning policy is dependent on a table of the Q-values of a finite number of state-action pairs, which is called Q-table ($Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$). To maximize the total reward, the Q-learning agent tends to choose an action a at a particular state s which has the largest Q-value ($\underset{a}{argmax} Q(s, a)$). In practice, we need the Q-learning agent to explore actions other than $\underset{a}{argmax} Q(s, a)$. To balance the exploration and the exploitation, we can apply an exploration rate ϵ ($0 \leq \epsilon \leq 1$) in the decision making process (the algorithm is named ϵ -greedy algorithm). Each time the agent is making a decision, it first generates a random value between 0 and 1. If the random value is greater than ϵ , then the agent chooses $\underset{a}{argmax} Q(s, a)$ as the action; otherwise, the agent chooses a random action from \mathcal{A} . The Q-table is randomly initialized before the Q-value updating process. The algorithm for updating the Q-table (to maximize the reward) is formulated as

$$Q(s, a) = Q(s, a) + \alpha^\diamond [r + \gamma \underset{a'}{max} Q(s', a') - Q(s, a)], \quad (2.12)$$

where s is the current state; a is the current action; r is the instant reward after performing a ; s' is the next state; a' is the next action; α^\diamond is the learning rate (different

from the gradient descent learning rate α); γ is the discount factor ($0 \leq \gamma \leq 1$, $\gamma = 0$ means the Q-learning agent only focuses on the instant reward).

Although Q-learning is an effective reinforcement learning algorithm, it is inefficient when the state-action space is huge. One solution is to combine similar states to reduce the size of state space by explicitly calculating the similarity between states [32]. Inspired by the predictive power of multi-layer ANNs on unseen data, researchers explored a way of using the ANN as Q-table, which is called deep Q-learning (DQL). Correspondingly, the artificial neural network used as the Q-table is called deep Q network (DQN) [88]. We denote the DQN parameters as θ , $\mathcal{Q}_\theta(s)$ represents the output of the DQN on input state s , and we have $\mathcal{Q}_\theta(s) = \bigcup_{a \in \mathcal{A}} \mathcal{Q}_\theta(s, a)$. For a transition of MDP (s, a, r, s') , the target Q-value is defined as

$$y(s, a) = r + \gamma \max_{a'} \mathcal{Q}_{\theta'}(s', a'), \quad (2.13)$$

where θ' is a history copy of θ to avoid oscillations during training. Different from (2.12), in DQL we need to use gradient descent-based optimizer to update the DQN parameters θ through minimizing the following loss function

$$\mathcal{L}_\theta = [y(s, a) - \mathcal{Q}_\theta(s, a)]^2. \quad (2.14)$$

Chapter 3

Dense RFID Network Collision Avoidance

Radio-frequency identification (RFID) is widespread in industries such as supply-chain management and logistics due to its low-cost feature. In many real-world problems, one often needs to leverage a considerable amount of RFID readers to cover a large area. We call this type of system a dense RFID network. Many graph-based dense RFID network anti-collision algorithms were proposed to address the collision problems. However, state-of-the-art collision avoidance algorithms are centralized algorithms. In a dense RFID network, the graphs generated by the centralized algorithms could be very complicated. Therefore, a centralized algorithm increases the computational workload of the central server. We propose a distributed anti-collision algorithm based on the idea of a centralized collision avoidance algorithm called MWISBAII [86]. We found that due to the lack of global information, there is a gap between the performance of our distributed algorithm and the centralized MWISBAII. To narrow this gap, we introduce machine learning into the proposed algorithm. The machine learning model is an empirical model that mitigates the deficiency of the lack of global information. The experimental results show that the proposed distributed algorithm with machine learning can get almost the same performance as the centralized MWISBAII under different experimental settings.

3.1	Introduction	21
3.2	Related Works	24

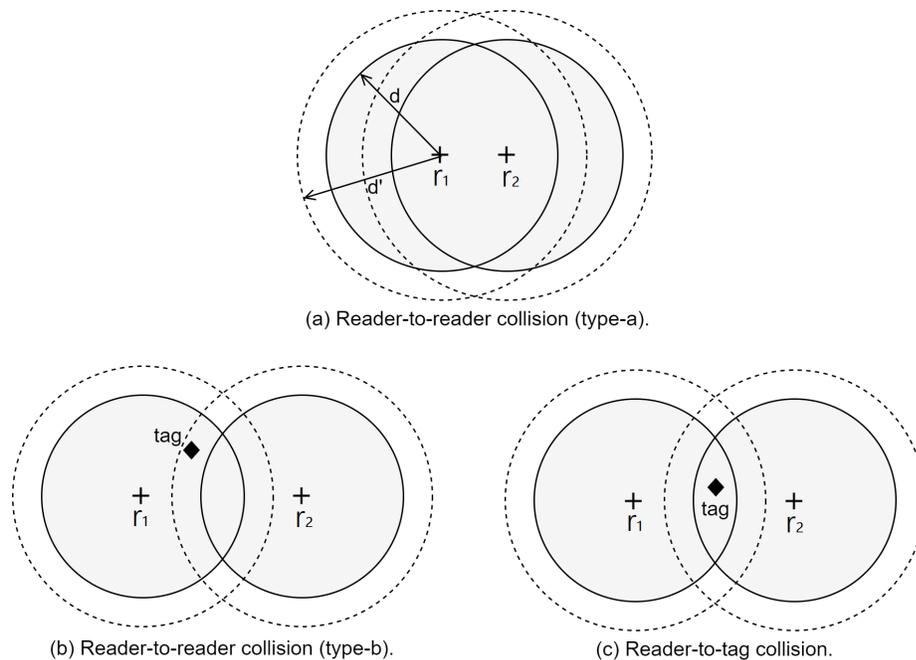
3.3	Problem Statement	26
3.4	MWIS Algorithms and MWISBAII	26
3.4.1	GWMIN2 Algorithm	27
3.4.2	MWISBAII	28
3.5	The Distributed MWISBAII	29
3.6	Machine Learning Auxiliary Approach	33
3.6.1	Machine Learning Stage	34
3.6.2	Application Stage	36
3.7	Experimental Results	38
3.8	Summary	42

3.1 Introduction

Since the RFID reader provides energy for passive RFID tags, and the valid energy transmission range is small, the coverage of a single RFID reader is limited. Therefore, in many real-world applications, one generally uses multiple readers to increase the coverage of the RFID system [9, 10, 11, 90]. This kind of RFID systems are referred to as dense RFID readers systems. Tags within the activated interrogation range of a reader can be read by the system if there is no collision. In reality, since the electromagnetic wave will not disappear beyond the interrogation range, there is an interference range, which is larger than the interrogation range [63]. In Figure 3.1 (a), for the reader r_1 , the radii of the interrogation range and the interference range are denoted by d and d' , respectively. We can use a coefficient β to represent the relationship between d and d' : $d' = d\beta$ ($\beta > 1.0$). The value of β can be measured through the experiment [64]. Whereas, in our work, the model of interrogation and interference ranges satisfies ideal assumptions. For instance, the interference signals of multiple RFID readers will not accumulate, and the wireless transmission is in free-space. We define the interference region as the region which is within the interference range but beyond the interrogation range. If we only deploy the RFID readers and tags on a two-dimensional plane, then we can abstract the interrogation and interference range to circles. To get full coverage of a field, the ranges of different readers may overlap (as shown in Figure 3.1), which may lead to some types of collisions.

Reader-to-reader collision (frequency interference) and reader-to-tag collision (tag interference) are two primary types of collisions in a dense RFID readers system [26,

34, 60, 144, 148]. There are two types of reader-to-reader collisions. Type-a reader-to-reader collision (see Figure 3.1a) occurs when a reader is within the interrogation range of another reader, and both readers are active. The radio-frequency electromagnetic wave emitted by the second reader prevents the first reader from communicating with tags within its interrogation range. Figure 3.1b depicts type-b reader-to-reader collision, which occurs when a tag is in the interrogation range of one reader (r_1), but also in the interference region of another reader (r_2). If r_1 wants to read the tag and r_2 is also active, then the signal of r_2 may interfere with the signal of the tag; meaning, r_1 may not be able to read the tag. Reader-to-tag collision occurs when one or more tags are in the activated interrogation ranges of more than one readers (see Figure 3.1c). In this example, if r_1 and r_2 attempt to communicate with the tag simultaneously, the reader-to-tag collision will occur. Furthermore, if the number of tags within an RFID reader's interrogation range is greater than the maximum number of tags that can be read by an RFID reader (we use *limit* to represent this upper bound), this RFID reader cannot be activated. This scenario can also be considered as a type of collision.



© Peizhi Yan

Figure 3.1: Some types of collisions in a dense RFID readers system.

In some scenarios, passive RFID tags not only serve as the object identifiers but

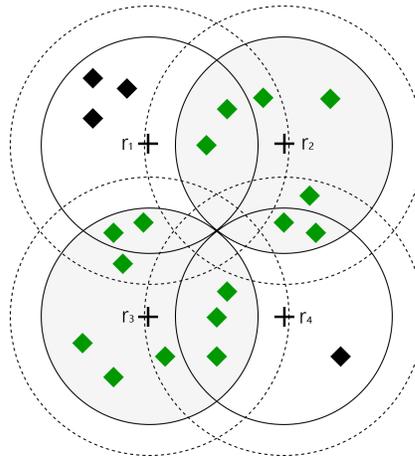


Figure 3.2: An example RFID system with four readers (r_1 , r_2 , r_3 , and r_4). r_2 and r_3 are active, while r_1 and r_4 are off. The rhombuses represent tags, where green indicates *can be read by the system*, and black indicates *cannot be read by the system*. The dashed circle represents the rim of the interference range, and the solid circle represents the rim of the interrogation range.

also have some complex functionalities. For instance, the wireless sensor could be integrated into an RFID tag, which leverages the energy harvested by the tag's antenna to drive the wireless sensor module and transmit the data collected by the sensor module [31, 145]. Because the wireless sensors need to work uninterruptedly, one needs to sacrifice some sensor nodes (deactivating some of the RFID readers could help to avoid the collision, but some RFID tags might be inaccessible by the system), and allow the whole system to enable as many sensor nodes online as possible. The problem of selectively activate or deactivate the interrogation ranges in a dense RFID readers system to allow the system to read as many tags at the same time as possible is known as reader-coverage collision avoidance (RCCA) problem [79]. Figure 3.2 depicts an example RFID system, where only two readers are active to enable the system to keep communicating with the maximum number of tags. One of the state-of-the-art RCCA algorithms is MWISBAII [86]. This algorithm first transforms the RCCA problem into a Maximum-Weight-Independent-Set (MWIS) problem and uses graph theory to solve the MWIS problem. The objective of the MWIS problem is to find a subset of graph vertices, where the vertices have no direct connections with each other in the graph, and the sum of their weights is as large as possible. In a dense RFID reader system, the graph representation of the MWIS problem could be huge,

increasing the burden on the central computer. We first present our initial distributed version of the MWISBAII in [138], which enables each reader to be involved in the decision-making process. To keep the performance of the centralized MWISBAII as much as possible, we further propose a machine learning assisted distributed RCCA algorithm [139]. We leverage the centralized MWISBAII to label the training data for machine learning and apply the trained model to our initial algorithm. We call the new algorithm a distributed MWISBAII with machine learning (DMWISBAII w/ ML). DMWISBAII w/ ML first utilizes the trained model to predict whether to activate and deactivate some of the readers and then use our initial algorithm to handle the rest of the readers. To the best of our knowledge, no machine learning-based approach has been proposed to solve the RCCA problem.

The remainder of this chapter is organized as follows. Section 3.2 provides a review of the existing dense RFID system anti-collision algorithms. We state the RCCA problem in detail in Section 3.3. In Section 3.4, we review a few algorithms that solve the MWIS problem and the MWISBAII, which is based on MWIS. We introduce our distributed MWISBAII algorithm in Section 3.5. Section 3.6 presents the machine learning auxiliary approach and gives an example of using this algorithm in solving the RCCA problem. Section 3.7 presents the experimental results of the performance of MWISBAII and our distributed algorithm (with or without machine learning auxiliary). In the end (in Section 3.8), we conclude this work and put forward some ideas for future work.

3.2 Related Works

The objective of most dense RFID readers system collision avoidance algorithms is to minimize the total time used for identifying (reading) all the tags without collision. Alternatively, to increase the read throughput (generally defined as the number of tags read per time slot). Based on access schemes, such algorithms are usually classified into time-division multiple access (TDMA), frequency-division multiple access (FDMA), and carrier-sense multiple access (CSMA) [45, 57, 85, 98, 105, 143]. Considering TDMA has a relatively low implementation complexity and operational cost, most of such algorithms are TDMA-based, which can be further divided into ALOHA-based and tree-based algorithms [114]. The basic idea of access scheme-based algorithms is to reduce or eliminate collision by optimally allocating temporal or frequency resources. Rezaie et al. [98] propose a centralized reader-to-reader col-

lision avoidance protocol which combines TDMA and FDMA mechanisms. Ho et al. [45] propose a distributed hierarchical Q-learning (HiQ) algorithm for minimizing the collision rate of a dense RFID readers system. HiQ makes the optimization by assigning different time and frequency resources to RFID readers. However, the objective of HiQ is to reduce the collision but not guarantee the elimination of collision. Moreover, it is not efficient to train when the network size is large [113]. A CSMA-based collision avoidance algorithm (named GENTLE) for mobile RFID networks is proposed in [143]. GENTLE assumes that the reader-to-reader collision problem is more severe than the reader-to-tag collision problem in a mobile RFID network (the RFID readers could be mobile phones). The basic idea of GENTLE is to use beacon messages to eliminate reader-to-tag collision and use the multi-channel solution to avoid the reader-to-reader collision. Su et al. [115] propose a tree splitting-based anti-collision algorithm for ultra-high frequency RFID systems. This algorithm accelerates the splitting process as well as increases the system read throughput. A dense RFID network anti-collision protocol stack named Season is proposed in [140]. Season does not assume the existence of the interference range, which may lead to a different result than the theoretical expectation in practice. Season utilizes one phase (at the beginning) to collect data from the tags which are not within the overlapping interrogation ranges of different readers. After the first phase, Season converts the anti-collision problem to the MWIS problem and employs two phases to selectively active some readers. Those two phases might be executed multiple iterations until all the tags have been read. The algorithm proposed in [151] requires a planned deployment of RFID readers, which makes it possible for the algorithm to get the accurate location information of each reader. Based on the interrogation and interference regions of RFID readers, this algorithm schedules the activation of readers to enable all the areas to be covered at least once at the end. Zhu et al. [152] presents a distributed approach (ADRA) for the scenarios when a central server does not exist. ADRA assumes that there could be multiple applications running in the system and issue identification requests. Based on the assumption, ADRA works in an adaptive way to make the idle readers not to participate in the coordination. A common limitation of the algorithms, as mentioned earlier, is that the readers cannot read tags within their overlapped interrogation ranges simultaneously, which causes delays. MRTI-BT addressed this issue through bit tracking [29]. Besides, MRTI-BT also prevents common tags from being identified multiple times by different readers.

3.3 Problem Statement

Reader-coverage collision avoidance (RCCA) problem [79] is about which reader(s) in a dense RFID readers system should be activated to allow the whole system read as many RFID tags without collision as possible at the same time. It is a combinatorial problem and is NP-hard [79]. We suppose that the readers in the dense RFID readers system have identical technical specification such as the radius of the interrogation range d ; the radius of the interference range $d' = d\beta$ ($\beta > 1.0$); and the maximum number of tags can be read by each reader (denoted by *limit*). Assume the dense RFID network has N readers and M tags, then we define the set of readers as $\mathfrak{R} = \{r_i | 1 \leq i \leq N, i \in \mathbb{N}\}$, and the set of tags as $\mathfrak{T} = \{t_i | 1 \leq i \leq M, i \in \mathbb{N}\}$. For the i^{th} reader r_i , we use $T_{r_i} \subseteq \mathfrak{T}$ and $T'_{r_i} \subseteq \mathfrak{T}$ to represent the set of tags within its interrogation range and the set of tags within its interference range respectively. Because d' is greater than d , $T_{r_i} \subseteq T'_{r_i}$. $|T_{r_i}|$ is the number of tags in T_{r_i} , and $|T'_{r_i}|$ is the number of tags in T'_{r_i} . The result of the RCCA algorithm can be represented as a subset $R \subseteq \mathfrak{R}$. Only the readers in R should be activated. The result should meet the following constraints:

1. if r_i and $r_j \in R$, $i \neq j$, then $T_{r_i} \cap T'_{r_j} = \emptyset$ and $T'_{r_i} \cap T_{r_j} = \emptyset$;
2. if $r_i \in R$, then $|T_{r_i}| \leq \textit{limit}$ and $|T_{r_i}| > 0$.

We define that $T = \{T_{r_i} : r_i \in R\}$, which is the set of tags can be read by the RFID system. The goal of the RCCA algorithm is to find a set R , such that $|T|$ is as large as possible. In practice, we also consider the total energy consumption of the system. Therefore, besides maximizing $|T|$, we also want to minimize the number of activated readers $|R|$. An efficient RCCA algorithm should also derive a solution with a high T/R ratio (the number of readable tags divided by the number of activated readers): $|T|/|R|$.

3.4 MWIS Algorithms and MWISBAII

The purpose of the MWIS problem is to find a set of vertices for any given undirected graph $G = (V, E)$, where no two vertices are adjacent in the original undirected graph and the total weight of vertices should be as large as possible. Since the MWIS problem is NP-hard [36], when the graph is complex, often we can only derive a near optimal solution.

A few simple, yet effective greedy algorithms for solving the MWIS problem are GMIN [27], GMAX [38], and GWMIN2 [106]. The GMIN algorithm repeats the following process until no vertex can be selected (the graph is empty): select a vertex of the minimum degree from the graph and put it into the set \mathcal{I} (\mathcal{I} is an empty set at the beginning of the algorithm), then remove this vertex and its neighbors. The GMAX algorithm deletes a vertex of the maximum degree at each step until no vertex can be deleted (no edge in the graph), then puts all the remaining vertices into the set \mathcal{I} . The result set \mathcal{I} is the MWIS. Sakai et al. [106] show that both GMIN and GMAX can give a MWIS where the total weight is greater than or equal to $\sum_{v_i \in V} w(v_i) / (\deg(v_i) + 1)$, while GWMIN2 can give a MWIS that the total weight is greater than or equal to $\sum_{v_i \in V} w(v_i)^2 / \sum_{v_j \in \text{neighbors}(G, v_i)} w(v_j)$.

Du et al. [22] propose a distributed MWIS algorithm. Their algorithm allows each node to make a partial solution, where each node broadcasts the partial solution as a message to each of its neighbors. To achieve the different trade-off between approximation accuracy and space complexity, they introduced a parameter \mathfrak{h} to lead the nodes to truncate some partial solutions before broadcasting the message. The higher the \mathfrak{h} value, the more accurate approximation their algorithm can achieve. When $\mathfrak{h} = +\infty$, the nodes will not truncate any partial solution. The problem of this algorithm is when the number of nodes is huge, the message size could be exponentially large.

3.4.1 GWMIN2 Algorithm

To find the MWIS on a given undirected graph $G = (V, E)$, the first step in GWMIN2 algorithm is to evaluate the cost of each vertex $v_i \in V$ (Equation 3.1).

$$\text{cost}(G, v_i) = \frac{w(v_i)}{\sum_{v_j \in \text{neighbors}(G, v_i)} w(v_j) + w(v_i)} \quad (3.1)$$

The second step in GWMIN2 is to pick the vertex v_i with the highest cost and then add v_i into the independent set \mathcal{I} . Finally, delete v_i and its neighbors from G , and repeat the first and the second steps until no vertex can be selected. Set \mathcal{I} is the solution.

In a simple example MWIS problem depicted in Figure 3.3, there are four vertices $\{v_1, v_2, v_3, v_4\}$ with weights $\{7, 7, 9, 6\}$ respectively. We use Equation 3.1 to get the cost of each of them:

- $cost(G, v_1) = 7/23 \approx 0.3043478261$
- $cost(G, v_2) = 7/20 = 0.35$
- $cost(G, v_3) = 9/22 \approx 0.4090909091$
- $cost(G, v_4) = 3/11 \approx 0.2727272727$

Vertex v_3 has the highest cost value. Thus, we add v_3 to \mathcal{I} , and remove v_3 and its neighbors from G . By repeating the above steps, we get the MWIS: $\mathcal{I} = \{v_3, v_2\}$. The total weight of vertices in \mathcal{I} is 16. Because this example is straightforward, the result is the best possible solution. However, when the graph is large, GWMIN2 can only guarantee a relatively optimal solution.

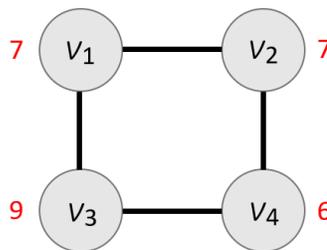


Figure 3.3: An example MWIS problem. Each circle represents a vertex; the red number beside each vertex represents the weight of that vertex.

3.4.2 MWISBAII

Maximum Weight Independent Set Based Algorithm (MWISBA) [79] ideally assumes that the interference range does not exist. Thereby, in nature, it cannot detect and avoid type-b reader-to-reader collisions. Based on MWISBA, MWISBAII [86] considers the interference range, which allows the RFID system to avoid all types of collisions depicted in Figure 3.1. The main idea of the MWISBAII is to transform the reader-coverage collision avoidance (RCCA) problem into a MWIS problem. The GEMIN2 algorithm is then used to solve this MWIS problem. Lastly, the solution of MWIS problem can be transformed back to the solution of RCCA problem.

For example (see Figure 3.4), there are four readers in a dense RFID reader system. Assume each reader only has one interrogation range, and the maximum number of tags that can be read by each reader is $limit = 10$. We use integer i to represent the i^{th} reader. In this example, $i \in \{1, 2, 3, 4\}$. To transform this RCCA problem into MWIS

problem, first, the MWISBAII will append a vertex v_i to graph G if the number of tags within the interrogation range of the i^{th} reader is less than or equal to $limit$. If there are any tags within both the interrogation range of the i^{th} reader and the interference/interrogation range of the j^{th} reader, an edge will be associated to v_i and v_j in G . Therefore, this RCCA problem will be transformed into the MWIS problem shown in Figure 3.3. The solution of the above MWIS problem is $\mathcal{I} = \{v_3, v_2\}$. This means that r_3 and r_1 should be activated to allow the RFID system to read as many tags as possible without collision.

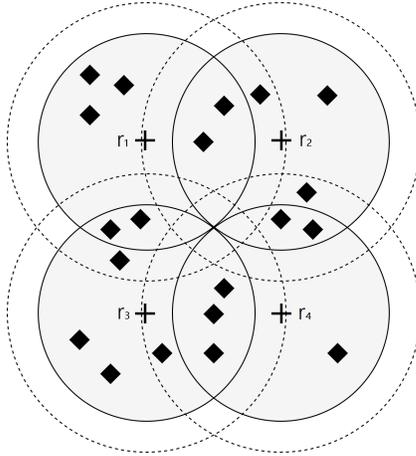


Figure 3.4: An example RFID system with four readers (r_1 , r_2 , r_3 , and r_4).

3.5 The Distributed MWISBAII

To allow each RFID reader to be involved in the computation and decision process, we propose a distributed MWISBAII [138]. In our distributed algorithm, we assume that each reader already has the information on how many tags within its interrogation and interference range (in practice, one could use RFID positioning technology to collect that information [130]). Besides, each reader should have a local data field that contains the following components:

- A local undirected graph (or local graph) $G = (V, E)$.
- The status of reader (STAT) that has four possible values:
 - LOCK: The reader will not receive signal from neighbor readers.
 - OPEN: The reader is waiting for signals from neighbor readers.
 - ACTIVE: The reader is activated.
 - OFF: The reader cannot be activated.

- A sender buffer BUFFER_{out} .
- A receiver buffer BUFFER_{in} .

Readers communicate with each other through signals, which we define as a six-tuple: $(i_-, j_-, \text{CODE}, \text{VALUE})$. Here, we assume that communication among readers will not interfere with the process of reading tags. The readers are using the communication range, which has a radius that is at least twice larger than d' , to ensure two readers can communicate with each other if there is any tag with each others' ranges that could cause a collision. The main idea of the distributed MWISBAIL is to let each reader build a local graph with the one-hop information. Then, let each reader compute and broadcast the cost. Readers make local decisions (whether to activate or not) regarding their local graphs. If a reader cannot decide on this iteration, it will move on to the next iteration until the decision been made.

The algorithm can be described as the following steps (suppose this reader is the i^{th} reader):

Step-1: Initialize the graph $G = (V, E)$. First, set STAT to LOCK. For the i^{th} reader, if $|T_{v_i}| \leq \text{limit}$, then we associate a vertex v_i in G . The weight of v_i is equivalent to $|T_{v_i}|$. We call v_i a local vertex. For all readers, other than the i^{th} reader ($\forall j \in \{j | 1 \leq j \leq N\}$): if $T'_{v_i} \cap T'_{v_j} \neq \emptyset$, we associate a vertex v_j (we call it non-local vertex) and an edge (v_i, v_j) to G . Go to step-3 (skip step-2).

Step-2: Remove all the redundant vertices in G (if the program just finished executing step 1, then this step will be skipped). For each non-local vertex $v_j (j \neq i, v_j \subseteq V)$, if the status (STAT) of the j^{th} reader is OFF, we simply remove v_j from G . If the status (STAT) of the j^{th} reader is ACTIVE, we remove v_j and its neighbors from G . Go to step-3.

Step-3: Compute the cost value of the local vertex. For each local vertex v_i , the cost value of it is calculated by Equation 3.1. Go to step-4.

Step-4: Prepare the signals to be sent. For each non-local vertex v_j in G , if there is an edge (v_i, v_j) between it and a local vertex, we create a signal $(i_- = i, j_- = j, \text{CODE} = \text{UPDATE}, \text{VALUE} = \text{cost}(G, v_i))$. Afterwards, we put this signal into the sender buffer BUFFER_{out} . Go to step-5.

Step-5: Send and receive signals alternatively. Set the status (STAT) to LOCK. For each signal $(i_-, j_-, \text{CODE}, \text{VALUE})$ in BUFFER_{out} ; if the i_-^{th} reader's status (STAT) is ACTIVE, we send this signal to the i_-^{th} reader (put into the i_-^{th} reader's BUFFER_{in}), and remove this signal from BUFFER_{out} . Change the status (STAT) to OPEN. If there are non-local vertices in G , wait for a short period of time to

receive signals. Repeat step 5 until $BUFFER_{out}$ is empty and the number of signals in $BUFFER_{in}$ equals the number of non-local vertices in G . Go to step-6.

Step-6: Process the signals in $BUFFER_{in}$. Set the status (STAT) to LOCK. For each signal $(i_-, j_-, CODE, VALUE)$ in $BUFFER_{in}$: if the $CODE = UPDATE$, we update the cost value of v_{i_-} to $VALUE$; else, if $CODE = ACTIVATED$, we remove the vertex v_{i_-} and the vertex v_{j_-} from G (just ignore it if v_{j_-} has already been removed); else ($CODE = DEACTIVATED$), we simply remove v_{i_-} from G . Go to Step-7.

Step-7: Make a local decision. If G is empty, change the status (STAT) to OFF, and for each non-local vertex v_j , send a signal $(i_- = i, j_- = j, CODE=DEACTIVATED, VALUE=N/A)$ to the j^{th} reader. Following this, stop the algorithm. If G is not empty, find the vertex with the highest cost value. If this vertex is a local vertex v_i : change the status (STAT) to ACTIVATE, and for each non-local vertex v_j , send a signal $(i_- = i, j_- = j, CODE=ACTIVATED, VALUE=N/A)$ to the j^{th} reader, and stop the algorithm. If the vertex with the highest cost value is not a local vertex, then go to step-2 (next iteration).

The deletion of vertices in a reader's local graph may cause the need for its neighbor readers to delete some of the vertices in their local graph. This may result in a domino effect. Due to the consideration of efficiency, we need to reduce the number of messages transmitted between readers. Therefore, in some cases, when a reader has made the decision, it will not send any signal to the other readers. This may finally cause a deadlock in the system [17]. To resolve the deadlock problem and assure a higher efficiency, we set a threshold of time limit for each reader. If the amount of time that a reader spends in any step is larger than the threshold, the reader will spontaneously set its status to OFF and stop the algorithm.

Each step in this algorithm has a time complexity of $O(|V|)$, and the time complexity of each iteration is also $O(|V|)$. Moreover, because each reader only needs to get its one-hop neighbor readers updated, the complexity of message exchange is $O(|V|)$. In Figure 3.5, we show the execution of the distributed MWISBAII on the example in Figure 3.4 step by step. After the first iteration, reader 2 (r_2) and reader 3 (r_3) are activated. In the second iteration (which is not shown in Figure 3.5), reader 1 (r_1) and reader 4 (r_4) will receive and process the signals sent by reader 2 and reader 3. At the end of the second iteration, reader 1 and reader 4 will be deactivated.

