HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Minimizing node failure in Wireless Sensor Rechargeable Sensor Network by utilizing GraphSAGE and Deep Q-Learning

Vuong Dinh An
an.vd180003@sis.hust.edu.vn

Major: Information Technology

Supervisor: Assoc. Prof. Huynh Thi Thanh Binh _____

PhD. Bui Quoc Trung Signature

Department: Computer Science

School: School of Information and Communications Technology

HANOI, 08/2022

ACKNOWLEDGMENT

This thesis would not have been possible without the assistance of many people. I gratefully thank my supervisor, Assoc. Prof Huynh Thi Thanh Binh, who revised many versions before this final material. Thank you for providing me with helpful advice throughout the work. I sincerely appreciate guidance and support from my committee lecturer, PhD. Bui Quoc Trung.

Thanks to the researchers and students from MSOLab, who constantly give me comments and suggestions to complete my thesis. I cannot thank MSc. Tran Thi Huong enough, who directly helped me establish this work. You have constantly been giving conscientious recommendations and orientations to my topic for the past years. Also, I want to thank my research team for providing technical assistance, especially Nguyen Ngoc Bao, for helping me when I was conducting experiments. Thanks to the lecturers, students, and other Hanoi University of Science and Technology members for your encouragement, contribution, and endeavor over the last few years. It is an honor for me to be a part of this university.

Lastly, thanks to my family and friends who always offer me support and love.

ABSTRACT

Wireless Sensor Networks (WSNs) consist of sensors deployed over a Region of Interest (RoI) to supervise the environment. Over the past decades, WSNs have been widely applied to numerous domains from infrastructure to military. However, sensors' energy limitation remains the bottleneck phenomenon to maintaining the network operation. Fortunately, in recent years, the development of Wireless Energy Transfer (WET) technology has created a new generation of network, so-called the Wireless Rechargeable Sensor Network (WRSN). Unlike WSNs, WRSNs comprise of one or more Mobile Charger(s) (MC) travelling around the network to charge sensors. Since the sensor lifetime is decided by the MC's charging strategy, designing an effective one has become a primary challenge in WRSNs.

Although many efforts have been made to establish charging schedules, existing works face several limitations. Most works assume that the MC's battery capacity is infinite and adopt the point-to-point charging method, where the MC only charges one sensor at a time. These studies are only suitable for small-sized, sparse networks. Moreover, the previous works usually fix a charging threshold for the sensors, which leads to charging congestion. The ultimate goal of the charging task is to avoid the sensors' energy depletion; nevertheless, former research address indirectly this objective.

Motivated by the above shortcomings, this thesis investigates the multi-point charging scheme optimization under the MC's limited-energy constraint to minimize the number of exhausted sensors (node failure). To this end, the concerned problem was formulated as a Markov Decision Process (MDP) representing the interactions between the MC (agent) and the network (environment) through choosing the optimal charging location (action). The thesis proposes an algorithm hybrid of Graph Neural Network (GNN) and Deep Reinforcement Learning (DRL) to determine an optimal charging policy for the MC. Additionally, the proposed algorithm automatically determines which sensors to charge based on the sensors' residual energy and average consumption rate. This work conducts extensive simulations and experiments to evaluate the effectiveness of the proposed algorithms. Experimental results demonstrate that the thesis proposal significantly reduces the node failure compared to other benchmarks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

LIST OF ABBREVIATIONS

Abbreviation	Definition
AI	Artificial Intelligence
BS	Base station
CL	Charging location
DL	Deep Learning
DQL	Deep Q-Learning
DRL	Deep Reinforcement Learning
GNN	Graph Neural Network
GSADQL	The hybrid of GraphSAGE and Deep Q-Learning algorithm (proposal)
MC	Mobile charger
MDP	Markov decision process
ML	Machine Learning
RL	Reinforcement Learning
RNN	Recurrent neural network
RoI	Region of Interest
WET	Wireless energy transfer
WRSN	Wireless rechargeable sensor network
WSN	Wireless sensor network

CHAPTER 1. INTRODUCTION

This chapter briefly introduces the Wireless Sensor Networks, Wireless Rechargeable Sensor Networks, their applications, and some challenges in network deployment. The problem statement is presented in this chapter, as well as some related works.

1.1 Wireless Sensor Network

Nowadays, with the rising of the Internet of things (IoT), Wireless sensor networks (WSNs) are becoming more and more significant. There are numerous real-world applications, such as environment monitoring, agriculture, healthcare system, military, innovative infrastructure, industrial control systems, and so on [1]. Wireless sensor networks are known as autonomous systems, which are sets of sensors that track the surrounding environment and transmit the collected data to an intermediate station. This intermediate station deputizes the network to serve the gathering data to end-users through the Internet connection.

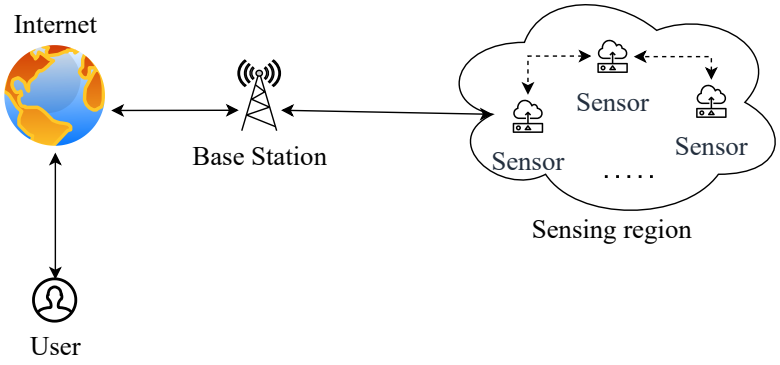

Figure 1.1: An example of wireless sensor network

Each sensor participating in the network has three tasks: sensing, communicating, and computing [1]. Hence, this task-based view segregates the sensor node architecture into three different units:

- The sensing unit captures data from the territory and produces an analog signal.
- The computing unit consists of a memory board and a micro-controller, which converts the analog signal to digital signal and processes data.
- The communication unit exchanges the information through the WSN.

Additionally, each sensor has a battery to maintain its activity. According to [2], a sensor can capture a wide range of information, such as temperature, moisture,

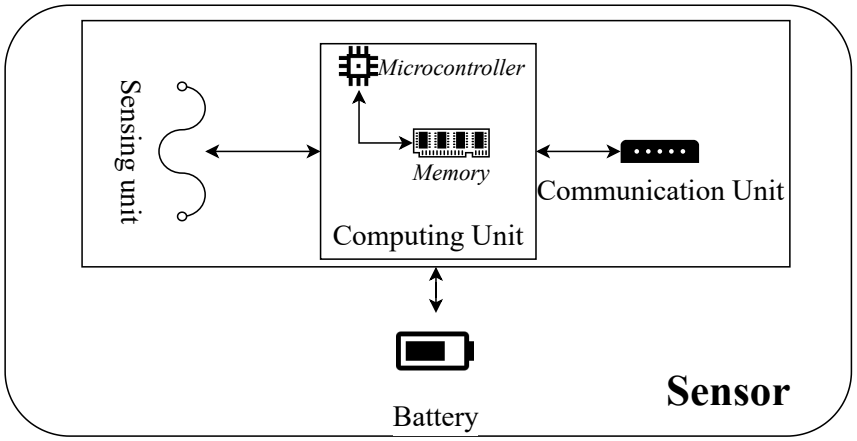

Figure 1.2: Sensor node architecture

velocity, pH, loudness, brightness, mechanical stress quantification, and so on. Thanks to the ability to collect a variety of data, WSNs plays a significant role in our real life.

- In health applications, WSNs can be utilized to track and diagnoses a group of people inside a hospital [3]. These applications provide doctors with the capability of telemonitoring patients and drug management.
- In environmental applications, WSNs are deployed to track the alteration of specific objects: birds, insects, plants, forests. Some of the numerous applications can be listed here: forest fire alarm [4], flood detection [5], earthquake detection [6], and so on.
- In home applications, sensors are embedded in light bulbs, vacuum cleaners, air conditions, and air purifiers to operate automatically and remotely [7]. Because of this, people can now manage their home devices comfortably.
- In military applications, WSN can be used as the backbone of a C4ISRT military system [1]. These systems are served for essential missions, such as reconnaissance of opposing infrastructures, monitoring friendly forces, and battlefield surveillance.
- In commercial applications, WSNs are installed to office control, inventory control [8], car theft alarm [9], etc.

In recent times, WSNs have brought innovations and inspirations to human civilization. The deployments of WSNs benefit significantly from low-cost batteries. However, these batteries also have low capacities and are usually non-replenished, leading to energy shortages. When a sensor runs out of energy, the network's performance can be significantly reduced. Consequently, energy conservation has become one of the most critical problems in WSNs.