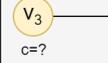
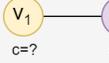
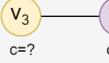
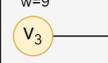
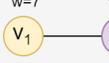
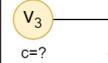
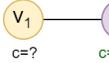
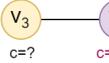
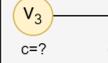
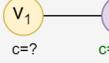
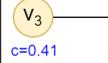
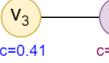
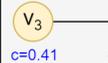
	<i>Reader 1</i>	<i>Reader 2</i>	<i>Reader 3</i>	<i>Reader 4</i>
Step-1	STAT: LOCK	STAT: LOCK	STAT: LOCK	STAT: LOCK
	G^* : w=9 w=7 w=7  c=? c=? c=?	G^* : w=7 w=7 w=6  c=? c=? c=?	G^* : w=7 w=9 w=6  c=? c=? c=?	G^* : w=9 w=6 w=7  c=? c=? c=?
	BUFFER _{out} :	BUFFER _{out} :	BUFFER _{out} :	BUFFER _{out} :
BUFFER _{in} :	BUFFER _{in} :	BUFFER _{in} :	BUFFER _{in} :	
Step-2	skip	skip	skip	skip
Step-3	STAT: LOCK	STAT: LOCK	STAT: LOCK	STAT: LOCK
	G^* : w=9 w=7 w=7  c=? c=0.30 c=?	G^* : w=7 w=7 w=6  c=? c=0.35 c=?	G^* : w=7 w=9 w=6  c=? c=0.41 c=?	G^* : w=9 w=6 w=7  c=? c=0.27 c=?
	BUFFER _{out} :	BUFFER _{out} :	BUFFER _{out} :	BUFFER _{out} :
BUFFER _{in} :	BUFFER _{in} :	BUFFER _{in} :	BUFFER _{in} :	
Step-4	STAT: LOCK	STAT: LOCK	STAT: LOCK	STAT: LOCK
	G^* : w=9 w=7 w=7  c=? c=0.30 c=?	G^* : w=7 w=7 w=6  c=? c=0.35 c=?	G^* : w=7 w=9 w=6  c=? c=0.41 c=?	G^* : w=9 w=6 w=7  c=? c=0.27 c=?
	BUFFER _{out} : (1, 3, UPDATE, 0.30) (1, 2, UPDATE, 0.30)	BUFFER _{out} : (2, 1, UPDATE, 0.35) (2, 4, UPDATE, 0.35)	BUFFER _{out} : (3, 1, UPDATE, 0.41) (3, 4, UPDATE, 0.41)	BUFFER _{out} : (4, 3, UPDATE, 0.27) (4, 2, UPDATE, 0.27)
BUFFER _{in} :	BUFFER _{in} :	BUFFER _{in} :	BUFFER _{in} :	
Step-5	STAT: OPEN	STAT: OPEN	STAT: OPEN	STAT: OPEN
	G^* : w=9 w=7 w=7  c=? c=0.30 c=?	G^* : w=7 w=7 w=6  c=? c=0.35 c=?	G^* : w=7 w=9 w=6  c=? c=0.41 c=?	G^* : w=9 w=6 w=7  c=? c=0.27 c=?
	BUFFER _{out} : (2, 1, UPDATE, 0.35) (3, 1, UPDATE, 0.41)	BUFFER _{in} : (1, 2, UPDATE, 0.30) (4, 2, UPDATE, 0.27)	BUFFER _{in} : (1, 3, UPDATE, 0.30) (4, 3, UPDATE, 0.27)	BUFFER _{in} : (2, 4, UPDATE, 0.35) (3, 4, UPDATE, 0.41)
BUFFER _{out} :	BUFFER _{out} :	BUFFER _{out} :	BUFFER _{out} :	
Step-6	STAT: LOCK	STAT: LOCK	STAT: LOCK	STAT: LOCK
	G^* : w=9 w=7 w=7  c=0.41 c=0.30 c=0.35	G^* : w=7 w=7 w=6  c=0.30 c=0.35 c=0.27	G^* : w=7 w=9 w=6  c=0.3 c=0.41 c=0.27	G^* : w=9 w=6 w=7  c=0.41 c=0.27 c=0.35
	BUFFER _{out} :	BUFFER _{out} :	BUFFER _{out} :	BUFFER _{out} :
BUFFER _{in} :	BUFFER _{in} :	BUFFER _{in} :	BUFFER _{in} :	
Step-7	STAT: LOCK	STAT: ACTIVE	STAT: ACTIVE	STAT: LOCK
	G^* : w=9 w=7 w=7  c=0.41 c=0.30 c=0.35	G^* : w=7 w=7 w=6  c=0.30 c=0.35 c=0.27	G^* : w=7 w=9 w=6  c=0.3 c=0.41 c=0.27	G^* : w=9 w=6 w=7  c=0.41 c=0.27 c=0.35
	BUFFER _{out} : (2, 1, ACTIVATED, N/A) (2, 4, ACTIVATED, N/A)	BUFFER _{out} : (3, 1, ACTIVATED, N/A) (3, 4, ACTIVATED, N/A)	BUFFER _{out} : (3, 1, ACTIVATED, N/A) (3, 4, ACTIVATED, N/A)	BUFFER _{out} :
BUFFER _{in} :	BUFFER _{in} :	BUFFER _{in} :	BUFFER _{in} :	

Figure 3.5: The first iteration by applying the distributed MWISBAIL algorithm on a simple example. “w” denotes vertex weight; “c” denotes vertex cost.

3.6 Machine Learning Auxiliary Approach

In practice, the tags are randomly distributed in the RFID system. Some RFID readers may have more tags, while other RFID readers may have fewer tags. This uneven distribution could be highly skewed. If the RFID readers merely make decisions on their local (1-hop) information, some valuable RFID readers (contribute more tags to the whole system) may not be successfully activated. We assume that there are hidden patterns that can help the distributed algorithm to make a better decision if the geological position of each RFID reader is fixed. This means that, with only the 1-hop local information and some empirical knowledge on the system environment, we could improve the performance of the distributed algorithm. Based on the assumption, we propose a machine learning auxiliary approach for our initial distributed MWISBAII algorithm. We use DMWISBAII w/o ML and DMWISBAII w/ ML to represent our initial algorithm and the algorithm with ML auxiliary, respectively.

To implement the proposed approach, we require that each RFID reader has a neural network model and a local ego-network graph. The RFID readers could communicate with their nearby readers through a wired connection. However, if the wireless connection among RFID readers is required, the communication should leverage a channel that will not interfere with the RFID interrogation. Moreover, the wireless communication range should be at least $d + d'$. Because when the distance between two RFID readers is greater than $d + d'$, the collisions we mentioned previously will not happen, and these two readers do not need to contact each other directly.

There are two stages in the proposed approach. The first stage is the machine learning stage, which is required only when the dense RFID system is set up. This stage primarily happens on the central server because running simulations and training a neural network model require high-performance computing resources. At the end of the machine learning stage, the central server broadcasts the trained neural network weights to each reader, and each reader updates its neural network model with the received weights. The second stage is the application stage, which runs in each reader. In this stage, each reader collects the information from its neighbor readers to initialize a local graph. The neural network model is used to score each reader afterward. A reader will be activated if it has the highest score among its neighbor readers. Finally, the initial distributed MWISBAII algorithm is used to arrange the rest of the readers. In this section, we describe the proposed approach in detail.

3.6.1 Machine Learning Stage

This stage has three sub-phases: (1) data collection, (2) training the neural network, and (3) broadcasting the trained weights. We run simulations in the data collection phase to collect training data. We assume the simulated area is $100\text{m} \times 100\text{m}$ (m is the unit meter), each reader has the identical specification (same *limit* and β), and the readers are uniformly distributed (see Figure 3.6). At the start of each simulation, we record the following information of each reader (r_i denotes the i^{th} reader): weight w_i , cost c_i , average weight of neighbors ϖ_i , and average cost of neighbors ι_i . Weight w_i denotes the number of tags within the i^{th} reader; cost c_i is computed through Equation. 3.1. Then, we run the MWISBAII to get the solution. The solution is recorded in Λ , where $\Lambda_i \in \{0, 1\}$ ($\Lambda_i = 0$ represents the i^{th} reader is not active; $\Lambda_i = 1$ represents the i^{th} reader is active;). Each sample is a quintuple: $(w_i, c_i, \varpi_i, \iota_i, \Lambda_i)$, where $(w_i, c_i, \varpi_i, \iota_i)$ is the input to the neural network, and Λ_i is the target output. If the number of positive samples ($\Lambda_i = 1$) and the number of negative samples ($\Lambda_i = 0$) are not equal, we randomly drop some samples from the majority group of samples to make the number of samples in both groups are equal.

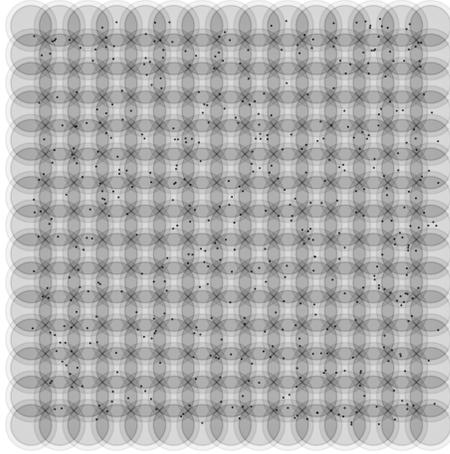


Figure 3.6: A $100\text{m} \times 100\text{m}$ simulated area with 500 randomly assigned tags (tags are shown as dots). The interval of nearest readers is 7m ; $d = 5\text{m}$; $\beta = 1.25$. A darker circle represents the interrogation range; and a lighter circle, which is slightly larger than that interrogation range, represents the interference range.

The neural network architecture we use is a fully-connected feed-forward network with two hidden layers. Each hidden layer has 16 neurons; the input layer has 4

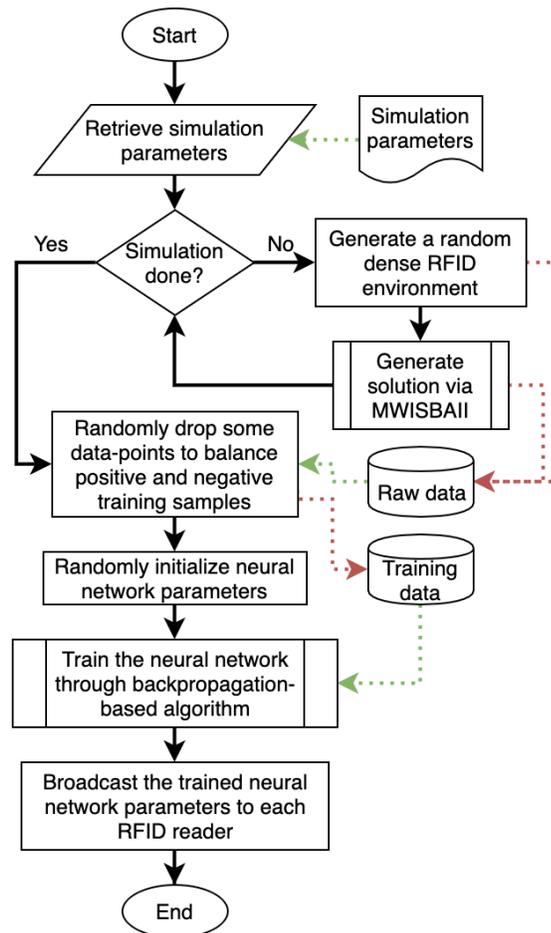


Figure 3.7: The flowchart of machine learning stage.

neurons which match the 4 input values $(w_i, c_i, \varpi_i, l_i)$; the output layer only has 1 neuron. The reason for using a two-hidden-layer architecture is that, with two hidden layers, the architecture can represent an arbitrary decision boundary [54]; whereas adding more hidden layers will significantly increase the computational complexity. The selection of the number of neurons in each hidden layer is based on the consideration among experimental performance, model complexity, and network generalization trade-off. Because the neural network weights can be represented as matrices, the total amount of weights of this neural network is $4 \times 16 + 16 \times 16 + 16 \times 1 = 336$ (not consider biases). We use 32-bit floating-point data type for neural network weights. Therefore the network weights take 10,752 bits (equivalent to 1,344 bytes) memory. We use ReLU as the activation function of the hidden layers, and use the sigmoid

activation function in the output layer. The neural network loss value is the mean squared error (MSE) of the neural network outputs and the target outputs. We use Adam optimizer [65] with a fixed learning rate (10^{-3}) to optimize the neural network (minimize the loss value). We train the neural network on the training samples 500 times (epochs), and the training batch size is 16 (each training step takes 16 samples). After training, we broadcast the trained neural network weights to each reader, and each reader assigns the received weights to its neural network model. Figure 3.7 depicts the flowchart of machine learning stage.

3.6.2 Application Stage

This stage has two sub-phases. In the first sub-phase, we first let each reader establish its local ego-network graph G . The center node (ego node) of the local graph represents the reader itself, where the other nodes (external nodes) represent the neighbor readers that have a conflict with this reader (if we activate this reader, all of its neighbor readers should not be activated; otherwise, if any of its neighbor readers is activated, this reader cannot be activated). Each node in the local graph should contain the following information (i denotes the i^{th} reader): weight w_i , cost c_i , score \mathfrak{s}_i , and status Θ_i . The score \mathfrak{s}_i is initialized to 0 and will be generated by the neural network once this reader has all the input information ready. The status Θ_i is an indicator of the corresponding reader's status (STAT). $\Theta_i = -1$ represents the STAT of the i^{th} reader is either LOCK or OPEN; $\Theta_i = 0$ or 1 represent the STAT of the i^{th} reader is OFF or ACTIVE, respectively. If w_i is either 0 or greater than $limit$, the i^{th} reader should be deactivated. Here, we skip the detail of the communication between readers. We can run our previously proposed distributed algorithm up to step-6 (include step-6) to complete the node cost information in the local graph.

Based on the information (w and c of each node) stored in G , each reader computes ϖ_i and ι_i . Then, each reader (reader i) inputs $(w_i, c_i, \varpi_i, \iota_i)$ to its neural network, and use the neural network output as its score \mathfrak{s}_i . Once a reader updated the score of all the graph nodes (both the ego node and the external nodes), this reader will find the node that has the highest score in its local graph. If the node with the highest score is the ego node, this reader is activated (set Θ_i to 1). If any neighbor readers of a reader are activated before this reader, this reader should be deactivated (set Θ_i to 0). If there are more than one nodes (include the ego node) have the highest score, or the only node with the highest score is an external node, this reader will enter

the second sub-phase which leverages the algorithm proposed in [138] to let the rest of the readers whose STAT is neither OFF nor ACTIVE ($\Theta = -1$) make decision. Figure 3.8 depicts the flowchart of application stage.

The time complexity (each iteration) of the previously proposed distributed algorithm is $O(|V|)$. Thereby, the time complexity for initializing G in the application stage is $O(|V|)$. The time complexity for computing either ϖ_i or ι_i is also $O(|V|)$, because there are $|V| - 1$ external nodes in G on average. The neural network has a fixed number of parameters and operations. Thus, the time complexity for computing s_i is $O(1)$. In summary, the time complexity of the application stage is equivalent to the previously proposed distribute algorithm.

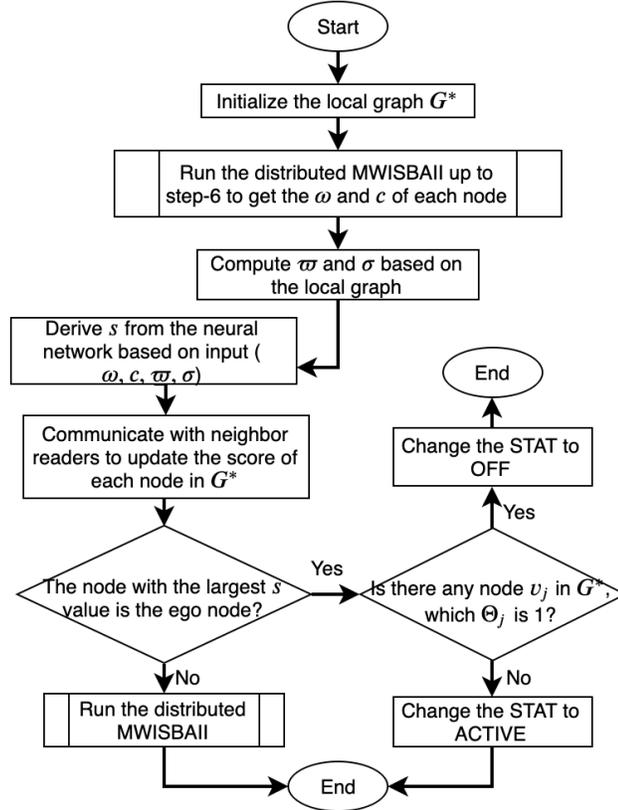


Figure 3.8: The flowchart of application stage.

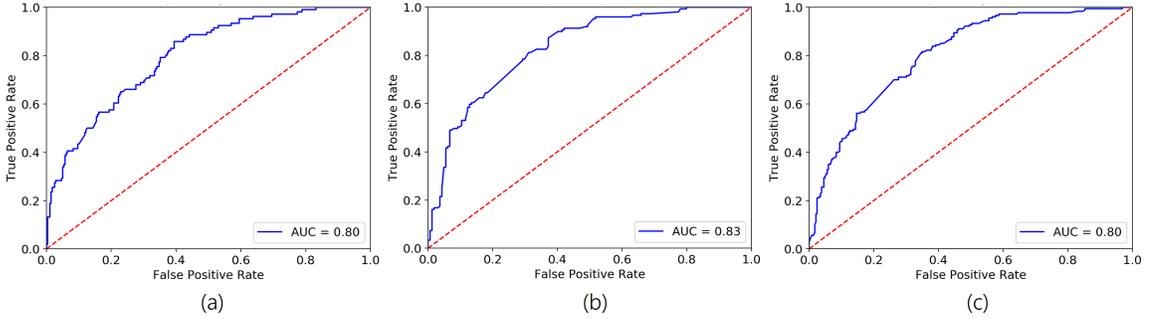


Figure 3.9: The testing ROC curves on setting 2 (a), setting 5 (b), and setting 8 (c).

3.7 Experimental Results

In the experiments, we simulated a square area ($100\text{m} \times 100\text{m}$) to deploy the readers and tags. We assume the interrogation range d of each reader is 5m ; the interference range d' of each reader is $d' = d\beta$, where β can be 1.15 , 1.25 , or 1.35 . Each reader can read at most 10 tags ($limit = 10$). The positions of tags are randomly generated. The readers are uniformly assigned. The interval (horizontally and vertically) between two nearest readers can be 6m , 7m , or 8m . Figure 3.6 depicts an example simulated area where the number of tags is 500 , the reader interval is 7m , and β is 1.25 .

Table 3.1 depicts the settings of the simulations for training the neural network. To collect data with more variety, for each setting, we simulate ten times for each number of tags in $\{100, 200, 300, 400, 500\}$. Therefore, in the later experiments of evaluating the performance of the proposed algorithm, the number of deployed tags can be from 100 to 500 . We randomly sampled 60% of the collected data points as training data, 20% data points as the validation data, and use the other 20% data points as testing data. Figure 3.9 shows the testing receiver operating characteristic (ROC) curves under setting 2, 5, and 8. In all of the experiments, the area under the curve (AUC) is greater than or equal to 0.8 .

To get some visual insights into the simulation, under simulation setting 5, we apply k-means clustering algorithm on the readers ($k = 2, 3, 4$). Note well, each reader has the average w_i , c_i , ϖ_i , and ι_i over simulations as its fingerprint. When $k = 2$, as shown in Figure 3.10 (the leftmost sub-figure), the readers at the edge are classified into one cluster. The reason is that in our simulations, the readers at the edge has a different number of neighbors than the other readers. When $k = 3$ or $k = 4$, the pattern of clusters is not obvious. Because theoretically, under our simulation setting, only the readers at the edge and the readers surrounded by the

readers at the edge have an evident difference.

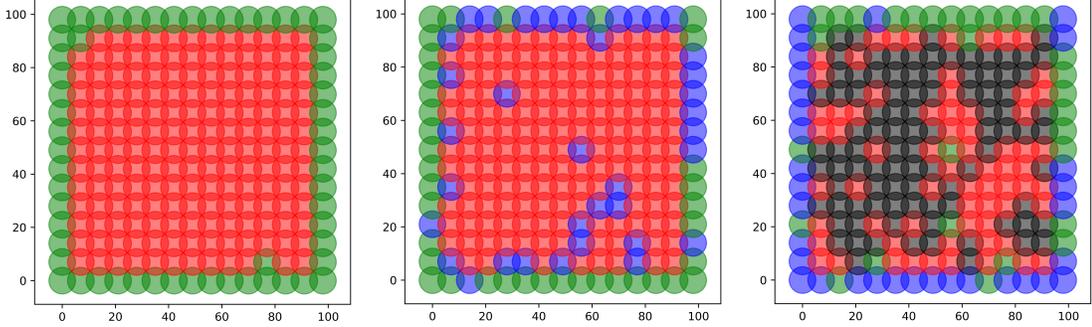


Figure 3.10: Readers clustered by k-means clustering algorithm. From left to right, the number of clusters (k) are two, three, and four. Each color represents a cluster. The circle represents the interference range.

In the performance evaluation experiments, we use all the simulation settings in Table 3.1. For each setting, we ran the centralized MWISBAII, the proposed machine learning auxiliary approach (DMWISBAII w/ ML), and the distributed MWISBAII algorithm without machine learning assistance (DMWISBAII w/o ML) separately. The number of deployed tags is from set $\{100, 150, 200, 250, 300, 350, 400, 450, 500\}$, for each, we ran the simulation ten times and show the average results in Figure 3.11, Figure 3.12, and Figure 3.13. The evaluation metrics are the number of tags can be read by the RFID system and the T/R ratio. From the experiments, we can see that the proposed algorithm with machine learning assistance can get almost the same performance as the centralized MWISBAII. Besides, the proposed algorithm with machine learning assistance is always better than the one without machine learning assistance.

Setting	1	2	3	4	5	6	7	8	9
Interval	6m	6m	6m	7m	7m	7m	8m	8m	8m
β	1.15	1.25	1.35	1.15	1.25	1.35	1.15	1.25	1.35
Number of tags	{100, 200, 300, 400, 500}								

Table 3.1: Simulation Settings for Training

An interesting phenomenon in the performance evaluation results is that the number of tags that can be read by the system is less than the number of deployed tags. Also, with the number of deployed tags increases, the number of tags can be read increases slower. The explanation is that since each reader can read at most 10 tags

($limit = 10$), once the number of deployed tags within the interrogation range of a reader is greater than 10, this reader cannot be activated (this situation can be seen as a type of collision), the system might fail to read those tags.

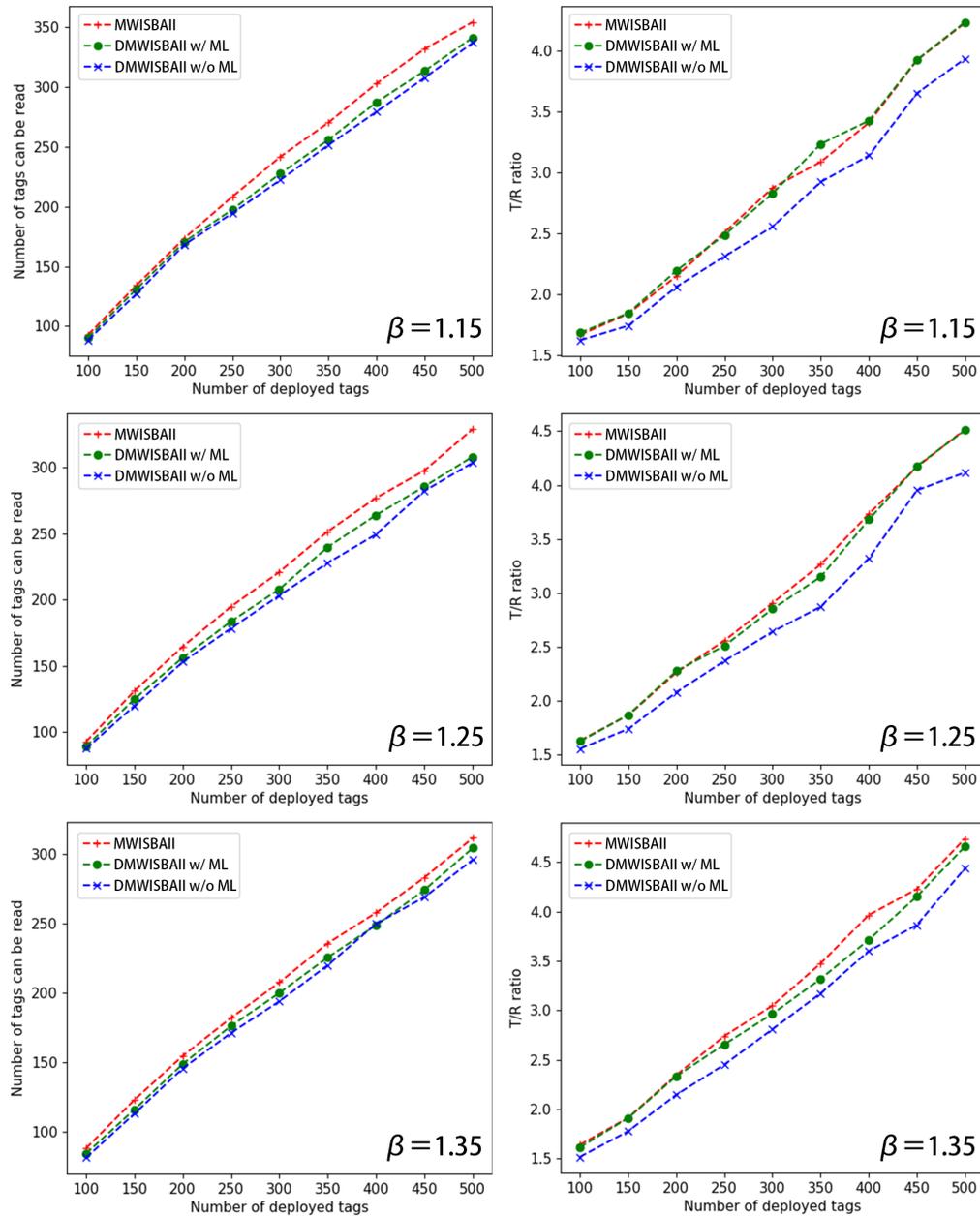


Figure 3.11: Performance evaluation results when reader interval is $6m$.

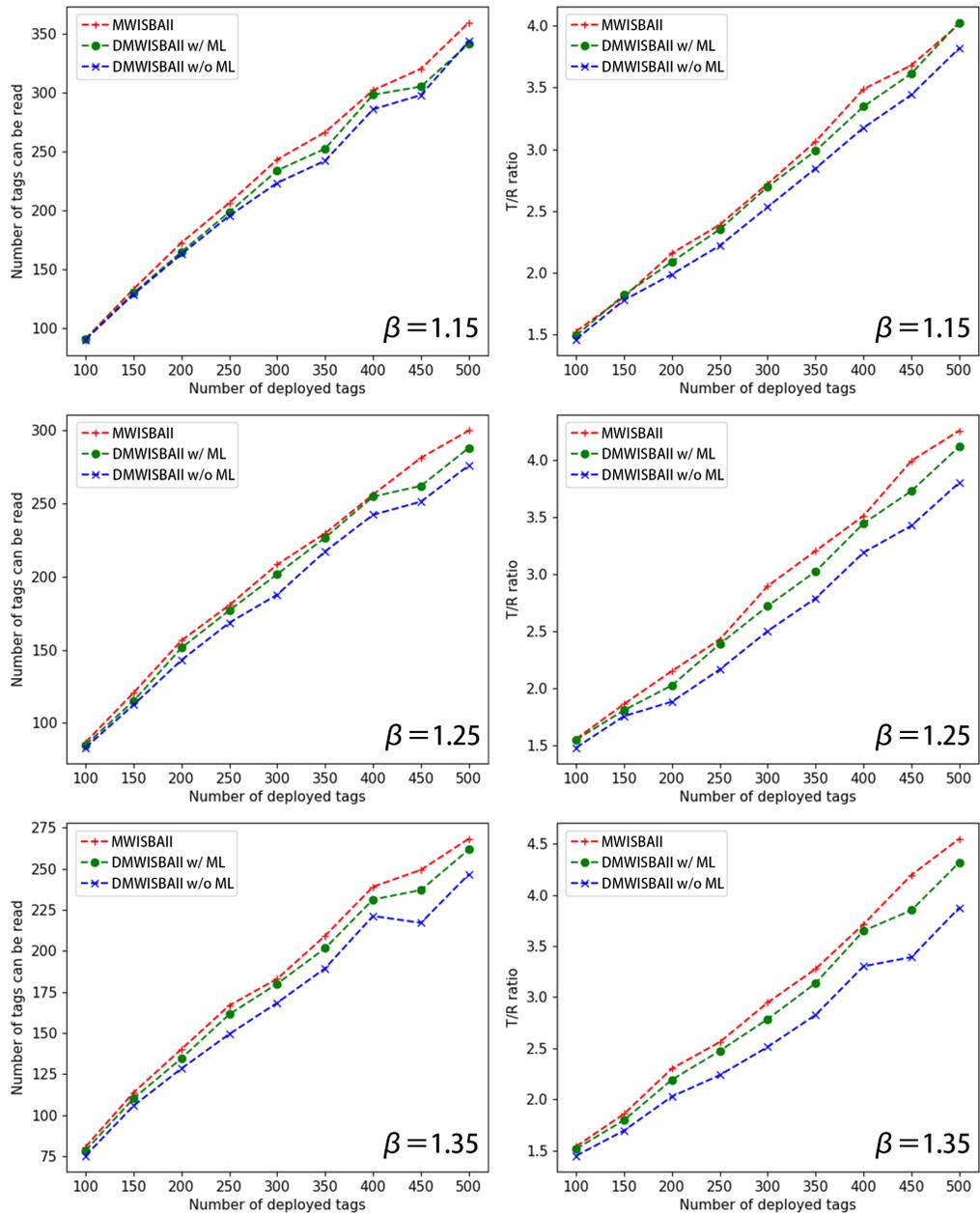


Figure 3.12: Performance evaluation results when reader interval is $7m$.

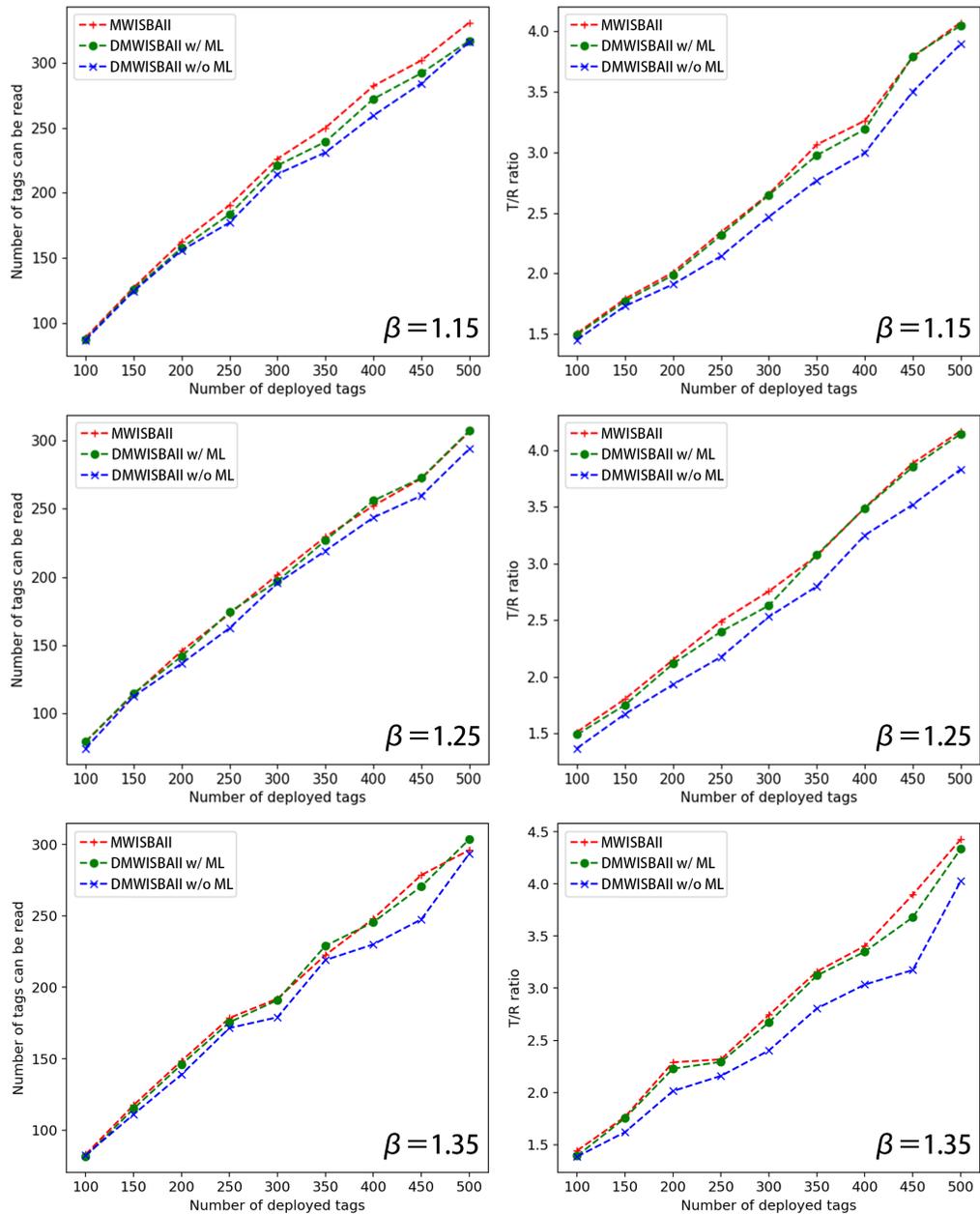


Figure 3.13: Performance evaluation results when reader interval is $8m$.