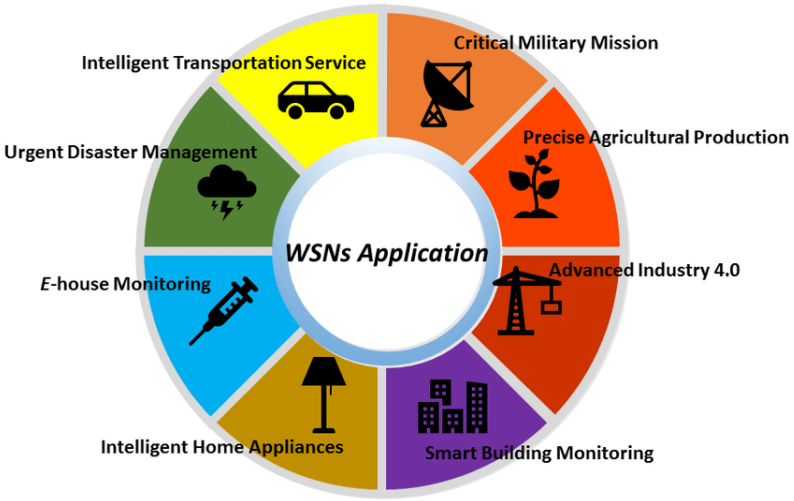

Figure 1.3: Applications of WSNs [10]

According to [11], there are three main methods to maintaining the energy of WSN:

- *Duty-cycling*: controls the sleep/wake activities of the sensors. Methods based on this idea usually rely on hardware costs, and there will be a minority of sensors that can wake the others in reality [12].
- *Data-driven*: focuses on optimizing the process of collecting data by sensors. For instance, in [13], the authors proposed an algorithm that predicts the succeeded data package based on previous samples. In general, data-driven approaches usually demands high-performance sensors, which is not a feasible solution in all cases [11].
- *Mobility-based*: changes the locations of sensors in WSN to achieve near-optimal energy-efficient data sensing scheme [11]. However, the main issue these approaches usually have to tackle is the difficulties of implementing mobilizers in real-life applications.

To sum up, these approaches are not widely applicable. Another method is to harness energy from renewable sources as proposed in [14]. Nevertheless, these mechanisms are still ambitious because such sources of energy (sun, wind, thermal, kinetic, etc.) are uncontrollable.

1.2 Wireless Rechargeable Sensor Network

Recently, the advances in Wireless Energy Transfer (WET) have opened up a new generation of network, so-called the Wireless Rechargeable Sensor Network (WRSN) [15]. WRSNs have emerged as a promising solution to the limited-energy issue in traditional WSNs.

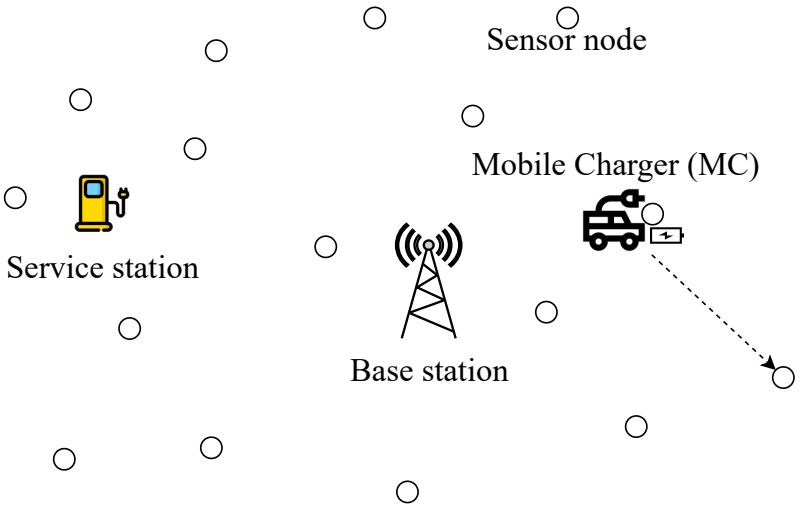

Figure 1.4: An example of a WRSN

Fig 1.4 illustrates an example of the Wireless Rechargeable Sensor Network (WRSN), in which the MC have finished charging a sensor and moving to the next charging location. WRSNs consist of a set of sensor nodes supplied with a wireless energy receiver through magnetic resonance coupling technology. One or more Mobile Chargers (MC), which are equipped with a wireless charger, depart from a service station and move across the charging locations to charge sensors in the network [16]. Additionally, a charging scheme comprises two main components: 1) charging path, which is the sequence of Charging Locations (CLs), and 2) charging time, which is the sequence of the amount of time spent at each respective CL. The sensors' lifetime depends greatly on the MC's charging strategy. Therefore, determining an effective charging schedule is greatly concerned by the research in WRSNs.