3.8 Summary

We proposed the distributed MWISBAll (w/o ML) at first. However, we found that due to the lack of global information, there is a gap in performance between the

distributed MWISBAII and the centralized MWISBAII. The centralized MWISBAII algorithm will active the reader with the highest cost value in each iteration, and this highest cost value is a global highest cost. However, in [138], the distributed algorithm tends to miss some critical readers, which should be activated. To solve this problem, we proposed a machine learning auxiliary approach to help each reader make a better decision. The machine learning model is a multi-layer neural network, which is trained on a central server only when the whole RFID system is set up for the first time. The training data is generated by running MWISBAII on multiple simulations. Therefore, the neural network is a supervised learning empirical model. The trained neural network model will be broadcast to each reader to predict the score of each reader. Although the training process is binary classification learning, we can interpret the output generated by the trained neural network model as some confidence level or score. Because of the constraint of the sigmoid activation function, the score is a continuous value between 0 and 1. A higher score means the neural network predicts that there is a higher probability of activating this reader to achieve better performance. The experimental results proved that the machine learning auxiliary approach helps the proposed distributed algorithm to make a more effective solution.

The experiments in this work followed the assumption that RFID readers are uniformly distributed. Nevertheless, in many real-world scenarios, the distribution of RFID readers could be manifold. Therefore, merely training a single neural network model and broadcast the trained model to every RFID reader might not improve the performance significantly. In the future, we can employ unsupervised learning clustering algorithm such as k-means and mean-shift clustering to divide the RFID readers into groups based on the features $(w_i, c_i, \varpi_i, \iota_i)$, and train a neural network model for each group of RFID readers individually. One challenge of this approach is that it is difficult to find an appropriate k value (the number of clusters). The other challenge is that, once the neural network model is trained for each cluster of readers, distributing the trained models to each cluster of readers increases the communication overhead. Besides the above-mentioned challenges, a significant limitation of the supervised learning-based approach is lack of flexibility. For instance, once the RFID system is deployed, the position of each RFID reader should not change, or we need to re-train the artificial neural network from scratch. Therefore, dealing with dynamic dense RFID readers systems is also a future research direction.

Chapter 4

Mobile Edge Computing Task Offloading Optimization

In a mobile edge computing (MEC) network, mobile devices could selectively offload tasks to the edge server(s) to save time and energy. However, we should consider many dynamic factors in task offloading optimization, which increases the complexity of this problem. Instead of executing the traditional optimization algorithm repeatedly, a well-trained empirical model such as an artificial neural network could be more efficient in decision making. In this research, considering the potential uneven spatial distribution of mobile devices in an MEC network with multiple wireless edge gateways, we allow an edge gateway to offload tasks to a nearby edge gateway further. We propose a deep reinforcement learning-based joint optimization approach for both device-level and edge-level task offloading. Experimental results show that the proposed approach achieves a near-optimal task delay performance and a better trade-off between the task delay and the energy consumption on tasks.

4.1	Introduction	45
4.2	Related Work	46
4.3	Network Model	47
4.4	Problem Statement	48
4.4.1	Local Computing Mode Cost	49
4.4.2	Task Offloading Modes Cost	50
4.4.3	Optimization Objective	51

4.5	The Proposed Optimization Approach	51
4.6	Simulation and Experimental Results	54
4.7	Summary	64

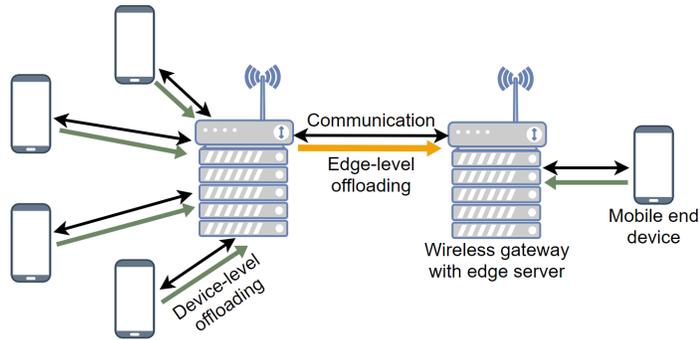
4.1 Introduction

The process of transferring a task from the local device to the cloud server or the edge server for execution and getting the result back is referred to as task offloading [1]. However, in many scenarios, task offloading may not always achieve satisfactory performance without optimization. Many energy-efficient MEC task offloading optimization problems consider minimizing mobile device energy consumption while meeting the demand for delays. In [42], Hao et al. introduce a new concept of optimizing task caching and task offloading jointly. They model the task caching problem as a 0-1 programming problem and model the task offloading problem as a mixed-integer nonlinear programming problem. Whereas, this approach has a limited use case since it does not support many types of computing tasks. Liu et al. propose an optimization approach to minimize computation and transmit power subject to latency and reliability constraints [80]. Different from many similar works, they consider the co-channel interference, queuing latency and offloading reliability. They also leverage the extreme value theory to deal with unusual extreme events.

The objective of energy-efficient task offloading optimization can also be achieved by optimizing the task offloading scheduling [84, 127]. Mao et al. consider the situation that all the computation tasks need to be offloaded due to extremely limited computational resources in mobile devices. However, they fail to take the potential dependency among tasks into consideration [84]. In [127], Wang et al. represent the dependency of tasks as a directed acyclic graph and utilize deep neural network-based reinforcement learning to make optimal task offloading schedule. The deep neural network includes an encoder for embedding the task graph and a decoder for predicting the offloading action for each task.

In this work, we assume that in a wireless MEC network, multiple edge gateways are deployed to cover a large area, and each edge gateway has a dedicated edge server. An end device can offload tasks to the nearest edge gateway (device-level offloading). Whereas, in some real-world scenarios, devices are not evenly distributed spatially [118]. Therefore, we allow a relatively-overloaded edge server to further offload a

task to one of its nearby edge gateways through wireless communication (edge-level offloading). For instance, in Figure 4.1, the left edge gateway could further offload the tasks to the right edge gateway, because the left edge gateway has relatively more connected devices. We leverage deep reinforcement learning to optimize this multi-level joint task offloading problem. In some relative studies, the objective of task offloading is to minimize energy consumption and latency [81, 150]. Similarly, the objective of our deep reinforcement learning algorithm is to minimize the average task execution delay and the end device energy consumption on each task.



© Peizhi Yan

Figure 4.1: A wireless edge computing network with two edge gateways.

The rest of this chapter is organized as follows. We review some related works in Section 4.2. In Section 4.3, we introduce the MEC network model. Section 4.4 depicts the proposed joint task offloading optimization problem and formulates the optimization objective. In Section 4.5, we propose the multi-level task offloading joint optimization approach. Section 4.6 shows the experimental results. We conclude this chapter in Section 4.7 and outline some directions for future work.

4.2 Related Work

In ultra-dense wireless networks such as 5G network with overlapping small base stations, an end device could choose to communicate with one of many small base stations. Some task offloading approaches allow multiple edge servers to provide service for a single end device [14, 23, 51]. Du et al. [23] propose an online optimization algorithm based on Lyapunov optimization to maximize the number of end devices served with minimum service cost in the long term. A deep reinforcement learning-based approach proposed in [51] assumes that each end device can choose to execute

a task locally, or offload the task to one of many base stations (each base station is equipped with an edge server), or offload the task to the cloud server through one of those base stations. However, this approach is not flexible since the neural network architecture is fixed, and once the network topology changes (such as the removal or addition of base station), one needs to rebuild a neural network and retrain it through reinforcement learning.

Traditional reinforcement learning-based approaches allow distributed decision making in nature since it simulates the learning process of an individual agent [81, 96]. Whereas, the capability of a traditional reinforcement learning-based approach is limited. Because many dynamic and stochastic factors affect the decision making of task offloading, exploring all the possibilities is infeasible. The deep reinforcement learning-based approach can address this limitation by storing the empirical knowledge into a deep neural network that works well on the input with an extensive and continuous range [14, 51, 82, 127]. Furthermore, most deep reinforcement learning-based optimization approach has a linear inference time complexity given the states with a fixed amount of features.

4.3 Network Model

We consider the MEC network with multiple end devices and multiple wireless edge gateways (see Figure 4.1). Each end device communicates to the nearest wireless edge gateway. Wireless edge gateways can communicate with each other if they are within each others' wireless communication range. The wireless channel access scheme is time division multiple access. Therefore we do not consider the interference of end devices. Each edge gateway is equipped with a dedicated edge server. We assume that every wireless edge gateway has the same configuration, and so does every end device. We use $\mathcal{U} = \{u_1, u_2, \dots, u_n\}$ to denote the set of mobile end devices, and u_i denotes the i^{th} end device in \mathcal{U} . We denote the set of edge gateways as $\mathcal{E} = \{\varepsilon_1, \varepsilon_2, \dots, \varepsilon_m\}$ (in most scenarios, $n \gg m$), and the j^{th} edge gateway is ε_j . We represent the connectivity between end devices and edge gateways as a bipartite graph $G = (\mathcal{U}, \mathcal{E}, E)$, where E is the set of edges. If u_i can communicate with ε_j , then edge $e_{u_i\varepsilon_j} \in E$. We do not allow an end device to communicate with more than one edge gateways, therefore, $\nexists k \neq j, e_{u_i\varepsilon_j} \in E$ and $e_{u_i\varepsilon_k} \in E$. Because an edge gateway could communicate with its nearby edge gateways, we represent this gateway-level connectivity as graph $G^* = (\mathcal{E}, E^*)$, where $E^* = \{e_{\varepsilon_j\varepsilon_k} | j \neq k, \varepsilon_j \in \mathcal{E}, \varepsilon_k \in \mathcal{E}\}$.

We denote the frequency of the wireless channel as φ (Hz); the frequency bandwidth is denoted as ϑ (Hz). The additive white Gaussian noise of the channel is denoted as σ . We assume that the power (measure in watts) received by the wireless receiver's antenna obey the Friis transmission equation [33]

$$p_{rx} = \frac{p_{tx}g_{tx}g_{rx}\lambda^2}{(4\pi d_{tx,rx})^2}, \quad (4.1)$$

where p_{rx} , p_{tx} represent the received power and the transmit power respectively; g_{tx} , g_{rx} represent the gain (watt) of transmitter's antenna and the receiver's antenna respectively; λ represents the wavelength which is approximately equivalent to $(3.0 \times 10^8)/\varphi$; $d_{tx,rx}$ denotes the distance between the transmitter and the receiver. The wireless transmission rate (bit/s) is estimated by

$$r_{tx,rx} = \vartheta \log_2 \left(1 + \frac{p_{rx}}{\sigma} \right). \quad (4.2)$$

It should be noted that the Equation 4.2 is only used in the simulation, to simplify the communication model. Therefore, the simulation assumes that: 1) the antenna is an ideal isotropic antenna that radiates its power uniformly in all directions; 2) the space is obstacle (such as the walls, ceilings, etc.) free.

4.4 Problem Statement

We use \mathcal{T} to represent the queue of tasks. Tasks in \mathcal{T} are processed in a first-in-first-out way. The task queues of an end device u_i and an edge gateway ε_j are denoted as $\mathcal{T}_{u_i} = \{\tau_{u_i,1}, \tau_{u_i,2}, \dots, \tau_{u_i,l}\}$ and $\mathcal{T}_{\varepsilon_j} = \{\tau_{\varepsilon_j,1}, \tau_{\varepsilon_j,2}, \dots, \tau_{\varepsilon_j,h}\}$ respectively. A task τ has two properties $\wp(\tau)$ and $\omega(\tau)$, where $\wp(\tau)$ denotes the task size (measured in bits), and $\omega(\tau)$ denotes the workload (average number of CPU cycles for processing each bit). In this research, we assume that each end device u_i will make a device-level task offloading decision \aleph ($\aleph \in \{0, 1\}$, where 0 represents not to offload; 1 represents to offload) for each task. The edge gateway also makes an edge-level task offloading decision Ψ ($\Psi \in \{0, 1\}$) for each task directly offloaded by the connected end device. This means that each task $\tau_{u_i,h}$ generated by an end device u_i could be offloaded to ε_j if $e_{u_i\varepsilon_j} \in E$ (one-hop offloading, or device-level offloading); and $\tau_{\varepsilon_j,h}$ could be further offloaded by ε_j to another edge gateway ε_k if $e_{\varepsilon_i\varepsilon_k} \in E^*$ (two-hop offloading, or device-level plus edge-level offloading). We do not allow any task offloading that is more than

two-hop to prevent chaos and fail in executing a task timely. We also assume that the size of the result after each execution of a task is negligible [13]. Therefore, we do not consider the efficiency of the transmission of results. In the rest of this section, we formulate the costs (delay and energy consumption) of local computing mode and two levels of task offloading modes and introduce the optimization objective.

4.4.1 Local Computing Mode Cost

Suppose an end device u_i has $\tau_{u_i,1}$ as the first task in \mathcal{T}_{u_i} , u_i needs to make a device-level decision $\aleph(\tau_{u_i,1})$ for this task. If $\aleph(\tau_{u_i,1}) = 0$, u_i executes $\tau_{u_i,1}$ locally. We use f_{u_i} (Hz) to denote the CPU frequency of u_i , which represents the amount of CPU cycles per second. The total amount of CPU cycles for executing $\tau_{u_i,1}$ is $\wp(\tau_{u_i,1})\omega(\tau_{u_i,1})$. Therefore, the approximate local execution time consumption on $\tau_{u_i,1}$ is

$$t_{exe_local} = \frac{\wp(\tau_{u_i,1})\omega(\tau_{u_i,1})}{f_u}. \quad (4.3)$$

We define the local waiting time of $\tau_{u_i,1}$, which is the time delay from the time $\tau_{u_i,1}$ is generated to the time $\tau_{u_i,1}$ is retrieved from \mathcal{T}_{u_i} , as t_{wait_local} . The estimated total delay of local computing mode (time consumption) on $\tau_{u_i,1}$ is

$$t_{local} = t_{exe_local} + t_{wait_local}. \quad (4.4)$$

Assume the average CPU working power consumption of end device is ρ (watt). Then, the estimated CPU energy consumption for executing $\tau_{u_i,1}$ is

$$e_{exe_local} = \rho t_{exe_local}. \quad (4.5)$$

The estimated total cost on $\tau_{u_i,1}$ in local computing mode is

$$c_{local} = t_{local} + \varkappa e_{exe_local}, \quad (4.6)$$

where \varkappa is the weight factor because time cost and energy consumption could have different scales; moreover, we may want to make an adjustable trade-off between time cost and energy consumption in practice.

4.4.2 Task Offloading Modes Cost

If the device-level decision made by u_i is $\aleph(\tau_{u_i,1}) = 1$, u_i will offload $\tau_{u_i,1}$ to the edge gateway ε_j ($e_{u_i\varepsilon_j} \in E$). Suppose the transmit power of u_i is p_{u_i} ; the antenna gain of u_i and ε_j are g_{u_i} and g_{ε_j} respectively. Based on Equation 4.1, we calculate the received signal power p_{rx} ($p_{tx} = p_{u_i}; g_{tx} = g_{u_i}; g_{rx} = g_{\varepsilon_j}; d_{tx,rx} = d_{u_i,\varepsilon_j}$). The estimated transmission rate from u_i to ε_j is derived by Equation 4.2, denoted as r_{u_i,ε_j} . We define the transmission time for offloading $\tau_{u_i,1}$ from u_i to ε_j as

$$t_{trans} = \wp(\tau_{u_i,1})/r_{u_i,\varepsilon_j}. \quad (4.7)$$

The transmission energy consumption is derived by

$$e_{trans} = p_{u_i}t_{trans}. \quad (4.8)$$

Once ε_j retrieves $\tau_{\varepsilon_j,1}$ (assume this task used to be $\tau_{u_i,1}$) from $\mathcal{T}_{\varepsilon_j}$, ε_j also make an edge-level decision $\Psi(\tau_{\varepsilon_j,1})$. Suppose $e_{\varepsilon_j\varepsilon_k} \in E^*$ (also, $\nexists \varepsilon_z \in \mathcal{E}$, $e_{\varepsilon_j\varepsilon_z} \in E^*$, $k \neq z$), if $\Psi(\tau_{\varepsilon_j,1}) = 0$, ε_j will not further offload $\tau_{\varepsilon_j,1}$ to ε_k . Assume the CPU frequency of ε_j is f_{ε_j} , then the execution time consumption of $\tau_{\varepsilon_j,1}$ is denoted by

$$t_{exe_edge} = \frac{\wp(\tau_{\varepsilon_j,1})\omega(\tau_{\varepsilon_j,1})}{f_{\varepsilon_j}}. \quad (4.9)$$

We denote the waiting time of $\tau_{\varepsilon_j,1}$ in $\mathcal{T}_{\varepsilon_j}$ as t_{wait_edge} . The estimated total cost on $\tau_{\varepsilon_j,1}$ in device-level task offloading mode is defined as

$$C_{device_level} = t_{exe_edge} + t_{wait_local} + t_{wait_edge} + t_{trans} + \varkappa e_{trans}. \quad (4.10)$$

Then we consider the situation when $\Psi(\tau_{\varepsilon_j,1}) = 1$. In this situation, ε_j decides to offload $\tau_{\varepsilon_j,1}$ to ε_k , which we call edge-level task offloading. The transmission rate from ε_j to ε_k is denoted as $r_{\varepsilon_j,\varepsilon_k}$, which is derived by Equation 4.1 and Equation 4.2, where $p_{tx} = p_{\varepsilon_j}; g_{tx} = g_{\varepsilon_j}; g_{rx} = g_{\varepsilon_k}; d_{tx,rx} = d_{\varepsilon_j,\varepsilon_k}$. The transmission time for this edge-level offloading is

$$t'_{trans} = \wp(\tau_{\varepsilon_j,1})/r_{\varepsilon_j,\varepsilon_k}. \quad (4.11)$$

We denote the waiting time of $\tau_{\varepsilon_k,1}$ (used to be $\tau_{\varepsilon_j,1}$) in $\mathcal{T}_{\varepsilon_k}$ as t'_{wait_edge} . Because we assume that edge servers have same configuration, f_{ε_k} is equivalent to f_{ε_j} . Therefore,

the estimated total cost on $\tau_{\varepsilon_k,1}$ with this task offloading mode is calculated by

$$C_{edge_level} = C_{device_level} + t'_{wait_edge} + t'_{trans}. \quad (4.12)$$

4.4.3 Optimization Objective

In practice, mobile end devices have limited power reserve. Thereby, we consider the energy consumption of mobile end devices in optimization. Regarding Equation 4.6, Equation 4.10, and Equation 4.12, the objective of an end device u_i is to optimize the device-level task offloading policy π_{\aleph} to minimize the following equation:

$$C_{u_i} = \frac{\sum_{\tau \in \mathcal{T}_{u_i}^*} \left[[1 - \aleph(\tau)] C_{local} + \aleph(\tau) \left[[1 - \Psi(\tau)] C_{device_level} + \Psi(\tau) C_{edge_level} \right] \right]}{|\mathcal{T}_{u_i}^*|}, \quad (4.13)$$

where $\mathcal{T}_{u_i}^*$ is the set of completed tasks of u_i . It is worth noting that, the edge-level offloading decision $\Psi(\tau)$ is dependent on edge gateway.

Unlike the mobile end device, an edge gateway usually equipped with a stable power source. Therefore, we do not consider the energy consumption of edge gateways. The objective of an edge gateway ε_j is to optimize the edge-level task offloading policy π_{Ψ} to minimize

$$C_{\varepsilon_j} = \frac{\sum_{\tau \in \mathcal{T}_{\varepsilon_j}^*} [t_{wait_edge} + \Psi(\tau)(t'_{wait_edge} + t'_{trans})]}{|\mathcal{T}_{\varepsilon_j}^*|}, \quad (4.14)$$

similarly, $\mathcal{T}_{\varepsilon_j}^*$ represents the set of completed tasks of ε_j .

4.5 The Proposed Optimization Approach

In this section, we present the joint optimization approach for the proposed edge computing network model. We model each level of task offloading as a continuous state Markov decision process [121]. In continuous state MDP, the values of a state belong to continuous ranges. Therefore, there is an infinite number of potential states. Since traditional Q-learning only deals with finite-state MDP, using traditional Q-learning requires discretizing the infinite set of states into a finite set of states, which in turn will lead to the loss of accuracy. Deep Q-learning (DQL) leverages an artificial neural network to imitate the function of Q-table in traditional Q-learning, while an

artificial neural network can take the arbitrary value(s) as the input, thus supports continuous state MDP.

We apply DQL for both device-level and edge-level task offloading optimizations. Each level of task offloading has a dedicated deep Q-network (DQN). We define the device-level DQN parameters as θ_{\aleph} , and edge-level DQN parameters as θ_{Ψ} . Both levels of policy (π_{\aleph} and π_{Ψ}) have the same action space $\mathcal{A} = \{0, 1\}$. Suppose the end device is u_i , and u_i is connected to ε_j . The input state of device-level DQN is defined as s_{\aleph} , which is a quintuple $(r_{u_i, \varepsilon_j}, |\mathcal{T}_{u_i}|, \ell_{u_i}, |\mathcal{T}_{\varepsilon_j}|, \ell_{\varepsilon_j}) \in \mathcal{S}_{\aleph} = \mathbb{R}^5$, where ℓ_{u_i} and ℓ_{ε_j} are the current workload of u_i and ε_j respectively. The workload ℓ is defined as $\ell = \sum_{\tau \in \mathcal{T}} \wp(\tau) \omega(\tau)$, which is the sum of required CPU cycles of each task in the task queue. Then, the probabilistic transition function can be defined as $\mathbb{P}_{\aleph}(s'_{\aleph} | s_{\aleph}, a) : \mathcal{S}_{\aleph} \times \mathcal{S}_{\aleph} \times \mathcal{A} = \mathbb{R}^5 \times \mathbb{R}^5 \times \{0, 1\} \rightarrow [0, 1]$, which represents the probability of the transition from a state s_{\aleph} to another state s'_{\aleph} under action a .

In edge-level task offloading, given an edge gateway ε_j , $\mathcal{N}_{\varepsilon_j}$ denotes the set of neighbor edge gateways of ε_j ($\forall \varepsilon_k \in \mathcal{N}_{\varepsilon_j}, e_{\varepsilon_j \varepsilon_k} \in E^*$). The input state of edge-level DQN is defined as s_{Ψ} , which is a septuple $(r_{\varepsilon_j, \varepsilon_k}, |\mathcal{T}_{\varepsilon_j}|, \ell_{\varepsilon_j}, |\mathcal{T}_{\varepsilon_k}|, \ell_{\varepsilon_k}, |\mathcal{D}_{\varepsilon_j}|, |\mathcal{D}_{\varepsilon_k}|) \in \mathcal{S}_{\Psi} = \mathbb{R}^7$, where $|\mathcal{D}_{\varepsilon_j}|$ and $|\mathcal{D}_{\varepsilon_k}|$ represent the number of connected end devices of ε_j and ε_k respectively. Same as device-level task offloading, the probabilistic transition function of edge-level task offloading can be defined as $\mathbb{P}_{\Psi}(s'_{\Psi} | s_{\Psi}, a) : \mathcal{S}_{\Psi} \times \mathcal{S}_{\Psi} \times \mathcal{A} = \mathbb{R}^7 \times \mathbb{R}^7 \times \{0, 1\} \rightarrow [0, 1]$, which represents the probability of the transition from a state s_{Ψ} to another state s'_{Ψ} under action a .

Instead of maximizing the total discounted reward, we modify Equation 2.13 to minimize the total discounted cost

$$y(s, a) = c + \gamma \min_{a'} Q_{\theta'}(s', a'), \quad (4.15)$$

where c is the immediate cost. For device-level task offloading, we use Equation 4.13 to calculate the immediate cost, and for edge-level task offloading, we use Equation 4.14 to derive the immediate cost. Therefore, the corresponding action of a state s is $\operatorname{argmin}_a Q_{\theta}(s, a)$, and a transition is defined as (s, a, c, s') .

Algorithm 1 depicts the proposed device-level task offloading and optimization approach. One can initialize the DQN parameters θ_{\aleph} randomly, or load the pre-trained parameters during the initialization stage. There are learning mode and non-learning mode. If the learning mode is on, the device needs to store the transitions into a transition history memory \mathcal{M}_{\aleph} for experience replay learning. The \mathcal{M}_{\aleph} has a

limited size of ψ transitions. Once the memory is full, the earliest transition will be removed for adding a new transition. It should be noted that we do not update the DQN instantly after the execution of each task. Instead, we train the DQN on \mathcal{M}_{\aleph} after the successful execution of κ tasks (a training step). Because we use gradient descent-based optimizer to optimize the DQN parameters, we define α as the network learning rate (different from the learning rate α^\diamond in Q-learning). We also decrease the exploration rate ϵ after each training step, the exploration rate decay factor is ζ ($0 < \zeta \leq 1$).

Algorithm 1: Device-level task offloading algorithm.

```

1 initialize the transition history memory  $\mathcal{M}_{\aleph}$ ;
2 initialize the DQN parameters  $\theta_{\aleph}$ ;
3 do in parallel
4   while device  $u_i$  in operation do
5      $\tau_{u_i,1} \leftarrow$  retrieve task from  $\mathcal{T}_{u_i}$ ;
6      $s_{\aleph} \leftarrow$  collect current state info.;
7     if learning mode is on then
8       if random value from  $[0, 1] \geq \epsilon$  then
9          $a_{\aleph} \leftarrow \operatorname{argmin}_a \mathcal{Q}_{\theta_{\aleph}}(s_{\aleph}, a)$ ;
10      else
11         $a_{\aleph} \leftarrow$  random value from  $\aleph$ ;
12      create a transition  $(s_{\aleph}, a_{\aleph}, -, -)$  for  $\tau_{u_i,1}$ ;
13    else
14       $a_{\aleph} \leftarrow \operatorname{argmin}_a \mathcal{Q}_{\theta_{\aleph}}(s_{\aleph}, a)$ ;
15    if  $a_{\aleph} = 0$  then
16      execute  $\tau_{u_i,1}$ ;
17    else
18      offload  $\tau_{u_i,1}$  to edge gateway;
19  do in parallel
20    counter  $\leftarrow 0$ ;
21    while learning mode is on do
22      if  $\tau_{u_i,1}$  is done then
23         $c_{\aleph} \leftarrow$  compute the current average cost through Equation 4.13;
24         $s'_{\aleph} \leftarrow$  collect next state info.;
25        complete the transition  $(s_{\aleph}, a_{\aleph}, c_{\aleph}, s'_{\aleph})$ ;
26        store  $(s_{\aleph}, a_{\aleph}, c_{\aleph}, s'_{\aleph})$  to  $\mathcal{M}_{\aleph}$ ;
27        counter  $\leftarrow$  counter + 1;
28      if counter mod  $\kappa = 0$  then
29         $\theta'_{\aleph} \leftarrow \theta_{\aleph}$ ;
30        train  $\theta_{\aleph}$  on  $\mathcal{M}_{\aleph}$ ;
31         $\epsilon \leftarrow \zeta \epsilon$ ;

```

The proposed edge-level task offloading approach (see Algorithm 2) is similar to

the device-level task offloading approach. The difference is that each edge gateway could communicate with more than one nearby edge gateways. Assume the current edge gateway is ε_j . For each of its neighbor gateway $\varepsilon_k \in \mathcal{N}_{\varepsilon_j}$, we first get the corresponding status s_Ψ , and then get the corresponding Q-values from the DQN: $Q_{\theta_\Psi}(s_\Psi)$. Based on the Q-values, we get the list of neighbour gateways to which the current gateway that could offload the current task. If there are more than one neighbour gateways in the list, we choose the gateway ε_k with the lowest Q-value for $a = 1$, and let ε_j to offload the task to ε_k . Furthermore, if the current task was offloaded by another edge gateway, this edge gateway should execute the current task instantly without any further offloading. Figure 4.2 depicts the proposed multi-level task offloading approach.

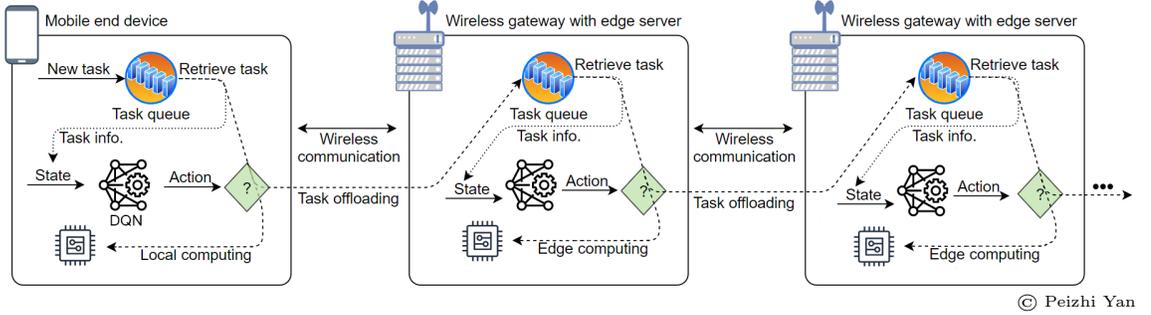


Figure 4.2: The proposed multi-level task offloading pipeline.

Assume during the simulation-based training stage, the average total executed tasks of each end device is k . Then, the total executed tasks in the MEC system is nk . For device-level Q-learning, the number of neural network training steps is $O(\frac{nk\psi}{\kappa})$. For edge-level Q-learning, the number of neural network training steps is also $O(\frac{nk\psi}{\kappa})$. The number of decision-making steps in device-level Q-learning is nk ; for edge-level Q-learning is nmk in the worst case. Therefore, the overall time complexity of the simulation-based training is $O(nmk + \frac{nk\psi}{\kappa})$. If $m \ll n$, and $\kappa < \psi$, the time-complexity can be simplified to $O(\frac{nk\psi}{\kappa})$.

4.6 Simulation and Experimental Results

We simulate a $100\text{m} \times 100\text{m}$ area to deploy the wireless edge gateways and end devices. We represent the square area in a Cartesian coordinate system, where the locations of four corners are $(0, 0)$, $(0, 100)$, $(100, 0)$, and $(100, 100)$. There are four wireless edge

Algorithm 2: Edge-level task offloading algorithm.

```

1 initialize the transition history memory  $\mathcal{M}_\Psi$ ;
2 initialize the DQN parameters  $\theta_\Psi$ ;
3 Procedure choose_gateway()
4    $\mathbb{Q} \leftarrow$  empty set;
5   foreach  $\varepsilon_k \in \mathcal{N}_{\varepsilon_j}$  do
6      $s_\Psi \leftarrow$  collect current state info.;
7     if  $\text{argmin}_a \mathcal{Q}_{\theta_\Psi}(s_\Psi, a) = 1$  then
8        $\mathbb{Q} \leftarrow \mathbb{Q} \cup \{(\varepsilon_k, \mathcal{Q}_{\theta_\Psi}(s_\Psi, 1))\}$ ;
9     if  $\mathbb{Q} = \emptyset$  then
10       $\varepsilon_k \leftarrow$  random edge gateway from  $\mathcal{N}_{\varepsilon_j}$ ;  $a_\Psi \leftarrow 0$ ;
11    else
12      get the pair  $(\varepsilon_k, \mathcal{Q}_{\theta_\Psi}(s_\Psi, 1))$  from  $\mathbb{Q}$  where  $\mathcal{Q}_{\theta_\Psi}(s_\Psi, 1)$  is the smallest;  $a_\Psi \leftarrow 1$ ;
13    return  $\varepsilon_k, a_\Psi$ ;
14 do in parallel
15   while server  $\varepsilon_j$  in operation do
16      $\tau_{\varepsilon_j,1} \leftarrow$  retrieve task from  $\mathcal{T}_\mathcal{E}$ ;
17     if  $\tau_{\varepsilon_j,1}$  offloaded by another edge gateway then
18       execute  $\tau_{\varepsilon_j,1}$ ; skip the rest steps and continue the while loop;
19     if learning mode is on then
20       if random value from  $[0, 1] \geq \epsilon$  then
21          $\varepsilon_k, a_\Psi \leftarrow$  choose_gateway();
22       else
23          $a_\Psi \leftarrow$  random value from  $\Psi$ ;  $\varepsilon_k \leftarrow$  random from  $\mathcal{N}_{\varepsilon_j}$ ;
24          $s_\Psi \leftarrow$  collect current state info.;
25         create a transition  $(s_\Psi, a_\Psi, -, -)$  for  $\tau_{\varepsilon_j,1}$ ;
26       else
27          $\varepsilon_k, a_\Psi \leftarrow$  choose_gateway();
28       if  $a_\Psi = 0$  then
29         execute  $\tau_{\varepsilon_j,1}$ ;
30       else
31         offload  $\tau_{\varepsilon_j,1}$  to edge gateway  $\varepsilon_k$ ;
32 do in parallel
33   counter  $\leftarrow 0$ ;
34   while learning mode is on do
35     if  $\tau_{\varepsilon_j,1}$  is done then
36        $c_\Psi \leftarrow$  compute the current average cost through Equation 4.14;
37        $s'_\Psi \leftarrow$  collect next state info.;
38       complete the transition  $(s_\Psi, a_\Psi, c_\Psi, s'_\Psi)$ ;
39       store  $(s_\Psi, a_\Psi, c_\Psi, s'_\Psi)$  to  $\mathcal{M}_\Psi$ ;
40       counter  $\leftarrow$  counter + 1;
41     if counter mod  $\kappa = 0$  then
42        $\theta'_\Psi \leftarrow \theta_\Psi$ ; train  $\theta_\Psi$  on  $\mathcal{M}_\Psi$ ;  $\epsilon \leftarrow \zeta\epsilon$ ;

```

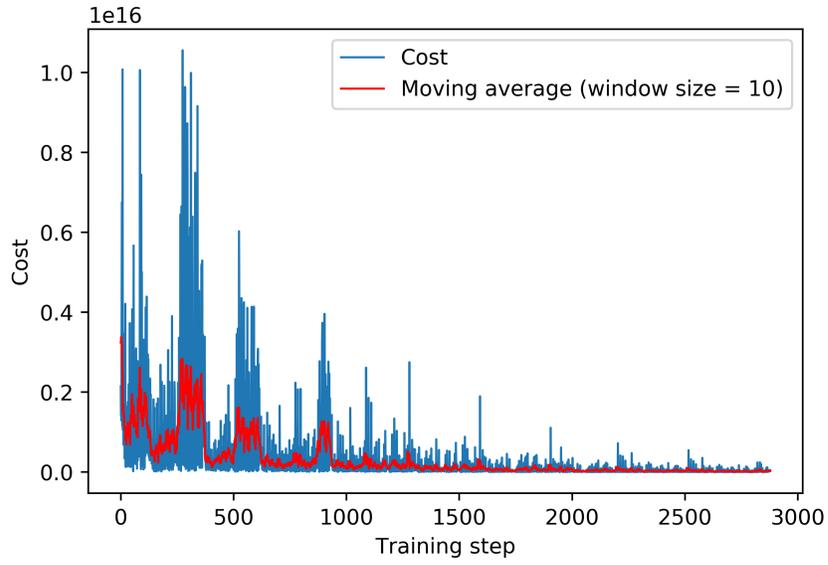
Parameter	Value
Transmit power of end device: p_u	0.04 (watt)
Transmit power of edge gateway: $p_{\mathcal{E}}$	0.1 (watt)
Gain of end device: g_u	3 (dB)
Gain of edge gateway: $g_{\mathcal{E}}$	10 (dB)
Channel frequency: φ	2.4 (GHz)
Bandwidth: ϑ	20 (MHz)
Gaussian noise*: σ	M:-80, SD:10 (dB)
Task size*: φ	M:25, SD:10 (Kbit)
Task workload*: ω	M:500, SD:100 (cycles/bit)
CPU frequency of end device: f_u	1 (GHz) \times 1 core
CPU frequency of edge server: $f_{\mathcal{E}}$	3 (GHz) \times 4 cores
End device CPU power: ρ	1 (watt)
Time slot: ν ; task probability: ξ	10 (ms); 0.5
Deep Q-learning hyperparameters: α ; \varkappa ; ϵ ; ζ ; γ ; κ ; ψ ; batch size	0.01; 0.1; 0.2; 0.999; 0.9; 100; 10,000; 32

* : random value from a normal distribution.

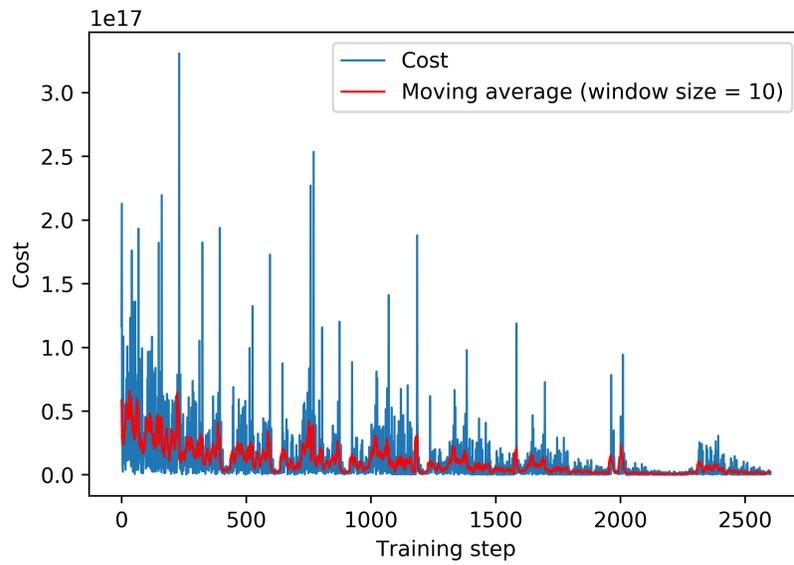
M: mean; SD: standard deviation.

Table 4.1: Experimental Parameters [2, 52, 92]

gateways evenly deployed at four locations: (25, 25), (25, 75), (75, 25), and (75, 75) to cover the experiment field. In the reinforcement learning stage, we consider the uneven spatial distribution of mobile end devices. We only utilize half of the experiment field to randomly deploy the end devices (suppose the location of an end device is (x, y) , then $0 \leq x \leq 50$, and $0 \leq y \leq 100$). The position of each end device is fixed in our simulation. In theory, two edge gateways will be much busier than the other two edge gateways due to this biased spatial distribution of end devices. We assume that in each time slot ν , a task has a chance of ξ to arrive. Considering the efficiency of training, we do not train a DQN for each end device or edge gateway separately. Instead, all the end devices share the identical DQN parameters θ_{\aleph} , and all the edge gateways share the identical DQN parameters θ_{Ψ} . To implement this idea, we store the transitions collected from all the end devices and edge gateways to \mathcal{M}_{\aleph} , and \mathcal{M}_{Ψ} respectively. During the training process, the number of deployed end devices is 400 (we assume that this is the maximum capacity for this MEC system). Each end device needs to execute 100 tasks in total in each simulation. A detailed list of the simulation parameters and the deep Q-learning hyper-parameters is shown in Table 4.1. Figure 4.3 depicts the training loss curves of both levels of DQL.



(a) Device-level Q-learning loss curve.



(b) Edge-level Q-learning loss curve.

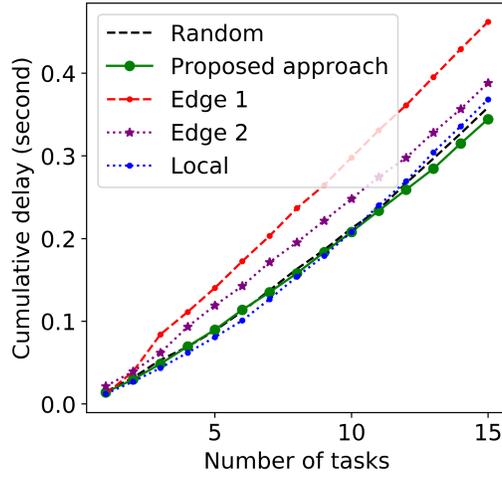
Figure 4.3: Deep Q-learning loss curves.

We carry out two groups of experiments (namely, experiment 1 and 2). In experiment 1, we only use half of the experiment field to randomly deploy the end devices (similar to the reinforcement learning stage). Figure 4.4 and Figure 4.5 depicts the experimental results of experiment one. Figure 4.4 shows the change of the average cumulative delay on an end device with the increase of the number of completed tasks. Figure 4.5 shows the change of the average cumulative energy consumption on an end device with the increase of the number of completed tasks. “Random” denotes the stochastic task offloading policy, which means that in both levels of task offloading, the decision is made randomly (each action has the same probability of been chosen). “Edge 1” denotes the uniform task offloading policy, where each end device always offloads the task to the edge gateway, but there is no edge-level task offloading. Similar to “Edge 1”, “Edge 2” has the same uniform device-level offloading policy. However, “Edge 2” allows stochastic edge-level task offloading. “Local” indicates that there is no task offloading allowed, which means that all the tasks should be executed locally. Experiment 2 has the identical simulation setting as experiment 1, except that the deployment range of each end device is the full $100\text{m} \times 100\text{m}$ experiment field. The experimental results of experiment two are shown in Figure 4.6 and Figure 4.7.

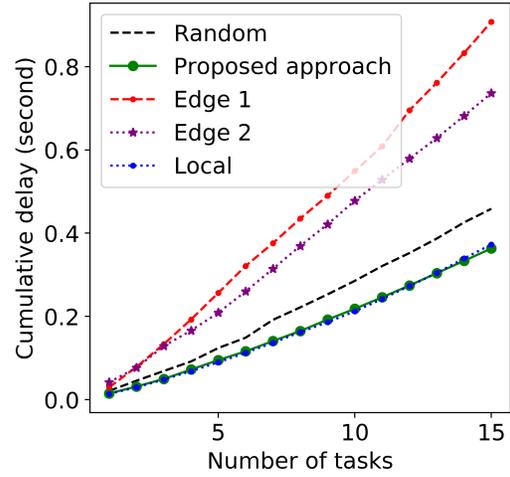
In Figure 4.4, we can see that when the number of deployed end devices is greater than 100, both “Edge 1” and “Edge 2” policies will cause higher delay than the “Local” policy. That is because the edge servers need to process all the tasks generated by all the end devices. Even edge servers have more computing power than each of the end devices, and the overload issue will break this advantage. With the number of deployed end devices increases, the gap between “Edge” policies and the “Local” policy becomes larger. Whereas, the proposed task offloading approach can always achieve a performance (in terms of delay), which is close to the “Local” policy. In Figure 4.5, because under both “Edge” policies, the end devices only need to consume energy on task transmission, the energy consumption of both “Edge” policies are lowest. On the contrary, under “Local” policy, the device energy consumption is the highest. “Edge” and “Local” policies tend to achieve the best performance in terms of either delay metric or energy consumption metric, whereas, also get the worst performance in terms of the other metric. The “Random” policy achieves a relatively balanced performance in terms of both metrics. However, the proposed method can always achieve near-optimal performance in terms of delay, but also have a reasonable energy consumption compared with the other polices. The results shown in Figure 4.6 and Figure 4.7 are similar with the results in Figure 4.4 and Figure 4.5. However, in

Figure 4.6a, “Edge 2” achieves the lowest delay, and “Local” gets the highest delay. The reason is that, because we use the full experimental field to deploy devices, the density of devices is not very high. Moreover, since the proposed task offloading policies are learned under the situation that the number of deployed devices is 400 (high density), when the number of density of devices is not very high, the proposed approach may not be able to achieve the best performance. However, the proposed method still achieves near-optimal performance.

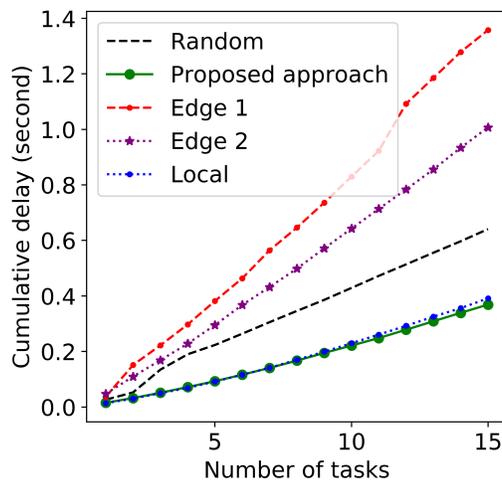
In summary, the “Local” approach always has the highest energy consumption than the other approaches under our experimental settings. Since all the tasks should be offloaded in both “Edge 1” and “Edge 2” approaches, the end device energy consumption in these two approaches is the lowest (only transmit energy consumption). The “Random” approach makes a balance between the “Local” approach, and the “Edge” approaches. The proposed approach achieves the near-optimal (compared with the other approaches) task delay performance while achieves a better trade-off of the task delay and the energy consumption than the stochastic approach.



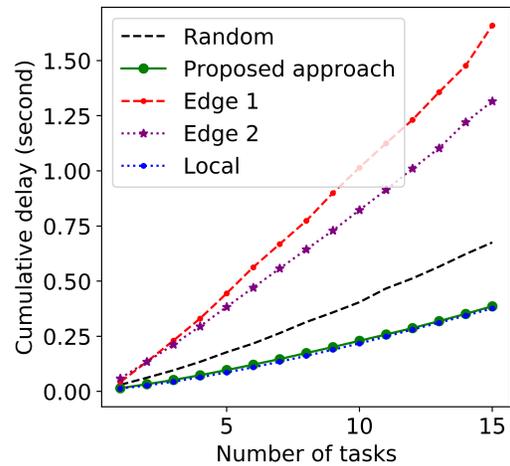
(a) 100 deployed devices



(b) 200 deployed devices



(c) 300 deployed devices



(d) 400 deployed devices

Figure 4.4: Average cumulative delay (Experiment 1).

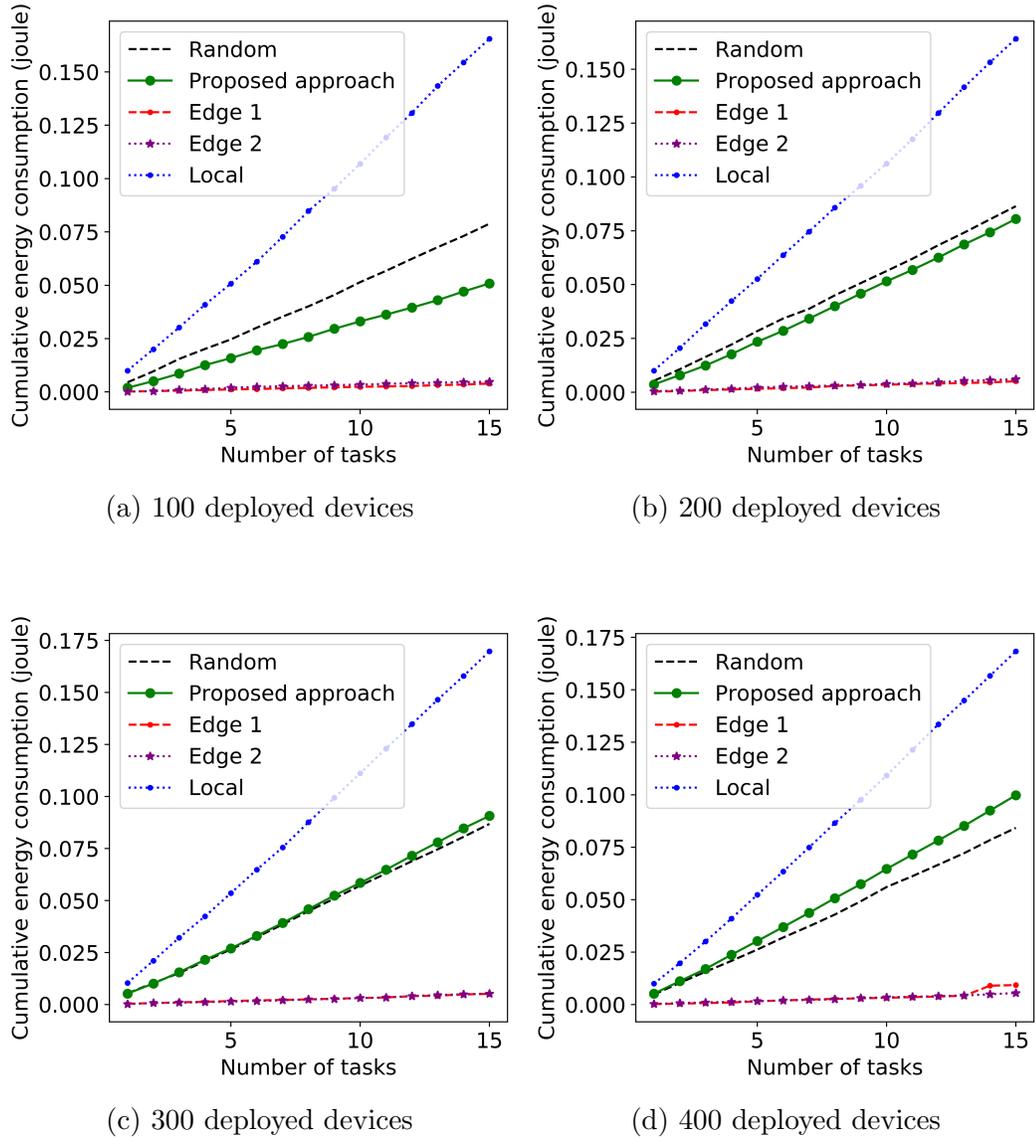
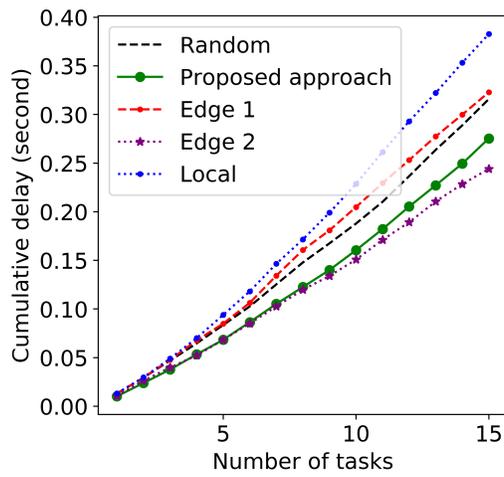
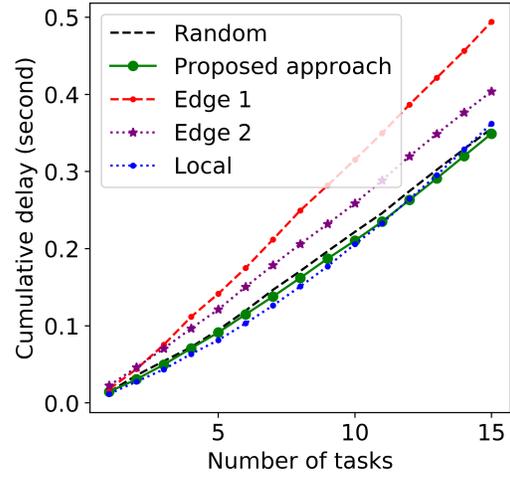


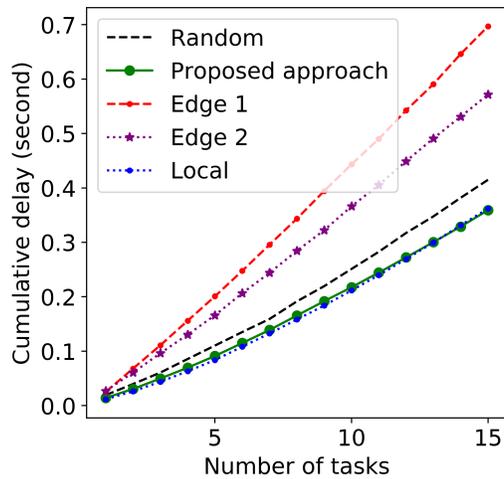
Figure 4.5: Average cumulative energy consumption (Experiment 1).



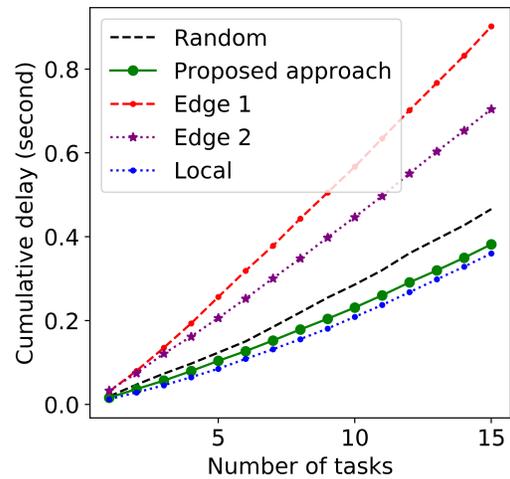
(a) 100 deployed devices



(b) 200 deployed devices

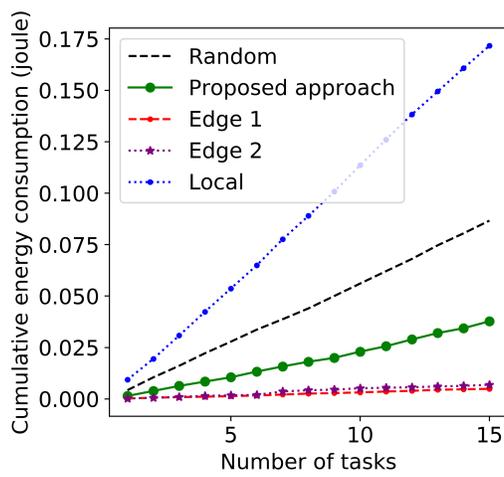


(c) 300 deployed devices

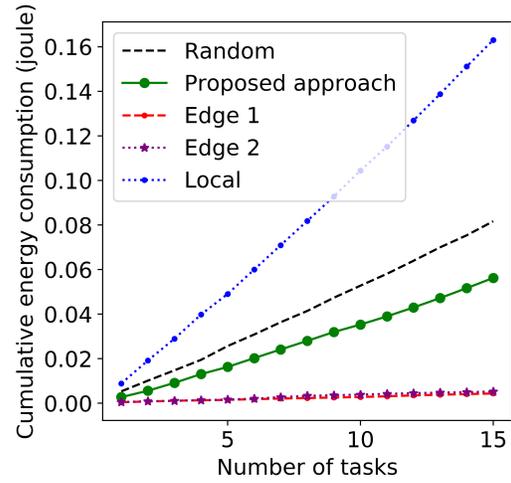


(d) 400 deployed devices

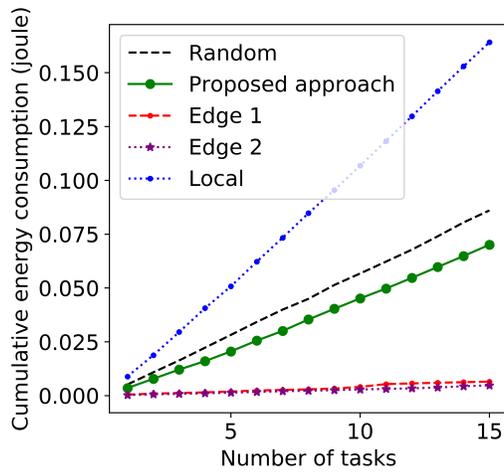
Figure 4.6: Average cumulative delay (Experiment 2).



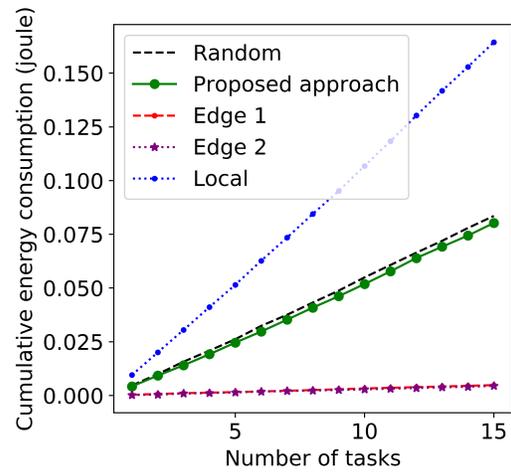
(a) 100 deployed devices



(b) 200 deployed devices



(c) 300 deployed devices



(d) 400 deployed devices

Figure 4.7: Average cumulative energy consumption (Experiment 2).

4.7 Summary

This work considers the situation that in a mobile edge computing network, mobile end devices can offload tasks to edge gateways (device-level task offloading), and edge gateways can further offload tasks to nearby edge gateways (edge-level task offloading). We formulate the problem of jointly optimizing the multi-level task offloading as a reinforcement learning problem and utilize the deep reinforcement learning method to solve the optimization problem. Experimental results indicate that the proposed approach achieves a near-optimal task delay performance and a better trade-off performance on task delay and task energy consumption than other task-offloading schemes. In the future, we will enable “on-device” reinforcement learning to handle the heterogeneity of end devices (such as different CPU and antenna configurations), and use the federated learning [68] approach to get the high-quality global model for initializing the Q-network of newly joined devices. Besides, we will investigate the feasibility of combining multiple tasks into a batch and make the batch-wise offloading decision.

Chapter 5

Wireless Ad-hoc Network Topology Control

Wireless ad-hoc IoT (WAIoT) is promising in providing connections for a considerable amount of devices in the next generation (5G and beyond 5G) networks. A challenge in WAIoT networks is that most network nodes are not stable due to the limited power supply (such as a battery). In this work, we focus on balancing node residual energy and node degree to prolong the network lifetime. We first present a statistic-based algorithm (named ED-index) for evaluating the network topology and further develop an energy-efficient topology control algorithm (named EDTC). The EDTC algorithm leverages the maximum spanning tree algorithm to build a robust backbone topology and utilizes the proposed ED-index algorithm to re-introduce some edges to the topology. We also present a graph convolutional network (GCN) based algorithm to imitate the initial EDTC algorithm through learning. Simulation results show that the proposed EDTC algorithm achieves better performance than the state-of-the-art topology control algorithm in terms of network lifetime. The GCN-based EDTC algorithm significantly reduces the topology construction time and also achieves a satisfactory performance compared with the initial EDTC algorithm.

5.1	Introduction	66
5.2	Existing Topology Control Algorithms	68
5.3	System Model and Problem Statement	72
5.4	The Proposed Approach	75

5.4.1	Energy Distribution Index	75
5.4.2	Energy-Degree Topology Control Algorithm	78
5.4.3	Graph Convolutional Network Based Energy-Degree Topology Control Algorithm	80
5.5	Experiments	83
5.6	Summary	95

5.1 Introduction

One challenge in WAIoT networks is that most WAIoT nodes have an independent but limited power source, such as a battery. Therefore, WAIoT nodes have a limited lifetime before the recharge/replacement of a battery. Moreover, the WAIoT network nodes could serve as both end-devices and relay-devices, yet the computing resources such as computing power, memory, and wireless bandwidth are scarce. A particular case of the WAIoT is the WSN, which is extremely resource-constrained [16]. Topology control is essential in prolonging network life and improving the efficiency of the network, considering the issues as mentioned above [50]. The primary methods of topology control include antenna power adjusting and neighbor node selection (we use “neighbor node” to represent the neighbor node appears in the topology graph, not the geographical neighbor). Different application scenarios often bring diverse requirements, so different kinds of topology control algorithms usually have entirely different assumptions and design goals.

We usually model a WAIoT as a topology graph, where the graph nodes represent wireless devices; edges represent the links between those wireless devices [95, 142]. Since the original graph often includes lots of redundant edges, one goal of most topology control algorithms is to eliminate the redundancy in the original graph [62]. Besides, a good topology control algorithm should be able to generate a subgraph that keeps the connectivity of the original graph as far as possible, and also meet some specific requirements such as high energy efficiency [62] (can be measured by the lifetime of the network) and high robustness (fault tolerance). Additionally, the run-time efficiency of the topology control algorithm itself is also significant. An ideal case of the WAIoT assumes that every device has the same wireless transmission range (not affected by the environment), and all the devices are arranged on a two-dimensional plane (Euclidean plane). One also assumes that there are no barriers that

could impede the wireless signal, and the communication between each pair of devices is bi-directional. In this ideal case, researchers often use a unit disk graph to model the network [129]. A unit disk graph regards each device’s wireless transmission range as a circle with equal (for example, unit-length) radius. There is an edge between two devices only if those two devices are within each other’s transmission range.

In this research, we assume that the network devices (nodes) have different residual energy. In fact, even the network nodes have the same residual energy at the start, they are likely to have different residual energy after some time of operation due to the uneven workload distribution. Theoretically, to prolong the overall network lifetime, we do not expect the network nodes with less residual energy to be the hub nodes (which have many neighbor nodes); whereas, we anticipate the network nodes with more residual energy to serve as the hub nodes. That is to say, the more residual energy a network node has, the more neighbor nodes it should possess; in contrast, if a network node has less residual energy, it should have a relatively small number of neighbor nodes. Based on the assumption, we introduce a statistic method-based topology evaluation algorithm (named ED-index algorithm) and further propose a novel energy-aware topology control algorithm (named EDTC algorithm). The proposed EDTC algorithm leverages the maximum spanning tree algorithm to establish an initial network topology. Then, EDTC utilizes the ED-index algorithm to re-introduce some edges into the initial topology to increase the network robustness. Different from many other energy-aware topology control algorithms, the EDTC algorithm does not require any location information. We also present a machine learning version of the proposed EDTC algorithm, which leverages the trained graph convolutional network (GCN) to predict the edges to be re-introduced into the maximum spanning tree topology. We summarize the contributions of this work as the following three points:

1. We propose an ED-index which can reflect whether a network topology satisfies our assumption and the extent;
2. We develop the EDTC algorithm, which leverages the maximum spanning tree algorithm and the proposed ED-index to optimize the network topology;
3. We present a GCN-based algorithm to imitate the initial EDTC through learning. We show that the GCN-based EDTC achieves satisfactory performance and much faster than the initial EDTC.

Algorithm	Algorithm Mode (and Information Type)	Location Info.	Residual Energy Info.	Method
MST	Centralized (global information)	*	*	Neighbor node selection
DT	Centralized (global information)	✓	✗	Neighbor node selection
GG	Centralized (global information)	✓	✗	Antenna power adjusting
IATC	Centralized (global information)	✗	✗	MST, antenna power adjusting
CTEF	Centralized (global information)	✗	✓	Clustering
EFTCG	Centralized (one-hop information)	✗	✓	Game theory, antenna power adjusting
LEACH	Distributed (k-hop information)	✗	✗	Clustering
EBTG	Distributed (one-hop information)	✗	✓	Game theory, antenna power adjusting
ECTC	Distributed (k-hop information)	✓	✗	MST, antenna power adjusting
LMST	Distributed (global/two-hop information)	*	✗	MST, neighbor node selection
ERTO	Distributed (one-hop information)	✓	✓	Antenna power adjusting
LTCA	Distributed (one-hop information)	✗	✗	Neighbor node selection

*: (✓/✗) depends on different scenarios.

Table 5.1: The Comparison of Different Topology Control Algorithms

The rest of this chapter is organized as follows. In Section 5.2, we review some existing topology control algorithms, including state-of-the-art algorithms, as well as classic algorithms. We formulate the network system model and describe the problem in Section 5.3. Section 5.4 introduces the proposed ED-index algorithm and the EDTC algorithm (w/o and w/ GCN). We analyze the experimental results in Section 5.5. Finally, we conclude our work in Section 5.6.

5.2 Existing Topology Control Algorithms

One of the straightforward but efficient approaches to reduce the redundant edges in a connected graph without destroying the connectivity is to generate a minimum/-maximum weight spanning tree (MST) of the original graph. The most famous MST algorithms are Kruskal's algorithm (1956) [69] and Prim's algorithm (1957) [94]. The

advantage of modeling a network topology control problem into an MST problem is that, if the costs (such as energy consumption or physical distance between nodes) of communication between each pair of wireless nodes are known, the MST-based topology control algorithm might significantly reduce the total cost of the whole network.

Delaunay triangulation (DT) algorithm (1934) [20] can also be used for topology control. The basic idea of Delaunay triangulation is to establish edges between nodes where no node is inside any triangle formed of the established edges. Delaunay also tends to maximize the minimum inner angles of the triangles. However, the edges established by the Delaunay triangulation may not exist in a unit disk graph, which means that some of the nodes may not be able to communicate with each other, but there could be an edge between them in the Delaunay triangulation. Another problem of Delaunay triangulation is that the communication and computation load of the whole network could be huge. The Gabriel graph (1969) [35] is a sub-graph of the Delaunay triangulation. In a Gabriel graph (GG), if we draw a circle using any pair of nodes as the endpoints of the line segment of the diameter, then no other nodes are inside this circle.

Besides the optimization of energy consumption, minimizing the interference in a WAIoT network is also important. For instance, when two nodes communicating with each other, the other nodes within either of both nodes' communication ranges might interfere. Li et al. take the low interference as a goal and proposed several interference-aware topology control (IATC) algorithms in terms of various criteria to measure the interference quality of a structure [76]. By considering the interference issue, their algorithms can reduce the number of re-transmissions.

Hong et al. proposed an energy forecast based clustering-tree topology control algorithm (CTEF) [46]. CTEF algorithm selects cluster heads at each round in terms of a synthesized cost function and their distance. The cost function is based on an energy model which estimates the difference between the ideal average residual energy and the actual average residual energy to obtain the average energy of network at the next round. Besides selecting cluster nodes, CTEF also chooses several non-cluster head nodes in each cluster to be relay nodes for multi-hop communication to reduce the burden of the cluster head. The experimental results proved that the CTEF algorithm could prolong the network lifetime by optimizing energy consumption. According to the assumptions of CTEF, each network node should be able to communicate with the base station. However, in most WAIoT networks, due to the limitation of communication range, not all the nodes have the ability to communicate

with the base station directly.

The low-energy adaptive clustering hierarchy (LEACH) algorithm is a dynamic scheme that allows different nodes to be chosen as cluster-heads at different iterations (rounds) to prevent nodes from running out of energy quickly [44]. The cluster-heads of the current round cannot be cluster-heads in the next several rounds. As a distributed algorithm, LEACH enables each network node to decide whether or not to be the cluster-head independently. Each non-cluster-head node determines which cluster it should belong to in order to minimize the communication energy. The cluster-heads are required to aggregate and compress the data before forwarding the data to the base station. One issue of the LEACH algorithm is that the node residual energy is not considered for choosing the cluster-heads. Moreover, the distribution of cluster-heads is random; therefore the distribution could be uneven [124].

Du et al. proposed an energy balance topology control game algorithm (EBTG) based on game theory [24]. In EBTG, the Thiel-index, which is primarily used in measuring economic inequality, is applied in a unity function to measure the competition between nodes and the balance of energy consumption. The goal of EBTG is to maximize the unity function by allowing each participant node adjusts power in a selfish manner. EBTG has an adaptation phase and a topology maintenance phase. Each node determines its transmit power in the adaptation phase; topology maintenance phase mitigates the imbalance between nodes dynamically.

Based on the idea of minimum spanning tree, the local minimum spanning tree topology control algorithm (LMST) was proposed in [74]. LMST models the transmission power between each pair of nodes as the weight of the corresponding edge. Therefore, the result derived by LMST is power efficient. However, in LMST, each node needs to build its LMST independently at the start, which is not very flexible for future maintenance of the network topology. Moreover, because many redundant paths are eliminated by LMST, if one or more nodes die, the network may suffer from malfunction for a while. This problem also pointed out by the authors.

Most topology control algorithms consider energy efficiency but fail to take the optimization of end-to-end capacity between different nodes into consideration. In [40], the authors proposed a multi-objective topology control algorithm (ECTC), which jointly optimizes the network capacity and energy efficiency. The ECTC algorithm is based on a localized minimum spanning tree. ECTC also considers the heterogeneity of network nodes, such as the nodes might have different computing power, battery level, bandwidth, and path loss. One drawback is that ECTC assumes that each node

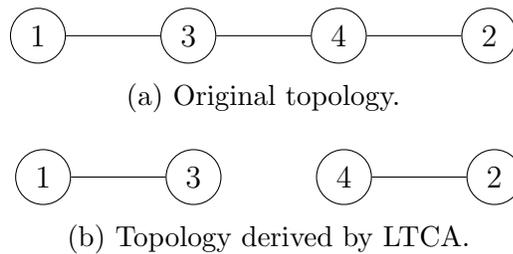


Figure 5.1: An example topology graph with four wireless network nodes before (a) and after (b) the application of LTCA. The value on each node represents the node ID.

is equipped with the Global Position System (GPS), which is expensive and not very accurate.

In [75], Li et al. introduced a topology control based opportunistic WSNs routing algorithm (ERTO). They proposed a packet delivery ratio between source node and candidate set calculation model (PDRsc) that considers the network interference. The topology control part of ERTO is a multi-objective optimization, which optimizes PDRsc, the expected energy consumption, and the degree of relay nodes. However, similar to ECTC, ERTO also requires each node to know its geolocation, which is usually done with GPS technology.

The localized topology control algorithm (LTCA) [53] only considers node IDs, which requires a meager amount of information exchange between nodes, thus conserves the energy consumption for topology construction. In LTCA, each node broadcasts its ID (assume the ID is unique) to its neighbor nodes, and also records the received IDs of the neighbor nodes. Later, each node decides whether or not to keep the connection with each neighbor node by comparing the IDs. However, on a general undirected graph, LTCA may not be able to preserve the connectivity. For instance, in Figure 5.1, the original graph is connected (see Figure 5.1a), whereas the graph derived by LTCA is not connected (see Figure 5.1b).

In [25], a state-of-the-art non-cooperative game-based algorithm EFTCG was proposed to establish energy-efficient topology for wireless sensor networks. EFTCG leverages a dedicated utility function to balance the transmit power, residual energy, and network connectivity. In EFTCG, each node is modeled as a player who tends to maximize its benefit. There are two phases in EFTCG, topology information collection phase, and topology game phase. Players use their maximum transmission power to broadcast “Hello Message” to their neighbors and initialize strategy sets

regarding the collected information in the first phase. The second phase lasts for many iterations to guarantee that each player can converge to the Nash equilibrium state.

A comparison of the algorithms mentioned above is in Table 5.1, whereas there are many other topology control algorithms for WAIoT networks, which are summarized in [12]. From previous literature (not limited to topology control algorithms), the centralized algorithm usually achieves a better result than its distributed counterpart. That is because a centralized algorithm has omniscience of the global information, which allows the algorithm to find the near global-optimal. A drawback of the centralized algorithm is that the number of message transmissions could be huge, especially when there are hundreds, even thousands of nodes. Moreover, to preserve connectivity, even distributed algorithms require many message transmissions.

5.3 System Model and Problem Statement

In this research, we assume that the network system has two types of wireless nodes: normal nodes and sink nodes. Normal nodes could be wireless sensor nodes, or other types of wireless IoT devices. We use $V^* = \{v_1^*, v_2^*, \dots, v_n^*\}$ to represent the set of normal nodes, where n is the number of normal nodes ($n = |V^*|$). Similarly, we use $V^+ = \{v_1^+, v_2^+, \dots, v_m^+\}$ to represent the set of sink nodes, where m denotes the number of sink nodes. The set of all network nodes is $V = V^* \cup V^+ = \{v_1, v_2, \dots, v_{n+m}\}$. To eliminate redundant nodes, we set the minimum distance between each pair of normal nodes to be \mathfrak{d} . Therefore,

$$\forall \{v_i^*, v_j^*\} \subseteq V^*, d_{v_i^*, v_j^*} \geq \mathfrak{d}.$$

Assume that every node (normal node or sink node) in V has an identical maximum communication range \mathfrak{r} . Then, there is an edge between v_i and v_j (denoted by e_{v_i, v_j}) if the Euclidean distance between v_i and v_j (denoted by d_{v_i, v_j}) is less than or equal to \mathfrak{r} . We use E to represent the set of edges, then,

$$\forall e_{v_i, v_j} \in E, d_{v_i, v_j} \leq \mathfrak{r}.$$

The residual energy of node v_i is denoted by \mathfrak{E}_{v_i} . We assume that the sink node has a stable power supply, therefore,

$$v_i \in V^+ \Rightarrow \mathfrak{E}_{v_i} = +\infty.$$

We represent the original network topology (assume that every node uses its maximum transmission power) as an undirected graph $G = (V, E)$. If we have the accurate location of each node, building a global topology graph will be straightforward. However, accurate positioning is expensive and faces many technical challenges (for instance, the accuracy of consumer-level GPS [134] is not acceptable in many topology control scenarios), we do not require the location information of each network node. In this case, network nodes can find their neighbors by broadcasting “Hello Messages” with their IDs (the ID should be unique, such as MAC address). We denote the set of neighbor nodes of v_i as \mathcal{N}_{v_i} , then,

$$\forall v_i \in V, v_j \in \mathcal{N}_{v_i} \Rightarrow e_{v_i, v_j} \in E.$$

We define the node degree of v_i as \mathfrak{D}_{v_i} . In an undirected graph G , the node degree of v_i equals to the number of its neighbor nodes $|\mathcal{N}_{v_i}|$ ($\mathfrak{D}_{v_i} = |\mathcal{N}_{v_i}|$).

We use v_{tx} and v_{rx} to represent the transmitter node and the receiver node, respectively. The received signal power by v_{rx} 's antenna is denoted by p_{rx} , which can be estimated through the Friss wireless transmission formula [33]:

$$p_{rx} = \frac{p_{tx} g_{tx} g_{rx} \lambda^2}{(4\pi d_{v_{tx}, v_{rx}})^2}, \quad (5.1)$$

where p_{tx} is the transmitter's antenna power; g_{tx} and g_{rx} represent the antenna gain of the transmitter and the receiver respectively; λ is the wavelength of the wireless signal. The relationship between λ and the channel frequency φ is $\lambda \approx (3.0 \times 10^8)/\varphi$. Assume the bandwidth is ϑ , and the additive white Gaussian noise of the channel is σ , we use the following equation to estimate the data transmission rate (bits/s):

$$r_{tx, rx} = \vartheta \log_2 \left(1 + \frac{p_{rx}}{\sigma} \right). \quad (5.2)$$

We have the same assumptions as mentioned in Chapter 4 (Equation 4.2), for using Equation 5.2. If the data transmission rate is fixed (to ensure the communication efficiency), we can derive the required p_{rx} through Equation 5.2. Further, through

Equation 5.1, we can estimate the required transmission power p_{tx} :

$$p_{tx} = \frac{p_{rx}(4\pi d_{v_{tx},v_{rx}})^2}{g_{tx}g_{rx}\lambda^2}. \quad (5.3)$$

Define the size of data as $\bar{\delta}$. According to Equation 5.2, the transmission time for the data of $\bar{\delta}$ is

$$t(\bar{\delta}) = \frac{\bar{\delta}}{r_{tx,rx}} = \frac{\bar{\delta}}{\vartheta \log_2(1 + \frac{p_{rx}}{\sigma})}. \quad (5.4)$$

Through Equation 5.3 and Equation 5.4, we can estimate the transmitter's power consumption (in joule) on transmitting the data of size $\bar{\delta}$ to the receiver:

$$\mathcal{J}(\bar{\delta}) = p_{tx} \times t(\bar{\delta}) = \frac{p_{rx}(4\pi d_{v_{tx},v_{rx}})^2 \bar{\delta}}{g_{tx}g_{rx}\lambda^2 \vartheta \log_2(1 + \frac{p_{rx}}{\sigma})}. \quad (5.5)$$

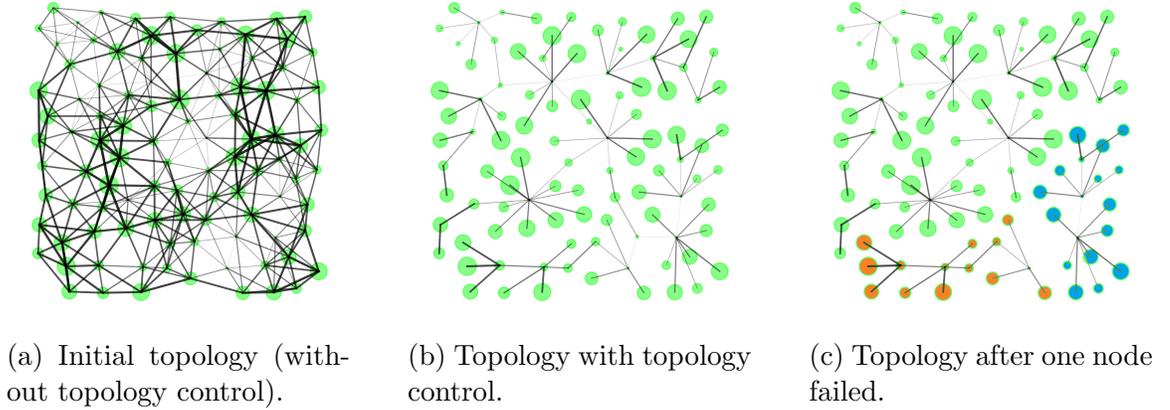


Figure 5.2: A demonstration of unsatisfactory network topologies. The size of the node has a positive correlation with the corresponding node residual energy.

The common objective of most topology control algorithms is to reduce the redundancy (to find a $G' = (V', E')$, where $V' \subseteq V$, and $E' \subseteq E$) and, as a result, prolong the network lifetime. Whereas, if we allow a normal node with relatively low residual energy to have more neighbors, it will speed up the energy consumption of that node, and could impact on the network connectivity once that node fails (run out of energy). For example, Figure 5.2a is the original topology, where the average node degree is high (node degree equals to the number of neighbor nodes). In this topology, nodes need to use their maximum transmission power, and each of the nodes needs to communicate with a large number of neighbor nodes. Therefore, the original topology is not efficient. Suppose Figure 5.2b shows the topology after

removing some redundant edges from Figure 5.2a. In this topology, the average node degree is much lower. However, if some critical nodes failed, the network will be disconnected. Figure 5.2c demonstrates the situation when one node in Figure 5.2b failed, the network became three disconnected sub-networks.

We assume that besides reducing the redundancy of network topology, we should also let the node with more residual energy to have a larger node degree, and let the node with less residual energy to have a smaller node degree. In different networks, the distribution of network node residual energy could be different. Therefore, it is tricky to design a standard for general network topologies. Whereas, if there are enough network nodes (enough sample size), we can leverage statistic methods in optimizing the network topology.

5.4 The Proposed Approach

The proposed topology control algorithm is based on our assumption that the node with relatively more residual energy should have a larger node degree; on the opposite, the node with relatively less residual energy should have a smaller node degree. In this section, we first introduce a statistic method-based algorithm (we name it energy distribution index or ED index) to evaluate whether and what is the extent a network topology meets our objective. Then, we present a maximum spanning tree-based baseline topology control algorithm (in the rest of this chapter, we use MST to represent this algorithm). In our proposed topology control algorithm (we name it energy-degree topology control or EDTC), we leverage the ED index to associate a small number of edges to the topology derived by the MST-based baseline algorithm, to increase the robustness of the network topology. We also introduce a GCN-based algorithm to imitate the EDTC algorithm, which could reduce the topology optimization time. In the rest of this chapter, we use EDTC w/o GCN to represent the initial EDTC algorithm and use EDTC w/ GCN to represent the GCN-based algorithm.

5.4.1 Energy Distribution Index

To measure to what extent a network topology meets our objective, we propose a statistic method-based metric named energy distribution index (ED index). Because the objective (more residual energy, larger node degree, and vice-versa) is based on our assumption that a network topology which meets the objective will have a longer

lifetime, we prove its validity through extensive simulation experiments in Section 5.5.

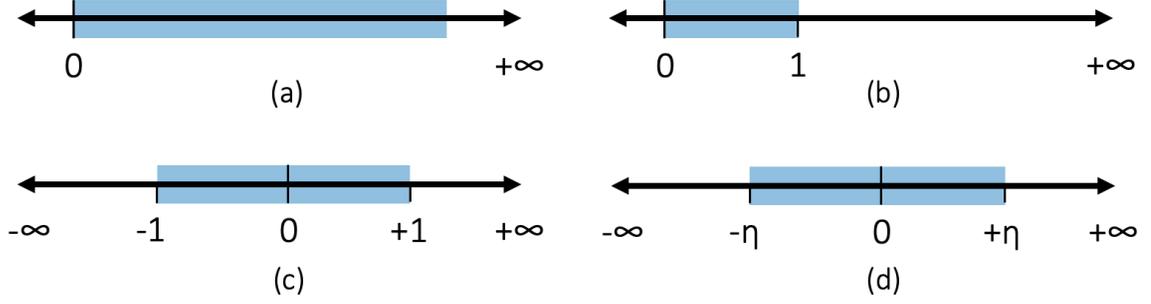
For different networks, the distributions of node residual energy and node degree are different. Thus, we need to normalize them to a controllable value range. We do not take the sink nodes into consideration in our ED index, that is because the normal node residual energy scale does not apply on the sink nodes ($\mathfrak{E}_{v_i^+} = +\infty$). Assume x_{min} and x_{max} denote the minimum and maximum values among all samples. For a given value x , we compute its min-max normalized value through the following formula,

$$z(x, x_{min}, x_{max}) = \frac{x - x_{min}}{x_{max} - x_{min}}. \quad (5.6)$$

Take the node residual energy as an example. Figure 5.3a shows the visualization of the original value range. First, we normalize the node residual energy value range to $[0, 1]$ (see Figure 5.3b) through Equation 5.6. We further normalize the range to $[-1, 1]$ (see Figure 5.3c) via subtracting each value by the mean. We also define a positive value $\eta > 0$ as the scale coefficient. Through multiplying each value by η , we can extend/shrink the value range to $[-\eta, \eta]$ (see Figure 5.3d). The choice of η 's value is not important. However, it should be noted that, when we compare the ED indices of different spanning graphs, the value of η should be consistent. We use the same method to normalize the value range of node degree to $[-\eta, \eta]$. Denote the normalized node residual energy as $\tilde{\mathfrak{E}}_{v_i^*}$, and the normalized node degree as $\tilde{\mathfrak{D}}_{v_i^*}$, then, $\tilde{\mathfrak{E}}_{v_i^*}, \tilde{\mathfrak{D}}_{v_i^*} \subseteq [-\eta, \eta]$.

After normalization, for each node v_i^* , we multiply its normalized node residual energy and normalized node degree together. The reason is that, if $\tilde{\mathfrak{E}}_{v_i^*} \rightarrow \eta$ and $\tilde{\mathfrak{D}}_{v_i^*} \rightarrow \eta$, then $\tilde{\mathfrak{E}}_{v_i^*} \times \tilde{\mathfrak{D}}_{v_i^*}$ is a relatively large positive value. Similarly, if $\tilde{\mathfrak{E}}_{v_i^*} \rightarrow -\eta$ and $\tilde{\mathfrak{D}}_{v_i^*} \rightarrow -\eta$, then $\tilde{\mathfrak{E}}_{v_i^*} \times \tilde{\mathfrak{D}}_{v_i^*}$ still is a relatively large positive value. On the contrary, if $\tilde{\mathfrak{E}}_{v_i^*} \rightarrow \eta$ and $\tilde{\mathfrak{D}}_{v_i^*} \rightarrow -\eta$, or $\tilde{\mathfrak{E}}_{v_i^*} \rightarrow -\eta$ and $\tilde{\mathfrak{D}}_{v_i^*} \rightarrow \eta$, then $\tilde{\mathfrak{E}}_{v_i^*} \times \tilde{\mathfrak{D}}_{v_i^*}$ will be a relatively small negative value. We compute the mean value of the multiplications, and theoretically, the more a network topology satisfies our objective, the larger the mean value is. Finally, we use *sigmoid* function to normalize the mean value to $[0, 1]$, which is the final ED index.

Algorithm 3 depicts the proposed ED index algorithm. The time complexity of each step in Algorithm 3 is $O(n)$. Therefore, the overall time complexity of the ED index algorithm is $O(n)$.



© Peizhi Yan

Figure 5.3: The demonstration of our normalization process. Blue bar represents the value range.

Algorithm 3: The ED index algorithm

Input: The topology: G ; η

Result: The ED index of a given topology G : $ED(G, \eta)$

```

1 // STEP 1: normalize the value ranges to [0,1]
2  $\mathfrak{E}_{min}, \mathfrak{E}_{max} \leftarrow$  the minimum and maximum node residual energy of  $V^*$ ;
3  $\mathfrak{D}_{min}, \mathfrak{D}_{max} \leftarrow$  the minimum and maximum node degree of  $V^*$ ;
4 foreach  $v_i^* \in V^*$  do
5   |  $\tilde{\mathfrak{E}}_{v_i^*} \leftarrow z(\mathfrak{E}_{v_i^*}, \mathfrak{E}_{min}, \mathfrak{E}_{max})$ ;
6   |  $\tilde{\mathfrak{D}}_{v_i^*} \leftarrow z(\mathfrak{D}_{v_i^*}, \mathfrak{D}_{min}, \mathfrak{D}_{max})$ ;
7 // STEP 2: further normalize the value ranges to [-1,1]
8  $\tilde{\mathfrak{E}}_{mean} \leftarrow$  the mean of  $\tilde{\mathfrak{E}}_{v_i^*}$ , where  $v_i^* \in V^*$ ;
9  $\tilde{\mathfrak{D}}_{mean} \leftarrow$  the mean of  $\tilde{\mathfrak{D}}_{v_i^*}$ , where  $v_i^* \in V^*$ ;
10 foreach  $v_i^* \in V^*$  do
11   |  $\tilde{\mathfrak{E}}_{v_i^*} \leftarrow \tilde{\mathfrak{E}}_{v_i^*} - \tilde{\mathfrak{E}}_{mean}$ ;
12   |  $\tilde{\mathfrak{D}}_{v_i^*} \leftarrow \tilde{\mathfrak{D}}_{v_i^*} - \tilde{\mathfrak{D}}_{mean}$ ;
13 // STEP 3: extend/shrink the value ranges to  $[-\eta, \eta]$ 
14 foreach  $v_i^* \in V^*$  do
15   |  $\tilde{\mathfrak{E}}_{v_i^*} \leftarrow \eta \times \tilde{\mathfrak{E}}_{v_i^*}$ ;
16   |  $\tilde{\mathfrak{D}}_{v_i^*} \leftarrow \eta \times \tilde{\mathfrak{D}}_{v_i^*}$ ;
17 // STEP 4: calculate the ED index
18  $result \leftarrow 0$ ;
19 foreach  $v_i^* \in V^*$  do
20   |  $result \leftarrow result + \tilde{\mathfrak{E}}_{v_i^*} \times \tilde{\mathfrak{D}}_{v_i^*}$ ;
21  $result \leftarrow result/n$ ;
22 return  $sigmoid(result)$ ;
```

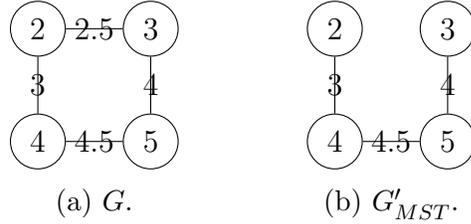


Figure 5.4: An example of the original topology G and the maximum spanning tree derived by the Kruskal's algorithm G'_{MST} . Value on each vertex represents the corresponding node residual energy; value on each edge represents the edge weight.

5.4.2 Energy-Degree Topology Control Algorithm

The proposed topology control algorithm is based on the maximum spanning tree algorithm and the proposed ED index. We name the proposed algorithm energy-degree topology control algorithm (EDTC) because it focuses on the balance between the node residual energy and node degree.

To utilize the maximum spanning tree algorithm to approach our objective, we need to define the weights of edges appropriately. We expect that an edge should have more chance to be in the maximum spanning tree if its two endpoints (network nodes) have more average residual energy. Thereby, we define the weight of an edge e_{v_i, v_j} as

$$w(e_{v_i, v_j}) = \frac{\epsilon_{v_i} + \epsilon_{v_j}}{2}. \quad (5.7)$$

We use Kruskal's algorithm to find the maximum spanning tree $G'_{MST} = (V, E'_{MST})$ of the original topology $G = (V, E)$, where $E'_{MST} \subseteq E$. The time-complexity of Kruskal's algorithm is $O(|E|\log|E|)$. Because we define the residual energy of sink nodes to be infinity, any edge in E , which connects a sink node, will be included in E'_{MST} . Figure 5.4 demonstrates a simple example of G and G'_{MST} . A problem with the maximum spanning tree topology is that there is only one path (in which all vertices are distinct) between any pair of nodes. In practice, we often allow some redundancy in the network topology (such as that there could be multiple paths between a pair of nodes), to enhance the robustness of the network. Therefore, in EDTC, we leverage the ED index to provide guidance of re-introducing some edges from E to E'_{MST} . We define a variable ϕ to represent the maximum number of edges to be re-introduced ($0 \leq \phi \leq |E| - |E'_{MST}|$). The actual number of re-introduced edges could be less than or equal to ϕ , depends on different situations. The detailed implementation of EDTC

Algorithm 4: The proposed EDTC (w/o GCN) algorithm

Input: The original topology: $G; \eta; \phi$
Result: G'_{EDTC}

- 1 // STEP 1: find the maximum spanning tree G'_{MST}
- 2 $G'_{MST} = (V, E'_{MST}) \leftarrow$ apply Kruskal's algorithm on $G = (V, E)$;
- 3 $E'_{EDTC} \leftarrow$ make a copy of E'_{MST} ;
- 4 $ed_{MST} \leftarrow ED(G'_{MST}, \eta)$;
- 5 // STEP 2: evaluate the edges in $E \setminus E'_{MST}$
- 6 $E_{temporary} \leftarrow E \setminus E'_{MST}$;
- 7 **foreach** $e_{v_i, v_j} \in E_{temporary}$ **do**
- 8 $E'_{EDTC} \leftarrow E'_{EDTC} \cup \{e_{v_i, v_j}\}$;
- 9 $G'_{temporary} \leftarrow (V, E'_{EDTC})$;
- 10 $w'(e_{v_i, v_j}) \leftarrow ED(G'_{temporary}, \eta)$;
- 11 $E'_{EDTC} \leftarrow E'_{EDTC} \setminus \{e_{v_i, v_j}\}$;
- 12 $E_{temporary} \leftarrow$ sort $E_{temporary}$ by weights w' in descending order;
- 13 // STEP 3: re-introduce edges to E'_{EDTC}
- 14 $counter \leftarrow 0$;
- 15 **foreach** $e_{v_i, v_j} \in E_{temporary}$ **do**
- 16 **if** $w'(e_{v_i, v_j}) \leq ed_{MST}$ **or** $counter \geq \phi$ **then**
- 17 **break the loop**;
- 18 $E'_{EDTC} \leftarrow E'_{EDTC} \cup \{e_{v_i, v_j}\}$;
- 19 $counter \leftarrow counter + 1$;
- 20 $G'_{EDTC} \leftarrow (V, E'_{EDTC})$;
- 21 **return** G'_{EDTC} ;

is shown in Algorithm 4. In STEP 1, we use Kruskal’s algorithm to find the maximum spanning tree $G'_{MST} = (V, E'_{MST})$ of the original topology graph $G = (V, E)$. We also make a copy of the set of edges E'_{MST} , and define it as E'_{EDTC} . We run the ED-index algorithm to evaluate the ED-index of G'_{MST} . In STEP 2, for each edge e_{v_i, v_j} that belongs to E but not in E'_{MST} , we temperately introduce it to E'_{EDTC} , and evaluate the ED-index of graph $G'_{EDTC} = (V, E'_{EDTC})$. We record the evaluated ED-index as the new weight of e_{v_i, v_j} , which is defined as $w'(e_{v_i, v_j})$. Then we remove e_{v_i, v_j} from E'_{EDTC} . By repeating the procedures, we can get the new weights of every edge that is in $E_{temporary} = E \setminus E'_{MST}$. We sort the edges in $E_{temporary}$ by their new weights in descending order. In STEP 3, we go over the edges in $E_{temporary}$. If the new weight of an edge is greater than the ED-index of G'_{MST} , and the number of re-introduced edges is less than the limit ϕ we set, we re-introduce the edge into E'_{EDTC} .

The STEP 1 in Algorithm 4 has the time-complexity of $O(|E|\log|E| + n)$. The time-complexity of STEP 2 is $O(n|E| + |E|\log|E|)$. The last step has the time-complexity of $O(\phi)$. Because $\phi < |E|$, the overall time-complexity of the proposed EDTC (w/o GCN) algorithm is $O(n|E| + |E|\log|E|)$.

5.4.3 Graph Convolutional Network Based Energy-Degree Topology Control Algorithm

Because the topology derived by EDTC (w/o GCN) is based on MST, we use $G^\dagger = (V, E^\dagger)$, where $E^\dagger = E'_{EDTC} \setminus E'_{MST}$, to represent the graph of the re-introduced links. Inspired by [78], we design and train a GCN model to predict the probability map which leads us to generate the approximate set \hat{E}^\dagger (the set of edges to be added to G'_{MST}). For convenience, we list some important notations used in this sub-section in Table 5.2. The weights in A are the min-max normalized edge weights of G . Each vertex has two features, min-max normalized residual energy and vertex degree. Therefore, $C = 2$. We use a fixed number of filters in each GCN layer, $F = 32$. To get a large receptive field, we use 20 graph convolution layers ($L = 20$) in our GCN model. We use *ReLU* as the activation function for each graph convolution layer except the last layer. The output of the first graph convolution layer is $H^{(1)} = ReLU(\tilde{S}XW^{(0)})$; the output of the last GCN layer is $H^{(L)} = \tilde{S}H^{(L-1)}W^{(L-1)}$. For each graph convolution layer, $W^{(l)} \in \mathbb{R}^{F \times F}$, except for $W^0 \in \mathbb{R}^{C \times F}$. We use the similar method used in graph auto-encoder model [66] to reconstruct the output to

Notation	Description
–	Element-wise matrix subtraction operation
◦	Element-wise matrix multiplication operation
⊤	Matrix transpose operation
A	The weighted adjacency matrix of G
\dot{A}	The binary adjacency matrix of G
$\dot{A}^{(MST)}$	The binary adjacency matrix of G'_{MST}
$\dot{A}^{(EDTC)}$	The binary adjacency matrix of G'_{EDTC}
$\dot{A}^\dagger = \dot{A}^{(EDTC)} - \dot{A}^{(MST)}$	The binary adjacency matrix of G^\dagger
\tilde{S}	The normalized graph Laplacian of G based on A
C	The number of features of each vertex
F	The number of filters in each GCN layer
L	The number of GCN layers
X	The matrix of vertex feature
$W^{(l)}$	The filter weight matrix at the $(l+1)^{th}$ GCN layer
$H^{(l)}$	The output of the l^{th} GCN layer

Some notations already defined previously, some notations are the first time appear.

Table 5.2: Some Mathematical Notations

$\tilde{M} \in \mathbb{R}^{|V| \times |V|}$:

$$\tilde{M} = \text{sigmoid}(H^{(L)}H^{(L)\top}). \quad (5.8)$$

To eventually use \tilde{M} as the probability map, we need to add some constraints to it. Some edges of G are already in G'_{MST} , and we do not need to get the probability of those edges. Therefore, the probability map \hat{M} is defined as:

$$\hat{M} = \tilde{M} \circ (\dot{A} - \dot{A}^{(MST)}). \quad (5.9)$$

Define $\theta = \{W^{(l-1)} | 1 \leq l \leq L, l \in \mathbb{Z}^+\}$ as the set of trainable parameters. We then use $\mathcal{F}_\theta(A, \dot{A}, \dot{A}^{(MST)}, X) = \hat{M}$ to represent our neural network model as a function,

where $(A, \dot{A}, \dot{A}^{(MST)}, X)$ is the neural network input, and \hat{M} is the output. We use \dot{A}^\dagger as the target to define the loss function:

$$\mathcal{L} = \frac{\sum_{i=1}^{|V|} \sum_{j=1}^{|V|} (\hat{M}_{ij} - \dot{A}_{ij}^\dagger)^2}{\sum_{i=1}^{|V|} \sum_{j=1}^{|V|} (\dot{A}_{ij} - \dot{A}_{ij}^{(MST)})}. \quad (5.10)$$

Algorithm 5: The proposed EDTC (w/ GCN) algorithm

Input: The original topology: G ; ϕ
Result: $G'_{EDTC-GCN}$

- 1 // STEP 1: generate the MST of G
- 2 $G'_{MST} \leftarrow$ apply Kruskal’s algorithm on G ;
- 3 // STEP 2: run the GCN model
- 4 $(A, \dot{A}, \dot{A}^{(MST)}, X) \leftarrow$ prepare the neural network input;
- 5 $\hat{M} \leftarrow \mathcal{F}_\theta(A, \dot{A}, \dot{A}^{(MST)}, X)$;
- 6 // STEP 3: sort the link-level probabilities
- 7 $list \leftarrow$ create a list;
- 8 **foreach** $e_{v_i, v_j} \in E$ **do**
- 9 **if** $\hat{M}_{i,j} > 0$ **then**
- 10 $list \leftarrow list \cup \{(\hat{M}_{i,j}, v_i, v_j)\}$;
- 11 $list \leftarrow$ sort the $list$ in descending order in terms of the first element of each triplet;
- 12 // STEP 4: re-introduce edges to G'_{MST}
- 13 $counter \leftarrow 0$;
- 14 **foreach** $(\hat{M}_{i,j}, v_i, v_j) \in list$ **do**
- 15 **if** $counter \geq \phi$ **then**
- 16 **break the loop**;
- 17 $G'_{MST} \leftarrow$ add edge e_{v_i, v_j} to G'_{MST} ;
- 18 $counter \leftarrow counter + 1$;
- 19 $G'_{EDTC-GCN} \leftarrow G'_{MST}$;
- 20 **return** $G'_{EDTC-GCN}$;

We run extensive simulations to generate a large set of graphs \mathcal{G} . For each graph $G \in \mathcal{G}$, we generate G'_{MST} and G'_{EDTC} . Eventually, we get an input 4-tuple $(A, \dot{A}, \dot{A}^{(MST)}, X)$, and a target \dot{A}^\dagger for each $G \in \mathcal{G}$. We train the neural network model through gradient descent and backpropagation-based optimizer to minimize the loss \mathcal{L} . The proposed GCN-based EDTC algorithm (EDTC w/ GCN) leverages the trained GCN \mathcal{F}_θ to predict a link-level probability map \hat{M} , and add some edges to the maximum spanning tree in terms of \hat{M} . Algorithm 5 shows the detail of the GCN-based EDTC. The time-complexity for each step is $O(|E|\log|E|)$, $O(|E|\log|E|)$,

$O(|E|\log|E|)$, and $O(\phi)$ respectively. Therefore, the overall time-complexity of Algorithm 5 is $O(|E|\log|E|)$.

Define k as the average degree of network nodes, then $|E| = \frac{kn}{2}$. Therefore, the time-complexity of Algorithm 4 can be written as $O(kn^2 + kn \cdot \log(kn))$, and the time-complexity of Algorithm 5 can be written as $O(kn \cdot \log(kn))$. In the worst case, when the initial topology is a complete graph, $|E| = \frac{n^2-n}{2}$. Because $\lim_{n \rightarrow +\infty} [\log(\frac{n^2-n}{2})/n] = 0$, $O(\log(|E|)) = O(\log(\frac{n^2-n}{2})) \in O(n)$. In this situation, the time-complexity of both Algorithm 4 and Algorithm 5 is $O(n|E|) \in O(n^3)$. This means that, only when $k \rightarrow n - 1$, the time-complexities of both versions of EDTC algorithm will be same. However, in practice, if $n \rightarrow +\infty$, $k \ll n$. Thus, $O(kn^2 + kn \cdot \log(kn)) \in O(n^2 + n\log(n)) \in O(n^2)$, and $O(kn \cdot \log(kn)) \in O(n\log(n))$. Therefore, the time-complexity of the GCN-based EDTC algorithm (Algorithm 5: $O(n\log(n))$) is lower than the initial EDTC algorithm (Algorithm 4: $O(n^2)$).

To better explain the GCN-based EDTC, we visualize an example in Figure 5.5. Figure 5.5b is an example graph G with 50 normal nodes, and Figure 5.5a is the weighted adjacency matrix of G . Figure 5.5c is the support matrix \tilde{S} . The predicted link-level probability map \hat{M} is shown in Figure 5.5e. In Figure 5.5d, the black edges represent the edges of G'_{MST} ; and the widths of the red edges are proportional to the probabilities in \hat{M} . Figure 5.5f is the topology derived by the GCN-based EDTC.

5.5 Experiments

We demonstrate the viability of the proposed ED index algorithm through a straightforward experiment. Denote the randomly generated original network topology as G ($n = 100$, $m = 0$). The edge weights are calculated through Equation 5.7. The maximum spanning tree and the minimum spanning tree of G are denoted as G'_{max} and G'_{min} , respectively. Figure 5.6 shows an example of G , G'_{max} , and G'_{min} . We randomly simulate 1,000 times, Figure 5.7 shows the distribution of the ED indices (curves are smoothed). Theoretically, the nodes with more residual energy in G'_{max} will have more neighbors, whereas the nodes with less residual energy in G'_{max} will have fewer neighbors. Therefore, G'_{max} should have a higher ED index than G . On the opposite, G'_{min} should have a lower ED index than G . The result in Figure 5.7 is consistent with our theoretical assumption. Because the curves in Figure 5.7 are bell-shaped, we assume the ED indices (from random simulation) obey normal distribution. We conduct the following goodness-of-fit test to verify this assumption.

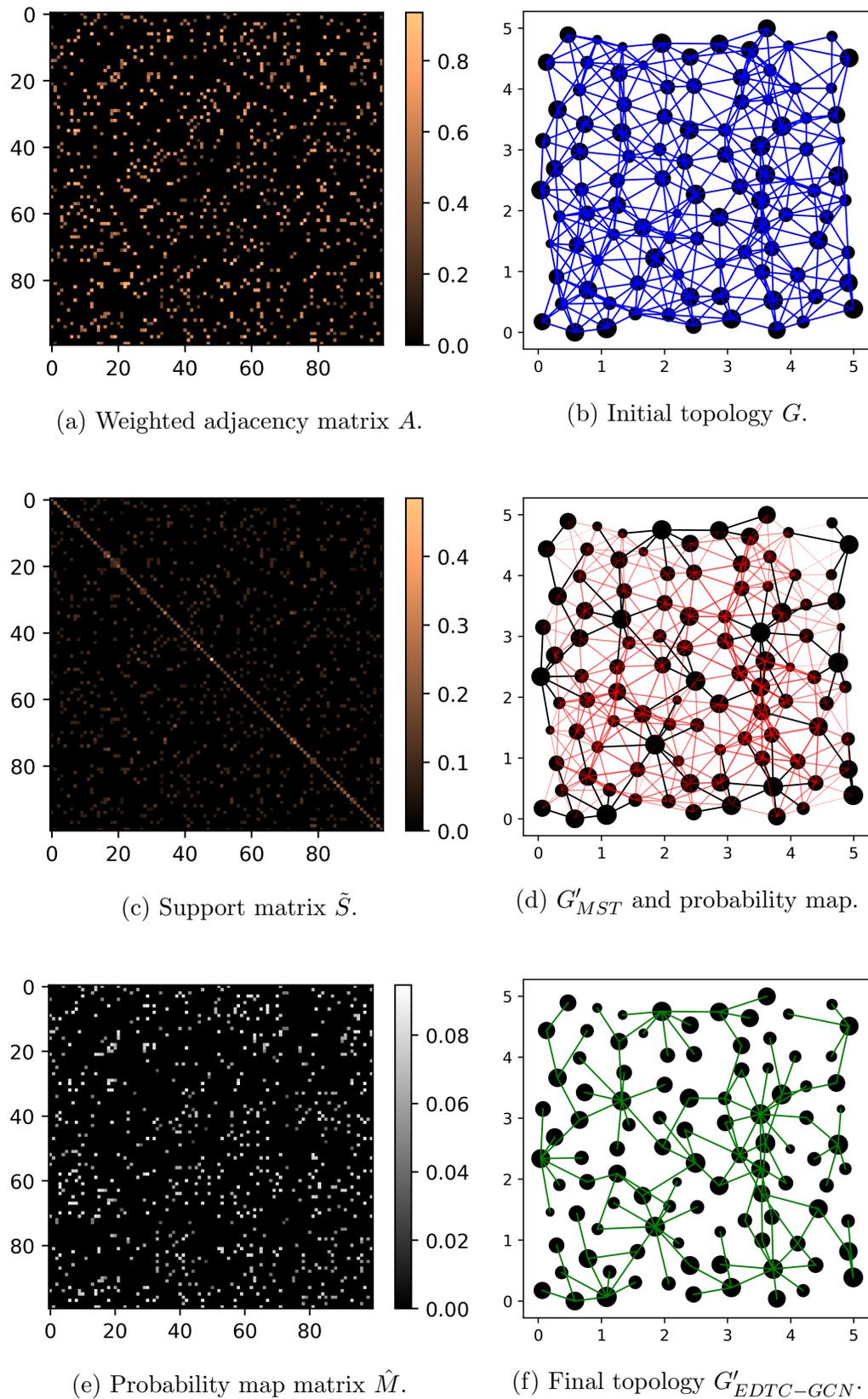


Figure 5.5: The visualization of some matrices and graphs.

First, we make the null-hypothesis H_0 : the ED indices are not consistent with the normal distribution. Accordingly, the alternative hypothesis H_A is the negate of H_0 . Then, we suppose the significance level is 0.05, which means that if the p-value is less than 0.05, we believe that the ED indices are not consistent with the normal distribution (reject H_0 and accept H_A). We perform the D'Agostino's K^2 test and the Shapiro-Wilk test separately (see Table 5.3). Regarding the goodness-of-fit testing results, we cannot reject H_0 . Hence, we conclude that the ED index is very likely to follow a normal distribution.

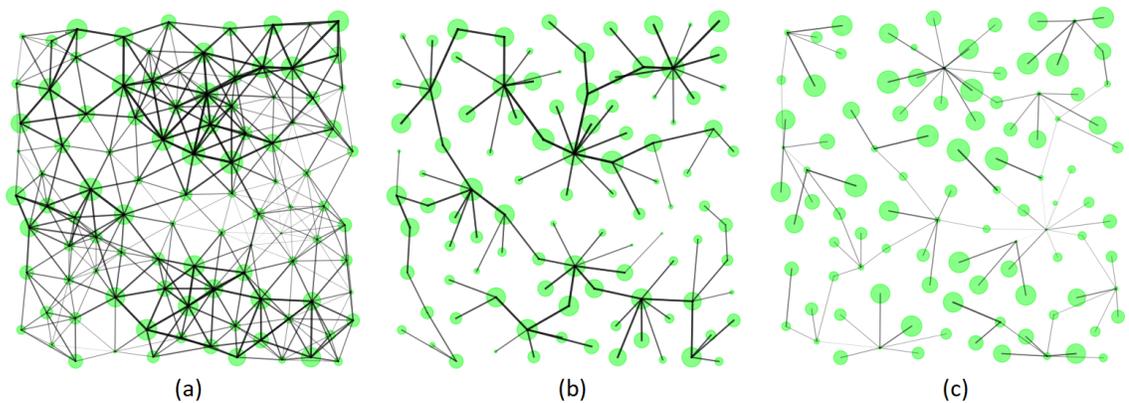


Figure 5.6: Comparison of the original topology G (a), the maximum spanning tree G'_{max} (b), and the minimum spanning tree G'_{min} (c).

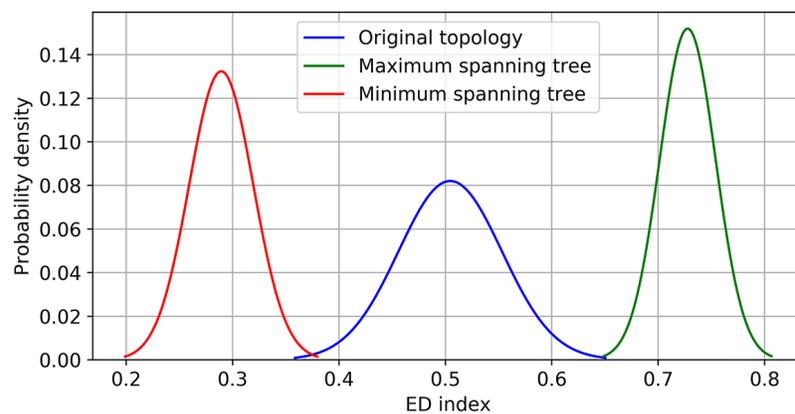


Figure 5.7: The distribution of ED index ($\eta = 5$) on 1,000 simulations.

D’Agostino’s K^2 Normality Test Results

Topology Type	Test Statistic	P-value	Interpretation
Original topology	1.35	0.508	Fail to reject H_0
LTCA	2.71	0.257	Fail to reject H_0
EFTCG	1.52	0.466	Fail to reject H_0
Minimum Spanning Tree	2.47	0.289	Fail to reject H_0
Maximum Spanning Tree	1.03	0.595	Fail to reject H_0
EDTC	1.74	0.418	Fail to reject H_0

Shapiro-Wilk Normality Test Results

Topology Type	Test Statistic	P-value	Interpretation
Original topology	0.911	0.292	Fail to reject H_0
LTCA	0.897	0.207	Fail to reject H_0
EFTCG	0.908	0.273	Fail to reject H_0
Minimum Spanning Tree	0.909	0.278	Fail to reject H_0
Maximum Spanning Tree	0.928	0.433	Fail to reject H_0
EDTC	0.926	0.418	Fail to reject H_0

Table 5.3: Goodness-of-Fit Test Results

In the following experiments, we simulate communication among network nodes on topologies derived by different topology control algorithms. Figure 5.8 shows the example topologies derived by different algorithms. We can find the potential issues that exist in LTCA and EFTCG, where some network nodes with low remaining energy have many connections or act as backbone nodes. We allow each node to adjust its antenna power (through Equation 5.2 and Equation 5.3) to ensure it can have a data transmission rate of r' with its farthest neighbor node. The sink node always communicates with its maximum antenna power (to ensure the minimum communication rate with any of its neighbor nodes is r'). We also assume that the minimum distance between any pair of nodes is d_{min} , to reduce node redundancy. Other experimental parameters are shown in Table 5.4. In addition, the GCN model used in our experiments is trained on 1,000 randomly simulated graphs ($n = 100, m = 0$).

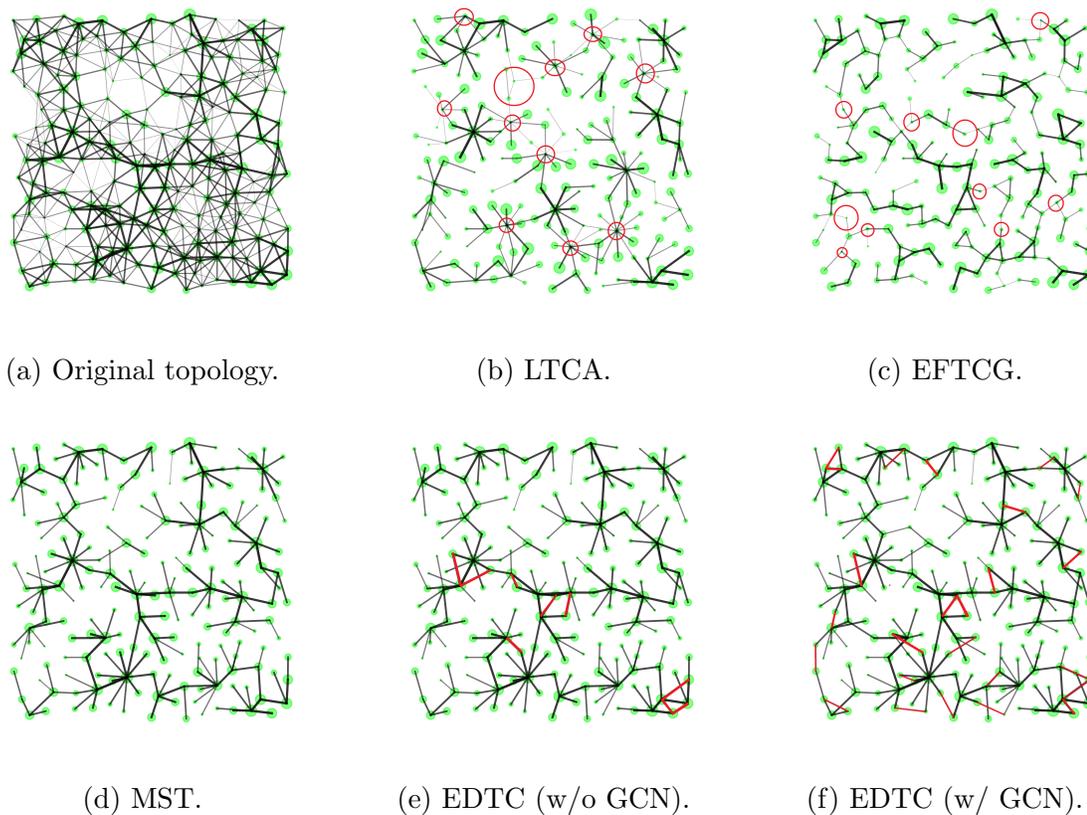


Figure 5.8: Topologies derived by different topology control algorithms. In (b) and (c), some potential issues are circled. In (e) and (f), the red edges represent the re-introduced edges.

Parameter	Value
Simulation area	$n \leq 200$: 80×80 (m ²) $200 < n \leq 800$: 180×180 (m ²)
Maximum communication range: \mathbf{r}	10 (m)
Residual energy: \mathfrak{E}	1.0 ~ 10.0 (J)
Minimum node distance: d_{min}	4 (m)
ϕ	$\lfloor 0.05 \times (E - E'_{MST}) \rfloor$
Antenna gain: g	-6 (dB)
Channel frequency: φ	916 (MHz)
Bandwidth: ϑ	1 (MHz)
Gaussian white noise: σ	-80 (dB)
Data size: \mathfrak{D}	1 (Mbits)
Minimum data transmission rate: r'	10 (Kbits/s)

Table 5.4: Experimental Parameters [4], [135]

In the second experiment, we assume that there is a sink node deployed at the center of the simulation area. In each time-step, every normal node in V^* sends a message of size s to the sink node. We suppose the length of a time step is enough for every normal node's message to be delivered to the sink node. Note that the length of each time-step could be different. As long as all the communications are complete, the simulation proceeds to the next time step. In each simulation, we randomly generate a network model and run 100 time steps to record the number of nodes (alive nodes) still can communicate with the sink node. For each experimental setting (the number of deployed normal nodes n), we run the simulation 100 times to compute the average number of alive nodes at each time-step. Figure 5.9 and Figure 5.10 show the experimental results. Take Figure 5.9f as an example, the number of alive nodes drops rapidly in the first 60 simulation time steps for both LTCA and EFTCG. For instance, at the 20th time step, the number of alive nodes for both LTCA and EFTCG are below 60%, whereas, our approaches still maintain more than 80% alive nodes. However, we can notice that after 60 simulation time steps, the numbers of alive nodes for all the approaches are similar. That is because we let all the alive nodes to send a message to the sink node in each time step. Therefore, the more nodes are still alive, the higher the overall consumption the network will have. We can also notice this effect by comparing the results, when the number of deployed network nodes increases, the curves are more skewed to the left. To compare the performance of the

Number of Nodes	LTCA	EFTCG	MST	EDTC w/o GCN	EDTC w/ GCN
$n = 50$	2661.0	2847.8	3303.7	3509.6	<u>3505.2</u>
$n = 60$	2890.4	3263.1	3692.9	<u>4019.6</u>	4055.9
$n = 70$	3264.3	3689.2	4114.3	<u>4627.7</u>	4672.2
$n = 80$	2616.8	2964.2	3426.0	3749.7	<u>3731.6</u>
$n = 90$	2802.5	3256.3	3778.3	<u>4170.1</u>	4203.2
$n = 100$	3034.4	3361.4	4028.1	<u>4479.3</u>	4560.3
$n = 110$	2663.0	3080.0	3308.5	3606.2	<u>3538.4</u>
$n = 120$	3031.0	3023.8	3563.1	4015.3	<u>3877.0</u>
$n = 130$	2933.5	3572.2	4089.2	4473.8	<u>4326.3</u>
$n = 140$	2724.5	2591.0	3304.9	3807.8	<u>3519.8</u>
$n = 150$	2683.6	2787.9	3293.7	3549.3	<u>3539.9</u>
$n = 160$	3018.0	3020.2	3590.5	3993.6	<u>3679.6</u>

The best results are in **bold** font; the second-best results are underlined.

Table 5.5: Area Under the Curve (The Second Experiment)

topology control algorithms quantitatively, we calculated the area-under-the-curve (see Table 5.5). A larger area-under-the-curve means that the more nodes are alive on average over the 100 time-steps. The proposed EDTC algorithm (w/ or w/o GCN) achieves the best performance under this metric. Figure 5.11 depicts the distribution of the ED indices of different typologies ($n = 100$), which further proved the viability of using the ED index as an evaluation metric.

We also collect the average node degree and the average topology construction time (see Figure 5.12). The proposed EDTC algorithm (w/ and w/o GCN) has a slightly higher average node degree than the other algorithms. However, in terms of Table 5.5, we conclude that the sacrifice in the average node degree helps to improve the network lifetime. Moreover, since the EFTCG algorithm is game-based, which requires multiple optimization iterations, the topology construction time is much higher than the other topology control algorithms (see Figure 5.12b, note that the vertical axis is log-scaled). The GCN-based EDTC is faster than the EDTC (w/o GCN) in our experiments.

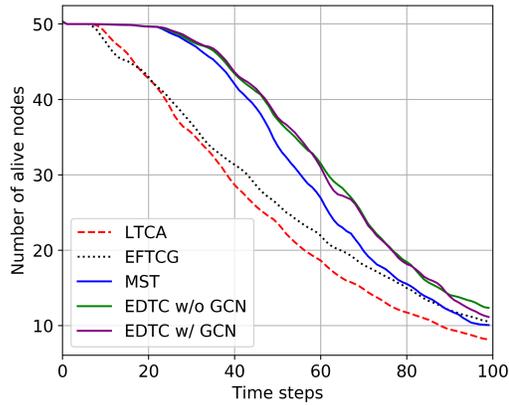
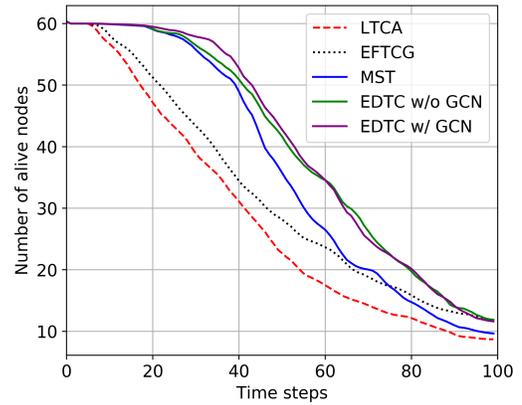
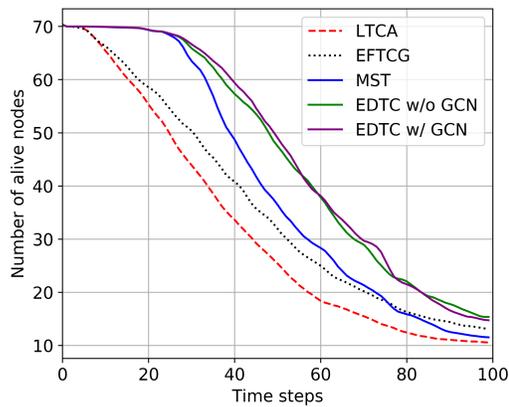
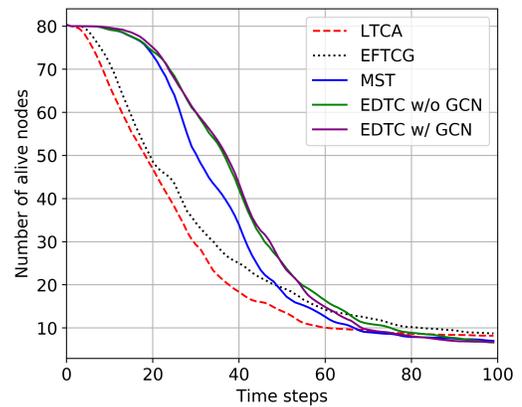
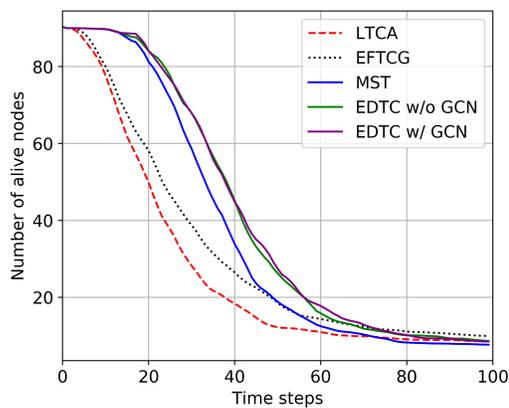
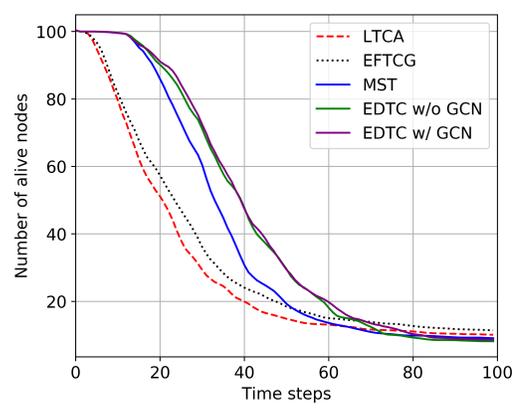
(a) $n = 50$.(b) $n = 60$.(c) $n = 70$.(d) $n = 80$.(e) $n = 90$.(f) $n = 100$.

Figure 5.9: The change of the number of alive nodes over 100 time-steps (the second experiment). $n \in \{50, 60, 70, 80, 90, 100\}$.

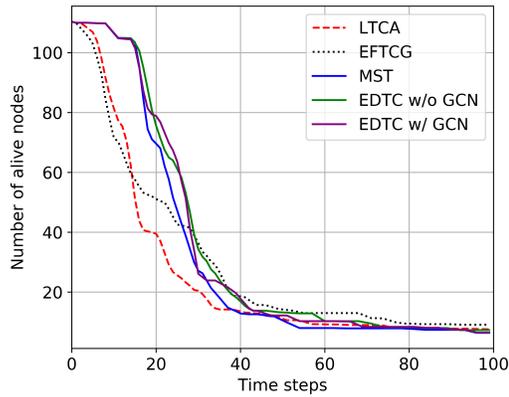
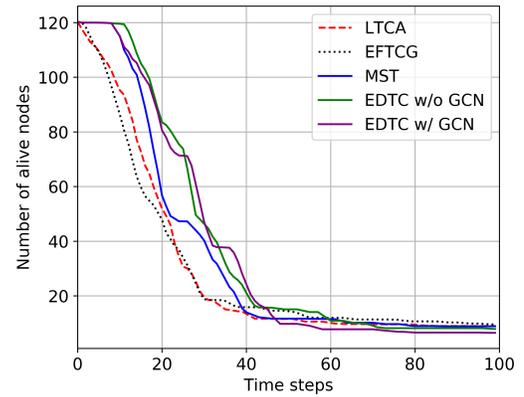
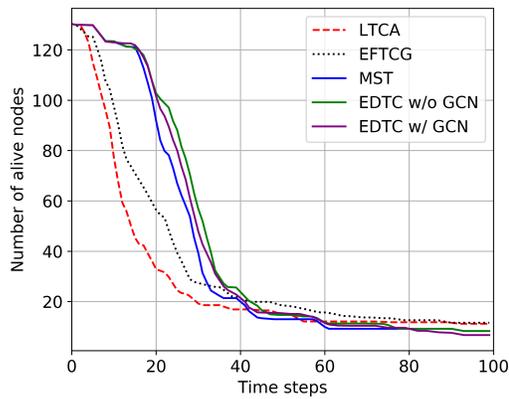
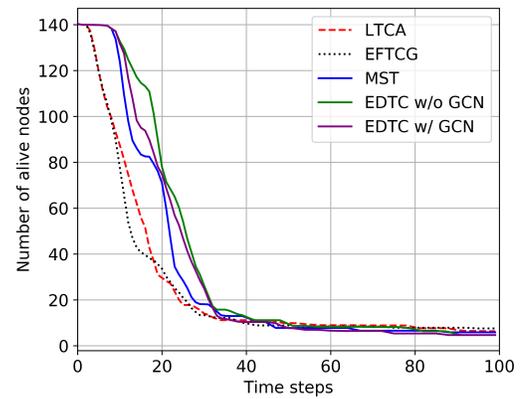
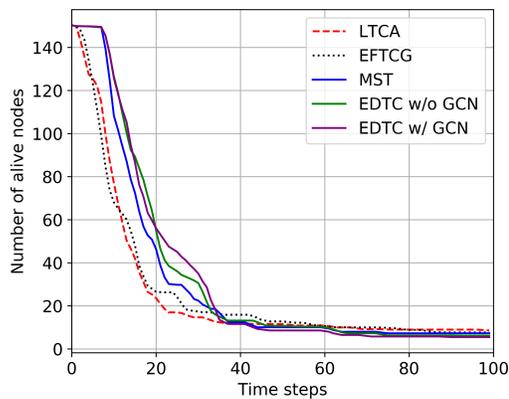
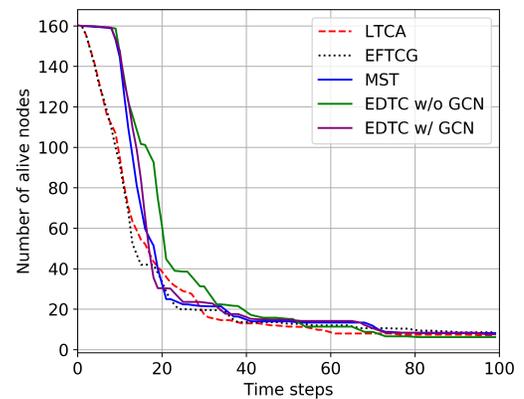
(a) $n = 110$.(b) $n = 120$.(c) $n = 130$.(d) $n = 140$.(e) $n = 150$.(f) $n = 160$.

Figure 5.10: The change of the number of alive nodes over 100 time-steps (the second experiment). $n \in \{110, 120, 130, 140, 150, 160\}$.

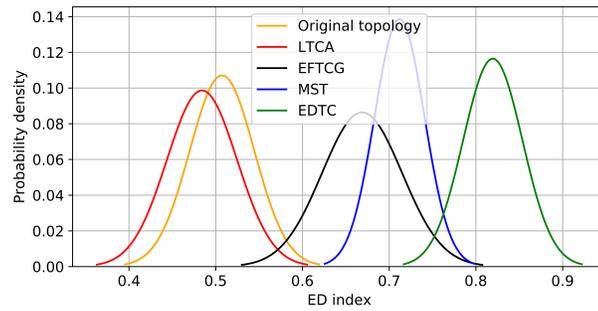
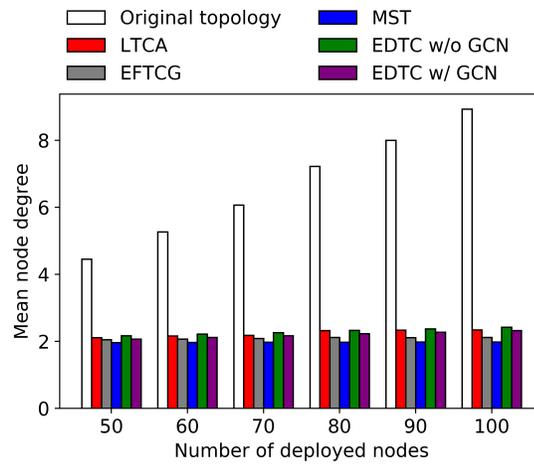
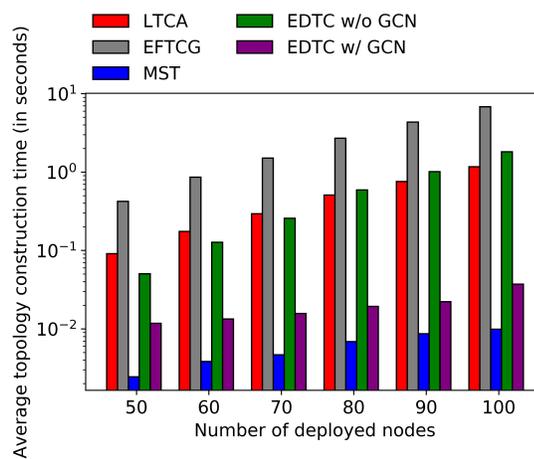


Figure 5.11: The distribution of ED indices ($\eta = 5$) of different topologies.



(a) Average node degree.



(b) Average topology construction time.

Figure 5.12: Bar plots of other experimental results.

In the third experiment, we do not deploy any sink node. We let each network node randomly communicate with one of the other network nodes in each time-step. The length of each time-step is still enough for the network nodes to complete the communication. Other experimental settings are the same as the second experiment. When some nodes run out of their energy, the network will be disconnected. Nevertheless, in practice, as long as the majority of the nodes are still connected, we continue the simulation on the largest connected sub-network. We define the connectivity threshold as τ , which represents the minimum allowed proportion of the number of nodes in the largest sub-network to the total number of network nodes in the original topology. For example, $\tau = 80\%$ means that if a network has 100 nodes initially, as long as there are more than 79 nodes are connected, we see the network to be alive and continue the simulation. The experimental results are shown in Figure 5.13 (smaller instances, $n \leq 200$) and Figure 5.14 (larger instances, $200 < n \leq 800$). For smaller instances, when the connectivity threshold is 90%, the proposed approaches can achieve up to around three times an extended network lifetime than LTCA and EFTCG. If we reduce the connectivity threshold, the proposed approaches still can achieve up to around two times an extended network lifetime than LTCA and EFTCG. However, the initial EDTC algorithm is not very stable for smaller instances. For example, sometimes, the topology generated by EDTC even get less network lifetime than the original maximum spanning tree topology. That is because the proposed EDTC leverages the statistical-based ED-index algorithm, and for smaller instances, the number of samples might not be enough. For larger instances, we can see that both versions of EDTC algorithms can always achieve a better result than the original maximum spanning tree algorithm. Under different threshold and number of deployed network nodes settings, the proposed EDTC algorithms achieve the most extended average network lifetime than the other algorithms.

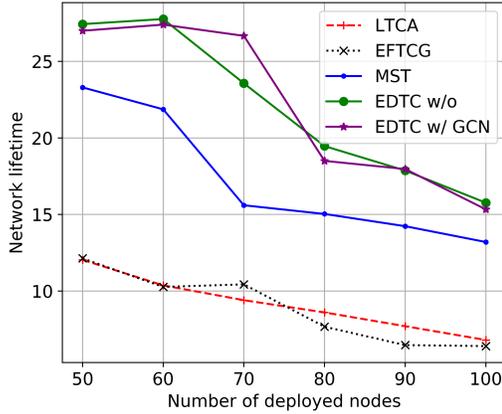
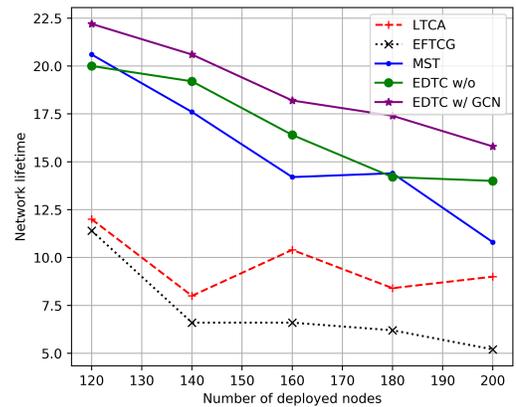
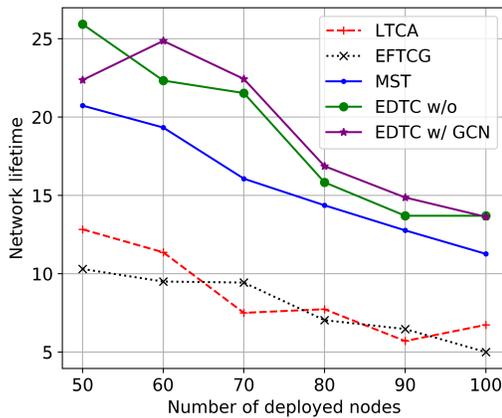
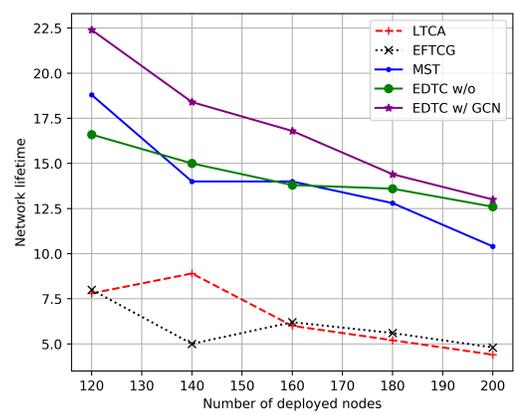
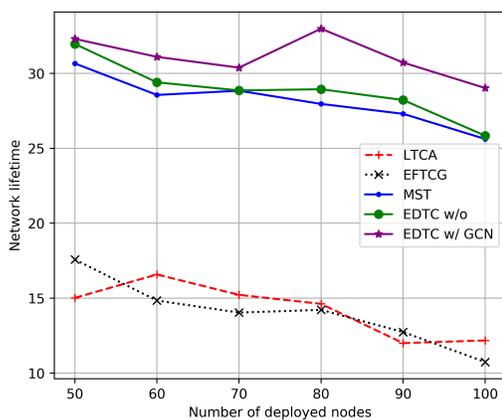
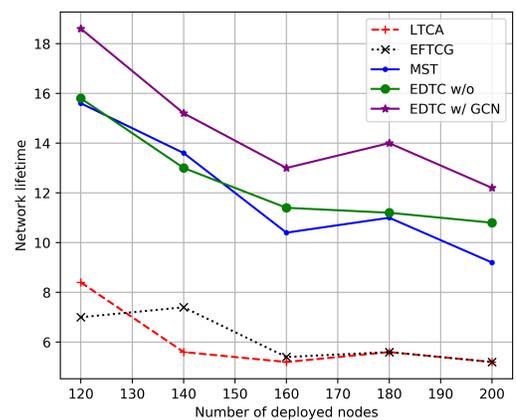
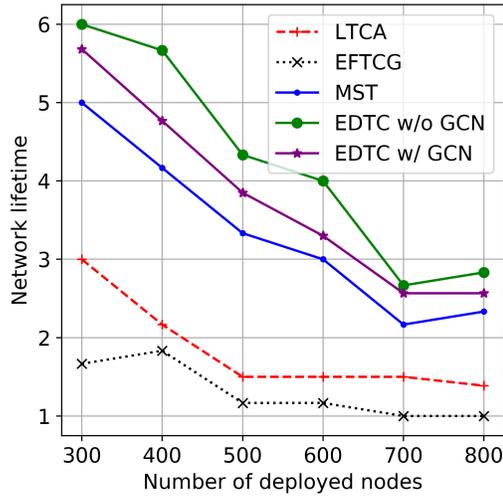
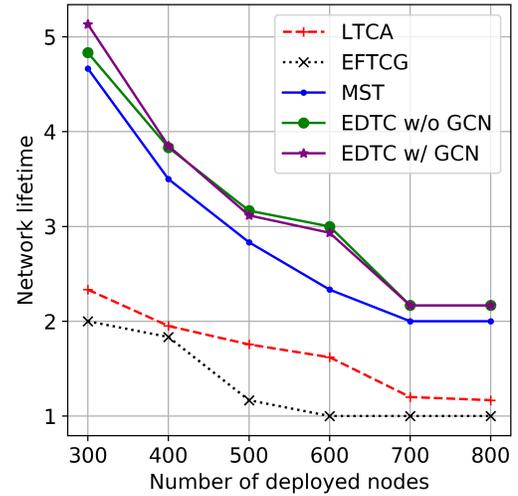
(a) $\tau = 70\%$.(b) $\tau = 70\%$.(c) $\tau = 80\%$.(d) $\tau = 80\%$.(e) $\tau = 90\%$.(f) $\tau = 90\%$.

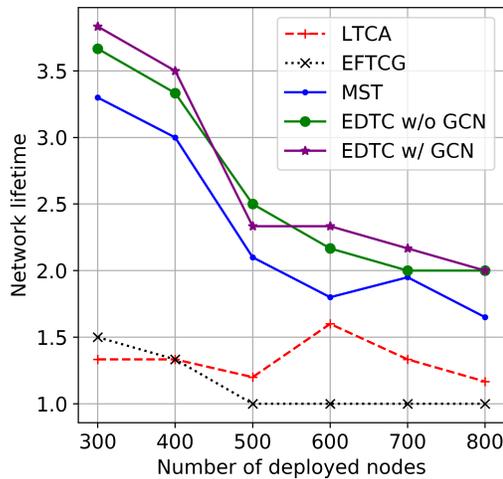
Figure 5.13: Network lifetime (the number of time-steps) in free communication experiment under different connectivity threshold settings.



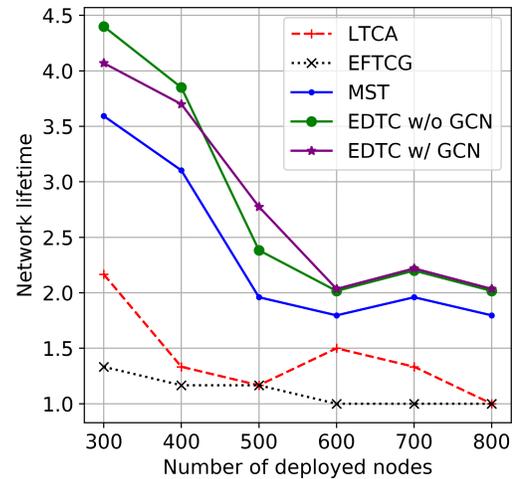
(a) $\tau = 60\%$;
 $n \in \{300, 400, 500, 600, 700, 800\}$.



(b) $\tau = 70\%$;
 $n \in \{300, 400, 500, 600, 700, 800\}$.



(c) $\tau = 80\%$;
 $n \in \{300, 400, 500, 600, 700, 800\}$.



(d) $\tau = 90\%$;
 $n \in \{300, 400, 500, 600, 700, 800\}$.

Figure 5.14: Network lifetime (the number of time-steps) in free communication experiment under different connectivity threshold settings.

5.6 Summary

In this work, we optimize the network lifetime by considering the balance between node residual energy and node degree. We assume that if a network topology allows the nodes with more residual energy to have a relatively higher degree, while the nodes

with less residual energy have a relatively lower degree, then the network lifetime will be prolonged. Based on this assumption, we introduce an energy distribution index (ED-index) algorithm as an evaluation metric, and further propose a novel topology control algorithm named EDTC. The EDTC (w/o GCN) leverages MST as the backbone of the network and re-introduces some edges to improve the ED-index. To reduce the topology construction time, we also design a GCN-based EDTC. We train the GCN to imitate the initial EDTC algorithm by predicting a link-level probability map, which is used as a guidance to re-introduce some edges to the MST. Simulation results show the prominent performance of the proposed EDTC algorithm (both versions) comparing with the state-of-the-art.

Since both versions of our EDTC algorithm are centralized algorithms, we require the existence of a central computer as the topology optimizer. When the amount of network nodes is not too large (depends on the central computer’s computing power), one can deploy either of the GCN-based EDTC algorithm or the initial EDTC algorithm on the central computer to optimize the network topology in real-time. However, if the amount of network nodes is large and the central computer’s computing power does not support real-time optimization via the initial EDTC algorithm, we suggest using the GCN-based EDTC algorithm. Note that, because the GCN-based EDTC algorithm requires training, we need to tune the simulation parameters in terms of different network characteristics and run massive simulations to train the GCN model before we implement the algorithm for real-time optimization.

We find that although the GCN-based method has satisfactory generalization ability (train on small instances and apply on large instances), when the size of an instance is very large (for instance, 1,000 nodes), the performance will be compromised. A straightforward approach for dealing with large instances is divide-and-conquer. By using this approach, we need to divide the original topology graph into several smaller sub-graphs, optimize the topology of each sub-graphs, and finally combine the local solutions into the global solution. In the future, we plan to leverage graph-based clustering methods to ensure the nodes in each sub-graph as close to each other as possible and explore the viability of using this approach in dealing with large instances.

Chapter 6

Conclusion

In this thesis, we explore how to apply machine learning in future-generation wireless network optimization problems. Just as the name implies, machine learning is the technique to enable the computer to learn from data. Therefore, how to derive the data contains hidden patterns, which can be learned by machine learning algorithms, is the first question we need to answer. For supervised learning, the data should be a collection of input and target pairs. Whereas, for reinforcement learning, the data is generated dynamically through the interaction of the learning agent with the environment (in our context, the environment is the network system). Because we have the centralized algorithms for both the dense RFID anti-collision problem and the energy-efficient topology control problem, we use the existing algorithms as “mentors” to “teach” the machine learning models. The process is to simulate a large set of situations, and leverage the existing algorithm to derive the solution, finally convert the situations and corresponding solutions to input and target pairs to enable supervised learning. However, for the mobile edge computing multi-level task-offloading problem, there is no existing algorithm. Also, the task-offloading is dynamic and sequential decision-making task. Instead of manually design an optimization algorithm, we can use the existing deep reinforcement learning algorithm to let the agents (in our case, agents are end devices and edge servers) to learn the optimized task-offloading policies through “playing”. The input (or observation) to an agent is the current network status, and the agent needs to predict a value (in our case, value is the cost) for each possible action (to offload or not) in terms of the observation.

To guide the agent to make reasonable predictions, we use our objective (minimizing latency and energy consumption) to design the cost function. As long as the agent learned to make reasonable predictions and picks the action with the lowest predicted cost each time, we deem the agent learned to make optimal task-offloading policy.

What types of machine learning models are appropriate to wireless network optimization problems is the second question. The answer is multi-fold because wireless network optimization problems are abstract and diverse. For instance, in our RFID anti-collision problem and topology control problem, we use graphs to represent the network systems, and leverage graph optimization approaches to solve the problems. We use the multi-layer feed-forward network for the RFID anti-collision problem because we only need one-hop node information, and thus we can compress the neighbor nodes' information into a fixed-dimension vector. However, in the topology control problem, we need the whole topology graph as the neural network input, it is infeasible to compress the arbitrary size graph into a fixed-dimension vector without losing much information. Moreover, a graph could have a considerable amount of matrix representations, which is also a challenge for training the machine learning model. In this situation, we choose graph convolutional network as the machine learning model. The advantage of graph convolution is that it can take an arbitrary size graph as input and not sensitive to different matrix representations of the same graph.

	DMWISBAII (w/ ML) ¹	DRL-MEC ²	EDTC (w/ GCN) ³
Type	Distributed algorithm (learning stage is centralized)	Distributed algorithm	Centralized algorithm
ANN	Multi-layer feed-forward network	Multi-layer feed-forward network	Graph convolutional network
Learning type	Supervised learning	Reinforcement learning	Supervised learning
Design philosophy	Imitate the centralized algorithm	Optimize through interaction and experience	Imitate the centralized algorithm
Objective	Maximize the total number of RFID tags can be read by the system at the same time	Reduce task execution latency; reduce end device energy consumption	Balance network connectivity and device residual energy
Superiority	Distribute workload; preserve performance	Dynamic optimization; support on-device learning	Reduce optimization time consumption

¹ The distributed MWISBAII algorithm with machine learning auxiliary.

² The deep reinforcement learning-based MEC network task offloading optimization algorithm.

³ The graph convolutional network-based WAIoT network optimization algorithm.

Table 6.1: Summary of the Proposed Machine Learning-Based Algorithms

Our proposed machine learning-based algorithms achieve satisfactory performance in our simulation experiments. Table 6.1 provides a summary of the proposed machine learning-based wireless IoT network optimization algorithms. We conclude this thesis

by putting forward several future research directions:

- **Introduce graph convolutional network and deep reinforcement learning to distributed RFID anti-collision algorithm.** We have demonstrated two specific network optimization tasks that could be done through the help of deep reinforcement learning or the graph convolutional network. In our future work, we can explore modeling each RFID reader as a reinforcement learning agent, and modeling our optimization problem as a sequence of interactions among RFID readers. The environment state could be the ego graph. A graph convolutional network is used to evaluate the value of actions that can be made at the current state. We can keep the agents interact with each other, learn to maximize the total number of tags that can be read by the system at the same time.
- **Utilize federated learning in learning optimal task-offloading policy.** The concept of federated learning is to get a global model through the distributed on-device learning and the centralized integration of distributed models [68]. Take the device-level task-offloading as an example. We can let each user device learns its task-offloading policy (in forms of a Q-network), and upload the trained Q-network model to the central server. The central server integrates the received Q-networks to a global Q-network, and let each user device to update its local Q-network to the global Q-network. We repeat this process to derive a high-quality centralized Q-network model. The goal is to leverage the computing power of user devices for training the models, but also get a global model with good generalization ability at last. The trained global model can be used to initialize the local models of new devices (new devices may join the network at any time).
- **Deal with large instances in the topology control problem through clustering.** Theoretically, the graph convolutional network can take an arbitrary size graph as the input, meaning that we can train the graph convolutional network on small instances and apply it on large instances. Here, the size of an instance indicates the number of graph nodes. Whereas, in practice, if the size of an instance is too large, the optimization performance could be compromised. A straightforward approach to handle the large instance is to split the graph into several smaller sub-graphs (through clustering methods), then optimize the topology of each sub-graph. Finally, connect the sub-topologies to get the final

topology. In the future, we will investigate the viability and efficiency of using graph-based clustering methods to derive the sub-graphs, where the nodes in a cluster should be as close as possible, and the corresponding sub-graph should be a connected graph.

Appendix A

List of Symbols and Notations

Some symbols or notations mentioned in Chapter 2 are also used in the later chapters.

Chapter 2

G : graph

V : the set of graph vertices

E : the set of graph edges

v_i : the i^{th} vertex in vertex set V

e_{v_i, v_j} : edge between vertex v_i and vertex v_j

$\text{deg}(v_i)$: the degree of v_i

$\text{neighbors}(G, v_i)$: the set of all the neighbor vertices of v_i in graph G

$w(v_i)$: weight of vertex v_i

$w(e_{v_i, v_j})$: weight of edge e_{v_i, v_j}

\mathbf{A} : binary adjacency matrix

\mathbb{R} : the set of real numbers

\mathbf{x} : the i^{th} input value of a single artificial neuron

\mathbf{a} : weight for the i^{th} input value

\mathbf{b} : bias of a single artificial neuron

$\text{activation}(\cdot)$: general representation of an element-wise activation function

o : output of a single artificial neuron

e : the Euler's number (approximately equals to 2.71828)

$\text{sigmoid}(x)$: sigmoid activation function

$\text{tanh}(x)$: hyperbolic tangent activation function

$\text{ReLU}(x)$: rectified linear unit activation function

$\max(x, y)$: the maximum value of x and y
 \vec{x} : input vector of an MLF network
 p : dimension of the input vector \vec{x}
 \vec{o} : output vector of an MLF network
 q : dimension of the output vector \vec{o}
 L : the number of hidden layers
 $\vec{h}^{(l-1)}$: the l^{th} hidden layer of MLF
 $nn(l-1)$: the number of neurons in the l^{th} hidden layer of MLF
 $\mathcal{W}^{(l-1)}$: weights of the l^{th} hidden layer of MLF
 $\mathcal{W}^{(L)}$: weights of the MLF output layer
 $\vec{b}^{(l-1)}$: biases of the l^{th} hidden layer of MLF
 $\vec{b}^{(L)}$: biases of the MLF output layer
 θ : the collection of weights and biases in an MLF
 $F_{\theta}(\vec{x}) = \vec{o}$: function representation of an MLF
 A : adjacency matrix (binary or weighted) of graph G
 D : vertex degree matrix
 \tilde{A} : adjacency matrix of graph G with added self-loops
 \tilde{D} : degree matrix of \tilde{A}
 $I_{|V|}$: the identity matrix of shape $|V| \times |V|$
 X : input signal of graph convolution
 W : graph convolution filters
 F : the number of graph convolution filters
 Z : output signal of graph convolution
 \tilde{S} : normalized graph Laplacian matrix (support matrix)
 $H^{(l)}$: the l^{th} graph convolution layer output signal
 $W^{(l)}$: filters of the l^{th} graph convolution layer
 $O(\)$: big-o notation of algorithm time-complexity
 $\mathcal{L}(\vec{o}, \vec{y})$: loss function of target \vec{y} and neural network output \vec{o}
 δ : gradient
 α : artificial neural network learning rate
 s : an MDP state
 s_0 : start state
 s_{end} : terminal state
 \mathcal{S} : the MDP state space
 a : an MDP action

\mathcal{A} : the MDP action space
 $\mathbb{P}(s'|s, a)$: the probabilistic transition function
 $R(s)$: reward at state s
 \mathcal{Q} : Q-table
 $argmax$: the arguments of the maxima
 max : the maxima
 $\mathcal{Q}(s, a)$: the Q-value of action a at state s
 ϵ : Q-learning exploration rate
 γ : Q-learning discount faction
 α^\diamond : Q-learning learning rate
 \mathcal{Q}_θ : deep Q network with the set of parameters θ
 \mathcal{L}_θ : deep Q network loss function

Chapter 3

d : radius of the interrogation range
 d' : radius of the interference range
 β : ratio of d'/d
 $limit$: the maximum number of RFID tags can be read by the RFID reader
 \mathbb{N} : the set of natural numbers
 N : the number of RFID readers in a dense RFID system
 M : the number of RFID tags in a dense RFID system
 \mathfrak{R} : the set of RFID readers in a dense RFID system
 \mathfrak{T} : the set of RFID tags in a dense RFID system
 r_i : the i^{th} RFID reader
 t_i : the i^{th} RFID tag
 T_{r_i} : the set of RFID tags within reader r_i 's interrogation range
 T'_{r_i} : the set of RFID tags within reader r_i 's interference range
 R : the result set of RFID readers that should be activated
 T : the result set of RFID tags that can be read by the RFID system
 \mathcal{I} : the maximum weight independent set
 \mathfrak{h} : the parameter used in [22] to control the truncation of partial solution
 $cost(G, v_i)$: cost function of GWMIN2 algorithm
 $STAT$: RFID reader status variable
 $BUFFER_{out}$: The buffer of signals waiting to be sent

BUFFER_{out} : The buffer of signals received signals waiting to be processed

CODE: the code of a signal

VALUE: the value of a signal

w_i : weight of the vertex that represents the i^{th} RFID reader

c_i : cost of the vertex that represents the i^{th} RFID reader

ϖ_i : average weight of the i^{th} RFID reader's neighbors

ι_i : average cost of the i^{th} RFID reader's neighbors

Λ : MWISBAII solution set

Θ_i : status of the i^{th} RFID reader

\mathfrak{s}_i : neural network predicted score of the i^{th} RFID reader

Chapter 4

\mathcal{U} : the set of user devices

u_i : the i^{th} user device

\mathcal{E} : the set of edge gateways

ε_i : the i^{th} edge gateway

G^* : gateway-level connectivity graph

φ : frequency of the wireless channel

ϑ : the frequency bandwidth

σ : additive white Gaussian noise

p_{rx} : received power

p_{tx} : transmit power

g_{rx} : receiver's antenna gain

g_{tx} : transmitter's antenna gain

$d_{tx,rx}$: the spatial distance between the transmitter and the receiver

π : the ratio of the circumference of a circle to its diameter

λ : wavelength

$r_{tx,rx}$: wireless transmission data rate

\mathcal{T} : task queue

\mathcal{T}_{u_i} : user device u_i 's task queue

$\mathcal{T}_{\varepsilon_i}$: edge gateway ε_i 's task queue

τ : task

\mathcal{T}^* : the set of completed tasks

$\wp(\tau)$: task size

$\omega(\tau)$: task workload
 \aleph : device-level task offloading decision
 Ψ : edge-level task offloading decision
 f : CPU frequency
 t_{exe_local} : local (on user device) execution time consumption of a task τ
 t_{wait_local} : local (on user device) waiting time consumption of a task τ
 t_{local} : overall local time consumption of a task τ
 ρ : average CPU working power consumption of user device
 e_{exe_local} : CPU energy consumption for executing a task τ
 \varkappa : weight factor
 c_{local} : local computing mode cost on task τ
 t_{trans} : transmission time consumption on a task τ
 e_{trans} : transmission energy consumption on a task τ
 t_{exe_edge} : edge (on edge server) execution time consumption on a task τ
 t_{wait_edge} : edge (on edge server) waiting time consumption on a task τ
 c_{device_level} : device-level task offloading mode cost on a task τ
 t'_{trans} : edge-level transmission time consumption on a task τ
 t'_{wait_edge} : edge (on the second edge server) waiting time consumption on a task τ
 c_{edge_level} : edge-level task offloading mode cost on a task τ
 π_{\aleph} : device-level task offloading policy
 π_{Ψ} : edge-level task offloading policy
 \mathcal{C}_{u_i} : average cost on user device u_i
 $\mathcal{C}_{\varepsilon_i}$: average cost on edge gateway ε_i
 ℓ : current total workload
 $\mathcal{D}_{\varepsilon_i}$: the set of connected user devices of edge gateway ε_i
 θ_{\aleph} : device-level task offloading deep Q network parameters
 θ_{Ψ} : edge-level task offloading deep Q network parameters
 \mathcal{M} : transition history memory
 ψ : size (number of transitions) of the transition history memory
 κ : deep Q-learning experience replay step size
 ζ : exploration rate decay factor
 $\mathcal{N}_{\varepsilon_i}$: the set of neighbor edge gateways of edge gateway ε_i
 ν : time slot length
 ξ : the change (probability) that a new task will come in each time slot

Chapter 5

V^* : the set of normal nodes

v_i^* : the i^{th} normal node

n : the number of normal nodes

V^+ : the set of sink nodes

v_i^+ : the i^{th} sink node

m : the number of sink nodes

V : the set of all network nodes

v_i : the i^{th} network node

\mathfrak{d} : the minimum distance between each pair of normal nodes

\mathfrak{r} : the maximum wireless communication range radius

\mathfrak{E}_{v_i} : residual energy of node v_i

\mathcal{N}_{v_i} : the set of neighbor nodes of v_i

\mathfrak{D}_{v_i} : degree of node v_i

φ : frequency of the wireless channel

ϑ : the frequency bandwidth

σ : additive white Gaussian noise

p_{rx} : received power

p_{tx} : transmit power

g_{rx} : receiver's antenna gain

g_{tx} : transmitter's antenna gain

$d_{v_{tx}, v_{rx}}$: the spatial distance between the transmitter node and the receiver node

π : the ratio of the circumference of a circle to its diameter

λ : wavelength

$r_{tx,rx}$: wireless transmission data rate

\mathfrak{d} : the size of data

$t(\mathfrak{d})$: time consumption of transferring data of size \mathfrak{d}

$\mathcal{J}(\mathfrak{d})$: energy consumption of transferring data of size \mathfrak{d}

G' : the optimized network topology graph

$z(x, x_{min}, x_{max})$: min-max normalization of given value x in terms of the minimum and maximum values (x_{min} and x_{max}) among all the samples

η : scale coefficient

$\tilde{\mathfrak{E}}$: normalized node residual energy

$\tilde{\mathfrak{D}}$: normalized node degree

- ϕ : the maximum number of edges to be re-introduced
- E^\dagger : the set of re-introduced links (edges)
- G^\dagger : $G^\dagger = (V, E^\dagger)$
- C : dimension of vertex feature vector
- F : the number of graph convolution filters for each graph convolution layer
- L : the number of graph convolution layers
- $H^{(l)}$: the output of the l^{th} graph convolution layer
- $W^{(l)}$: graph convolution filters of the l^{th} graph convolution layer
- \tilde{M} : re-constructed output of the last graph convolution layer
- \hat{M} : probability map
- \mathbb{Z}^+ : the set of positive integers
- $\mathcal{F}_\theta(A, \hat{A}, \hat{A}^{(MST)}, X) = \hat{M}$: the function representation of our graph convolution network model
- \circ : element-wise matrix multiplication operation
- \top : matrix transpose operation
- \hat{A}^\dagger : the binary adjacency matrix of G^\dagger
- \mathcal{L} : loss function
- \mathcal{G} : the set of randomly simulated graphs
- H_0 : null-hypothesis
- H_A : alternative hypothesis
- r' : expected lowest data transmission rate
- \top : network connectivity threshold

Appendix B

List of Abbreviations

3G: The third generation telecommunication

4G: The fourth generation telecommunication

5G: The fifth generation telecommunication

A

AE: Absolute error

AR: Augmented reality

AI: Artificial intelligence

ANN: Artificial neural network

AUC: Area under the curve

B

B5G: Beyond 5G

BP: Backpropagation

C

CDMA: Code division multiple access
CSMA: Carrier-sense multiple access
CCN: Content-centric network
CNN: Convolutional neural network
CPU: Central processing unit

D

D2D: Device-to-device
DQN: Deep Q-network
DQL: Deep Q-learning
DT: Delaunay triangulation

E

ED-index: Energy distribution index
EDTC: Energy-degree topology control

F

FDMA: Frequency-division multiple access

G

GCN: Graph convolutional network
GG: Gabriel graph
GPS: Global Position System

I

ID: Identity

IoE: Internet of Everything

IoT: Internet of Things

ICN: Information-centric network

IP: Internet protocol

L

LTCA: Localized topology control algorithm

M

MEC: Mobile edge computing

MIMO: Multiple input, multiple out

mmWave: Millimeter wave

ML: Machine learning

MLF: Multi-layer feed-forward

MDP: Markov decision process

MWIS: Maximum weight independent set

MSE: Mean squared error

MST: Maximum/minimum spanning tree

O

OFDM: Orthogonal frequency-division multiplexing

Q

QoS: Quality of service

R

RCCA: Reader-coverage collision avoidance

ReLU: Rectified linear units

RFID: Radio-frequency identification

ROC: Receiver operating characteristic

S

SE: Squared error

T

TDMA: Time-division multiple access

U

UAV: Unmanned aerial vehicle

V

VR: Virtual reality

W

WAIoT: Wireless ad-hoc IoT

WANET: Wireless ad-hoc network

WSN: Wireless sensor network

Bibliography

- [1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.
- [2] A. A. Abudaqa, T. M. Al-Kharoubi, M. F. Mudawar, and A. Kobilica. Simulation of arm and x86 microprocessors using in-order and out-of-order cpu models with gem5 simulator. In *2018 5th International Conference on Electrical and Electronic Engineering (ICEEE)*, pages 317–322. IEEE, 2018.
- [3] G. A. Akpakwu, B. J. Silva, G. P. Hancke, and A. M. Abu-Mahfouz. A survey on 5g networks for the internet of things: Communication technologies and challenges. *IEEE Access*, 6:3619–3647, 2017.
- [4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.
- [5] F. Al-Turjman, C. Altrjman, S. Din, and A. Paul. Energy monitoring in iot-based ad hoc networks: An overview. *Computers & Electrical Engineering*, 76:133–142, 2019.
- [6] K. Ashton et al. That ‘internet of things’ thing. *RFID journal*, 22(7):97–114, 2009.
- [7] M. J. Brady, T. Cofino, H. K. Heinrich, G. W. Johnson, P. A. Moskowitz, and G. F. Walker. Radio frequency identification tag, Oct. 28 1997. US Patent 5,682,143.
- [8] M. J. Brady, D.-W. Duan, and V. S. Kodukula. Radio frequency identification system, Aug. 8 2000. US Patent 6,100,804.

- [9] F. Campioni, S. Choudhury, and F. Al-Turjman. Readers scheduling for rfid networks in the iot era. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2018.
- [10] F. Campioni, S. Choudhury, and F. Al-Turjman. Scheduling rfid networks in the iot and smart health era. *Journal of Ambient Intelligence and Humanized Computing*, Jan 2019.
- [11] H. Chen, L. Liu, R. Che, K. Lin, X. Ai, and Y. Li. On using sampling bloom filter for unknown tag identification in large-scale rfid systems. *IEEE Access*, 6:57095–57104, 2018.
- [12] H.-s. CHEN, C. Bo, and C.-h. WU. A survey of topology control in wireless networks. *DEStech Transactions on Computer Science and Engineering*, (cna), 2018.
- [13] X. Chen, L. Jiao, W. Li, and X. Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2015.
- [14] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis. Performance optimization in mobile-edge computing via deep reinforcement learning. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–6. IEEE, 2018.
- [15] M. Cheney. *Tesla: man out of time*. Simon and Schuster, 2011.
- [16] D. Christin, A. Reinhardt, P. S. Mogre, R. Steinmetz, et al. Wireless sensor networks and the internet of things: selected challenges. *Proceedings of the 8th GI/ITG KuVS Fachgespräch Drahtlose sensornetze*, pages 31–34, 2009.
- [17] E. G. Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Computing Surveys (CSUR)*, 3(2):67–78, 1971.
- [18] L. Da Xu, W. He, and S. Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4):2233–2243, 2014.
- [19] Y. Dai, D. Xu, S. Maharjan, Z. Chen, Q. He, and Y. Zhang. Blockchain and deep reinforcement learning empowered intelligent 5g beyond. *IEEE Network*, 33(3):10–17, 2019.

- [20] B. Delaunay et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.
- [21] R. Diestel. Graph theory, volume 173 of. *Graduate texts in mathematics*, page 7, 2012.
- [22] P. Du and Y. Zhang. A new distributed approximation algorithm for the maximum weight independent set problem. *Mathematical Problems in Engineering*, 2016, 2016.
- [23] W. Du, T. Lei, Q. He, W. Liu, Q. Lei, H. Zhao, and W. Wang. Service capacity enhanced task offloading and resource allocation in multi-server edge computing environment. *arXiv preprint arXiv:1903.04709*, 2019.
- [24] Y. Du, J. Gong, Z. Wang, and N. Xu. A distributed energy-balanced topology control algorithm based on a noncooperative game for wireless sensor networks. *Sensors*, 18(12):4454, 2018.
- [25] Y. Du, J. Xia, J. Gong, and X. Hu. An energy-efficient and fault-tolerant topology control game algorithm for wireless sensor network. *Electronics*, 8(9):1009, 2019.
- [26] D. W. Engels and S. E. Sarma. The reader collision problem. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, volume 3, pages 6–pp. IEEE, 2002.
- [27] P. Erdos. On the graph theorem of turán. *Mat. Lapok*, 21(249-251):10, 1970.
- [28] E. Ezhilarasan and M. Dinakaran. A review on mobile technologies: 3g, 4g and 5g. In *2017 second international conference on recent trends and challenges in computational models (ICRTCCM)*, pages 369–373. IEEE, 2017.
- [29] A. Fahim and T. ElBatt. Multi-reader rfid tag identification using bit tracking (mrti-bt). In *2016 IEEE International Conference on RFID (RFID)*, pages 1–7. IEEE, 2016.
- [30] X. Feng, J. Zhang, J. Chen, G. Wang, L. Zhang, and R. Li. Design of intelligent bus positioning based on internet of things for smart campus. *IEEE Access*, 6:60005–60015, 2018.

- [31] R. M. Ferdous, A. W. Reza, and M. F. Siddiqui. Renewable energy harvesting for wireless sensors using passive rfid tag technology: A review. *Renewable and Sustainable Energy Reviews*, 58:1114–1128, 2016.
- [32] N. Ferns, P. Panangaden, and D. Precup. Metrics for markov decision processes with infinite state spaces. *arXiv preprint arXiv:1207.1386*, 2012.
- [33] H. T. Friis. A note on a simple transmission formula. *Proceedings of the IRE*, 34(5):254–256, 1946.
- [34] Y. Fu, X. Wang, E. Wang, and Z. Qian. A bit arbitration tree anti-collision protocol in radio frequency identification systems. *International Journal of Distributed Sensor Networks*, 13(11):1550147717741571, 2017.
- [35] K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic zoology*, 18(3):259–278, 1969.
- [36] M. R. Garey. Computers and intractability: A guide to the theory of np-completeness, freeman. *Fundamental*, 1997.
- [37] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 15554–15566, 2019.
- [38] J. R. Griggs. Lower bounds on the independence number in terms of the degrees. *Journal of Combinatorial Theory, Series B*, 34(1):22–39, 1983.
- [39] I. G. I. T. R. W. Group et al. Ieee 5g and beyond technology roadmap. *White Paper*, 2017.
- [40] J. Gui and K. Zhou. Flexible adjustments between energy and capacity for topology control in heterogeneous wireless multi-hop networks. *Journal of Network and Systems Management*, 24(4):789–812, 2016.
- [41] A. Guillen-Perez, R. Sanchez-Iborra, M.-D. Cano, J. C. Sanchez-Aarnoutse, and J. Garcia-Haro. Wifi networks on drones. In *2016 ITU Kaleidoscope: ICTs for a Sustainable World (ITU WT)*, pages 1–8. IEEE, 2016.
- [42] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim. Energy efficient task caching and offloading for mobile edge computing. *IEEE Access*, 6:11365–11373, 2018.

- [43] W. He, G. Yan, and L. Da Xu. Developing vehicular data cloud services in the iot environment. *IEEE Transactions on Industrial Informatics*, 10(2):1587–1595, 2014.
- [44] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd annual Hawaii international conference on system sciences*, pages 10–pp. IEEE, 2000.
- [45] J. Ho, D. W. Engels, and S. E. Sarma. Hiq: a hierarchical q-learning algorithm to solve the reader collision problem. In *International Symposium on Applications and the Internet Workshops (SAINTW'06)*, pages 4–pp. IEEE, 2006.
- [46] Z. Hong, R. Wang, and X. Li. A clustering-tree topology control based on the energy forecast for heterogeneous wireless sensor networks. *IEEE/CAA Journal of Automatica Sinica*, 3(1):68–77, 2016.
- [47] K. Hornik, M. Stinchcombe, H. White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- [48] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.
- [49] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.
- [50] J. Huang, Q. Duan, C.-C. Xing, and H. Wang. Topology control for building a large-scale and energy-efficient internet of things. *IEEE Wireless Communications*, 24(1):67–73, 2017.
- [51] L. Huang, X. Feng, L. Zhang, L. Qian, and Y. Wu. Multi-server multi-user multi-task computation offloading for mobile edge computing networks. *Sensors*, 19(6):1446, 2019.
- [52] Y. Huang, N. Shinohara, and H. Toromura. A wideband rectenna for 2.4 ghz-band rf energy harvesting. In *2016 IEEE Wireless Power Transfer Conference (WPTC)*, pages 1–3. IEEE, 2016.
- [53] K. Islam and S. G. Akl. Localized topology control algorithm with no geometric information for ad hoc sensor networks. In *2008 Second International*

- Conference on Sensor Technologies and Applications (sensorcomm 2008)*, pages 65–72. IEEE, 2008.
- [54] A. K. Jain, J. Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.
- [55] W. S. Johnson Jr and H. B. Reisgies. Radio frequency identification (rfid) payment terminal with display-embedded rfid antenna, Sept. 6 2011. US Patent 8,011,594.
- [56] C. K. Joshi, T. Laurent, and X. Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.
- [57] Z. jun Tang, Y. Guo, and Q. Liu. Research on an improved fusion rfid collision avoidance algorithm. *Journal of Communications Technology, Electronics and Computer Science*, 22:6–19, 2019.
- [58] W.-S. Jung, H. Ahn, and Y.-B. Ko. Designing content-centric multi-hop networking over wi-fi direct on smartphones. In *2014 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 2934–2939. IEEE, 2014.
- [59] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.
- [60] Y. Kang, M. Kim, and H. Lee. A hierarchical structure based reader anti-collision protocol for dense rfid reader networks. In *13th International Conference on Advanced Communication Technology (ICACT2011)*, pages 164–167. IEEE, 2011.
- [61] H. J. Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.
- [62] J. A. Khan, H. K. Qureshi, and A. Iqbal. Energy management in wireless sensor networks: A survey. *Computers & Electrical Engineering*, 41:159–176, 2015.
- [63] D.-Y. Kim, B.-J. Jang, H.-G. Yoon, J.-S. Park, and J.-G. Yook. Effects of reader interference on the rfid interrogation range. In *2007 European Microwave Conference*, pages 728–731. IEEE, 2007.

- [64] D.-Y. Kim, H.-G. Yoon, B.-J. Jang, and J.-G. Yook. Interference analysis of uhf rfid systems. *Progress In Electromagnetics Research*, 4:115–126, 2008.
- [65] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [66] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.
- [67] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *Proceedings of ICLR (2017)*, 2017.
- [68] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.
- [69] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.
- [70] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.
- [71] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [72] I. Lee and K. Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431–440, 2015.
- [73] B. Li, Z. Fei, and Y. Zhang. Uav communications for 5g and beyond: Recent advances and future trends. *IEEE Internet of Things Journal*, 6(2):2241–2263, 2018.
- [74] N. Li, J. C. Hou, and L. Sha. Design and analysis of an mst-based topology control algorithm. *IEEE Transactions on Wireless Communications*, 4(3):1195–1206, 2005.
- [75] N. Li, J.-F. Martinez-Ortega, and V. H. Diaz. Efficient and reliable topology control based opportunistic routing algorithm for wsns. *arXiv preprint arXiv:1709.10317*, 2017.

- [76] X.-Y. Li, K. Moaveni-Nejad, W.-Z. Song, and W.-Z. Wang. Interference-aware topology control for wireless sensor networks. In *2005 Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2005. IEEE SECON 2005.*, pages 263–274. IEEE, 2005.
- [77] Y. Li, X. Shi, A. Lindgren, Z. Hu, P. Zhang, D. Jin, and Y. Zhou. Context-aware data dissemination for icn-based vehicular ad hoc networks. *Information*, 9(11):263, 2018.
- [78] Z. Li, Q. Chen, and V. Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pages 539–548, 2018.
- [79] B.-H. Liu, N.-T. Nguyen, V.-T. Pham, and Y.-H. Yeh. A maximum-weight-independent-set-based algorithm for reader-coverage collision avoidance arrangement in rfid networks. *IEEE Sensors Journal*, 16(5):1342–1350, 2016.
- [80] C.-F. Liu, M. Bennis, and H. V. Poor. Latency and reliability-aware task offloading and resource allocation for mobile edge computing. In *2017 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7. IEEE, 2017.
- [81] X. Liu, Z. Qin, and Y. Gao. Resource allocation for edge computing in iot networks via reinforcement learning. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.
- [82] Y. Liu, H. Yu, S. Xie, and Y. Zhang. Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. *IEEE Transactions on Vehicular Technology*, 2019.
- [83] D. J. MacKay and D. J. Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.
- [84] Y. Mao, J. Zhang, and K. B. Letaief. Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems. In *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2017.
- [85] A. A. Mbacke, N. Mitton, and H. Rivano. A survey of rfid readers anticollision protocols. *IEEE Journal of Radio Frequency Identification*, 2(1):38–48, 2018.

- [86] A. Meddeb and A. Jaballah. Algorithm for readers arrangement without collision in rfid networks. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2017 18th International Conference on*, pages 316–321. IEEE, 2017.
- [87] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking. A review on internet of things (iot), internet of everything (ioe) and internet of nano things (iont). In *2015 Internet Technologies and Applications (ITA)*, pages 219–224. IEEE, 2015.
- [88] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [89] A. Moubayed, A. Shami, P. Heidari, A. Larabi, and R. Brunner. Edge-enabled v2x service placement for intelligent transportation systems. *IEEE Transactions on Mobile Computing*, 2020.
- [90] A. Munir, M. T. R. Laskar, M. S. Hossen, and S. Choudhury. A localized fault tolerant load balancing algorithm for rfid systems. *Journal of Ambient Intelligence and Humanized Computing*, Oct 2018.
- [91] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
- [92] U. Olgun, C.-C. Chen, and J. L. Volakis. Efficient ambient wifi energy harvesting technology and its applications. In *Proceedings of the 2012 IEEE International Symposium on Antennas and Propagation*, pages 1–2. IEEE, 2012.
- [93] K. Pehkonen, S. Uskela, K. Kalliojarvi, L. Oksanen, and K. Rikkinen. Key technologies and concepts for beyond-3g networks. In *Wireless and Mobile Communications*, volume 4586, pages 43–57. International Society for Optics and Photonics, 2001.
- [94] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.
- [95] T. Qiu, N. Chen, K. Li, D. Qiao, and Z. Fu. Heterogeneous ad hoc networks: Architectures, advances and challenges. *Ad Hoc Networks*, 55:143–152, 2017.

- [96] S. Ranadheera, S. Maghsudi, and E. Hossain. Mobile edge computation offloading using game theory and reinforcement learning. *arXiv preprint arXiv:1711.09012*, 2017.
- [97] D. G. Reina, S. L. Toral, F. Barrero, N. Bessis, and E. Asimakopoulou. The role of ad hoc networks in the internet of things: A case scenario for smart environments. In *Internet of things and inter-cooperative computational technologies for collective intelligence*, pages 89–113. Springer, 2013.
- [98] H. Rezaie and M. Golsorkhtabaramiri. A fair reader collision avoidance protocol for rfid dense reader environments. *Wireless Networks*, 24(6):1953–1964, 2018.
- [99] C. M. Roberts. Radio frequency identification (rfid). *Computers & security*, 25(1):18–26, 2006.
- [100] S. Rosati, K. Kruźelecki, G. Heitz, D. Floreano, and B. Rimoldi. Dynamic routing for flying ad hoc networks. *IEEE Transactions on Vehicular Technology*, 65(3):1690–1700, 2015.
- [101] F. Rosenblatt. *The perceptron: A theory of statistical separability in cognitive systems*. United States Department of Commerce, 1958.
- [102] F. Rosenblatt. Perceptron simulation experiments. *Proceedings of the IRE*, 48(3):301–309, 1960.
- [103] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [104] W. Saad, M. Bennis, and M. Chen. A vision of 6g wireless systems: Applications, trends, technologies, and open research problems. *IEEE network*, 2019.
- [105] H. Saadi, R. Touhami, and M. C. Yagoub. Tdma-sdma-based rfid algorithm for fast detection and efficient collision avoidance. *International Journal of Communication Systems*, 31(1):e3392, 2018.
- [106] S. Sakai, M. Togasaki, and K. Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, 126(2-3):313–322, 2003.

- [107] B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.
- [108] M. A. Sharkh, A. Shami, and M. Kalil. The era of the personal cloud: What does it mean for cloud providers? In *Edge Computing*, pages 1–15. Springer, 2019.
- [109] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.
- [110] D.-H. Shih, P.-L. Sun, D. C. Yen, and S.-M. Huang. Taxonomy and survey of rfid anti-collision protocols. *Computer communications*, 29(11):2150–2166, 2006.
- [111] D. Soldani, Y. J. Guo, B. Barani, P. Mogensen, I. Chih-Lin, and S. K. Das. 5g for ultra-reliable low-latency communications. *Ieee Network*, 32(2):6–7, 2018.
- [112] I. Stephen. Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, 50(2):179, 1990.
- [113] H. Su, Y. Li, Z. Siyi, H. Yan, and H. Chen. Multichannel reader collision avoidance mechanism in rfid-sensor integrated networks. *Journal of Computers*, 29(5):260–271, 2018.
- [114] J. Su, Z. Sheng, V. C. Leung, and Y. Chen. Energy efficient tag identification algorithms for rfid: survey, motivation and new design. *IEEE Wireless Communications*, 2019.
- [115] J. Su, Z. Sheng, L. Xie, G. Li, and A. X. Liu. Fast splitting-based tag identification algorithm for anti-collision in uhf rfid system. *IEEE Transactions on Communications*, 67(3):2527–2538, 2018.
- [116] X. Sun and N. Ansari. Edgeiot: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, 54(12):22–29, 2016.
- [117] Y. Sun, M. Peng, Y. Zhou, Y. Huang, and S. Mao. Application of machine learning in wireless networks: Key techniques and open issues. *IEEE Communications Surveys & Tutorials*, 21(4):3072–3108, 2019.

- [118] H.-T. Tai, W.-C. Chung, C.-J. Wu, R.-I. Chang, and J.-M. Ho. Sop: Smart offloading proxy service for wireless content uploading over crowd events. In *2015 17th International Conference on Advanced Communication Technology (ICACT)*, pages 659–662. IEEE, 2015.
- [119] R. E. Tarjan and A. E. Trojanowski. Finding a maximum independent set. *SIAM Journal on Computing*, 6(3):537–546, 1977.
- [120] M. N. Tehrani, M. Uysal, and H. Yanikomeroglu. Device-to-device communication in 5g cellular networks: challenges, solutions, and future directions. *IEEE Communications Magazine*, 52(5):86–92, 2014.
- [121] S. H. G. ten Hagen et al. *Continuous state space Q-learning for control of nonlinear systems*. PhD thesis, Universiteit van Amsterdam [Host], 2001.
- [122] S. J. Udoka. Automated data capture techniques: a prerequisite for effective integrated manufacturing systems. *Computers & industrial engineering*, 21(1-4):217–221, 1991.
- [123] M. van Otterlo and M. Wiering. Reinforcement learning and markov decision processes. In *Reinforcement Learning*, pages 3–42. Springer, 2012.
- [124] S. Varshney and R. Kuma. Variants of leach routing protocol in wsn: A comparative analysis. In *2018 8th International conference on cloud computing, data science & engineering (confluence)*, pages 199–204. IEEE, 2018.
- [125] J. Von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993.
- [126] J. Wang, Y. Guo, L. Guo, B. Zhang, and B. Wu. Performance test of mpmd matching algorithm for geomagnetic and rfid combined underground positioning. *IEEE Access*, 2019.
- [127] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas. Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning. *IEEE Communications Magazine*, 57(5):64–69, 2019.
- [128] J. Wang, W. Wei, W. Wang, and R. Li. Rfid hybrid positioning method of phased array antenna based on neural network. *IEEE Access*, 6:74953–74960, 2018.

- [129] Y. Wang. Topology control for wireless sensor networks. In *Wireless sensor networks and applications*, pages 113–147. Springer, 2008.
- [130] Z. Wang, N. Ye, R. Malekian, F. Xiao, and R. Wang. Trackt: Accurate tracking of rfid tags with mm-level accuracy using first-order taylor series approximation. *Ad hoc networks*, 53:132–144, 2016.
- [131] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
- [132] P. Werbos. Beyond regression:” new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.
- [133] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- [134] M. G. Wing, A. Eklund, and L. D. Kellogg. Consumer-grade global positioning system (gps) accuracy and reliability. *Journal of forestry*, 103(4):169–173, 2005.
- [135] A. Woo and D. E. Culler. A transmission control scheme for media access in sensor networks. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 221–235. ACM, 2001.
- [136] P. Yan, F. Al-Turjman, I. Al-Oqily, and S. Choudhury. An energy-efficient topology control algorithm for optimizing the lifetime of wireless ad-hoc iot networks in 5g and b5g. (submitted) Elsevier, 2020.
- [137] P. Yan and S. Choudhury. Optimizing mobile edge computing multi-level task offloading via deep reinforcement learning. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, (accepted) 2020.
- [138] P. Yan, S. Choudhury, and R. Wei. A distributed graph-based dense rfid readers arrangement algorithm. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.
- [139] P. Yan, S. Choudhury, and R. Wei. A machine learning auxiliary approach for the distributed dense rfid readers arrangement algorithm. *IEEE Access*, 2020.

- [140] L. Yang, J. Han, Y. Qi, C. Wang, T. Gu, and Y. Liu. Season: Shelving interference and joint identification in large-scale rfid systems. In *2011 Proceedings IEEE INFOCOM*, pages 3092–3100. IEEE, 2011.
- [141] X. Yang, Z. Chen, K. Li, Y. Sun, N. Liu, W. Xie, and Y. Zhao. Communication-constrained mobile edge computing systems for wireless virtual reality: Scheduling and tradeoff. *IEEE Access*, 6:16665–16677, 2018.
- [142] Y. Yao, Q. Cao, and A. V. Vasilakos. Edal: An energy-efficient, delay-aware, and lifetime-balancing data collection protocol for heterogeneous wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 23(3):810–823, 2015.
- [143] J. Yu and W. Lee. Gentle: Reducing reader collision in mobile rfid networks. In *2008 The 4th International Conference on Mobile Ad-hoc and Sensor Networks*, pages 280–287. IEEE, 2008.
- [144] G. Zhang, S. Tao, Q. Cai, W. Gao, J. Jia, J. Wen, et al. A fast and universal rfid tag anti-collision algorithm for the internet of things. *IEEE Access*, 7:92365–92377, 2019.
- [145] J. Zhang, G. Y. Tian, A. M. Marindra, A. I. Sunny, and A. B. Zhao. A review of passive rfid tag antenna-based sensors and systems for structural health monitoring applications. *Sensors*, 17(2):265, 2017.
- [146] S. Zhang, H. Tong, J. Xu, and R. Maciejewski. Graph convolutional networks: Algorithms, applications and open challenges. In *International Conference on Computational Social Networks*, pages 79–91. Springer, 2018.
- [147] T. Zhang, X. Xu, L. Zhou, X. Jiang, and J. Loo. Cache space efficient caching scheme for content-centric mobile ad hoc networks. *IEEE Systems Journal*, 13(1):530–541, 2018.
- [148] B. Zhi, W. Sainan, and H. Yigang. A novel anti-collision algorithm in rfid for internet of things. *IEEE Access*, 6:45860–45874, 2018.
- [149] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

- [150] Z. Zhou, J. Feng, Z. Chang, and X. Shen. Energy-efficient edge computing service provisioning for vehicular networks: A consensus admm approach. *IEEE Transactions on Vehicular Technology*, 68(5):5087–5099, 2019.
- [151] Z. Zhou, H. Gupta, S. R. Das, and X. Zhu. Slotted scheduled tag access in multi-reader rfid systems. In *2007 IEEE International Conference on Network Protocols*, pages 61–70. IEEE, 2007.
- [152] W. Zhu, Y. Hong, V. Raychoudhury, R. Zhao, and D. Wang. Adaptive distributed reader activation approach for large-scale rfid systems. In *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 82–90. IEEE, 2015.