1.3 Charging scheme optimization problem

Considers a WRSN deployed over a two-dimensional Region of Interests (RoI). There are four main components in this problem, they are: 1) the set of sensors collecting data from the surrounding environment, 2) the Base Station (BS) receiving packets that contain sensing data from the sensors, 3) the MC maintaining the sensors' energy, 4) the set of CLs indicating the positions the MC stop to charge the sensors. Periodically, the BS can calculate the average energy consumption of the sensors based on their packet generation rate [17]. To find which positions will the MC stop to charge the sensors, this thesis inherits the idea of [18] to determine CLs that are favorable for prolonging critical sensors' lifetime. The charging scheme of the MC can be described as follows: starts from the BS, visits some CLs to charge sensors and returns to the BS. **The ultimate goal of this problem is to determine the optimal charging scheme that minimizes the number of energy-depleted**

sensors (node failure). For an example, Fig 1.5 shows a charging scheme. In this

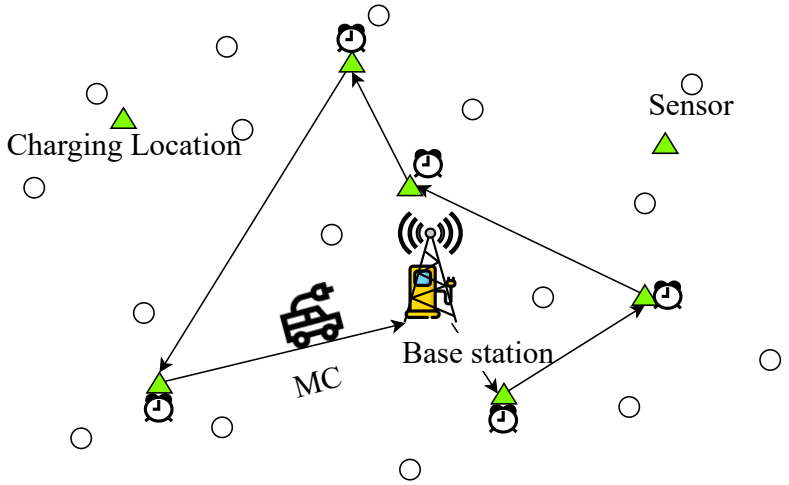

Figure 1.5: In WRSNs, MC starts from a service station, then visits some charging locations. Whenever MC stops at a charging location, it transmits energy to nearby sensor(s)

illustration, the service station coincides with the BS, which is also the assumption throughout this thesis.

1.4 Related works

There have been known several approaches to tackle the concerned problem. Based on the charging model of the MC, these methods can be classified as [19]:

- Multi-point charging: the MC can charge multiple sensors simultaneously. In this charging model, the sensors' lifetime depends not only on the charging scheme but also on the position of CLs as well-positioned CLs can significantly prolong the duration of the sensors. In [20], the authors proposed a grid-based algorithm for searching CLs and the MC's charging path.
- Point-to-point charging: the MC can only charge at most one sensor at a time. For instance, The authors in [19] presented a heuristic method to optimize the charging scheme with the assumption that the MC needs to get to the exact position of a sensor to charge it.

Based on the determinism of the scheduler, these approaches can be segregated into [21]:

- Offline (Deterministic) charging: the scheduler determines the charging scheme before the MC's charging process is executed. An instance of this mechanism was proposed in [22]. The authors designed a hybrid clustering algorithm that determines the MC's charging.
- Online (Non-deterministic) charging: the scheduler determines the charging scheme concurrent with the MC's charging process. The online charging mech-

anism is preferred in practical applications [21] since it usually considers a hypothesis of constantly changing networks. The node failure problem was introduced in [23], in which the authors then proposed an algorithm that selects the next node based on charging probability. Also, they contributed an algorithm that finds the minimum number of requests from energy-depleted nodes.

Based on the strategy of the charging schedule algorithm, there are two classes of strategies: on-demand charging and periodic charging [19].

- Periodic charging: periodically, the BS (scheduler) takes the network information as the input and then outputs the charging scheme for the next charging cycle. In [24], the authors came up with a novel energy transfer concept based on distance and angle. This new idea is exploited to minimize the total charging delay time. An optimal solution is provided thanks to linear programming. In [25], the authors minimized the number of MCs under no energy-depleted sensor constraint. To acquire this goal, they jointly determined MCs' charging paths and located positions of BSs. In [26], the authors combined Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) to determine a path that minimizes the idle time that the MC stays at the BS to regain energy. In [27], the authors proposed approximation algorithms to determine the charging path with the goal of maximizing the accumulative charging utility gain or minimizing the travelling energy of MC. Nonetheless, periodic scheduling is not widely applicable since node usually deactivates unexpectedly, which results in the change of network topology. The predetermined charging schedule will no longer be optimal in the succeed topology [28].

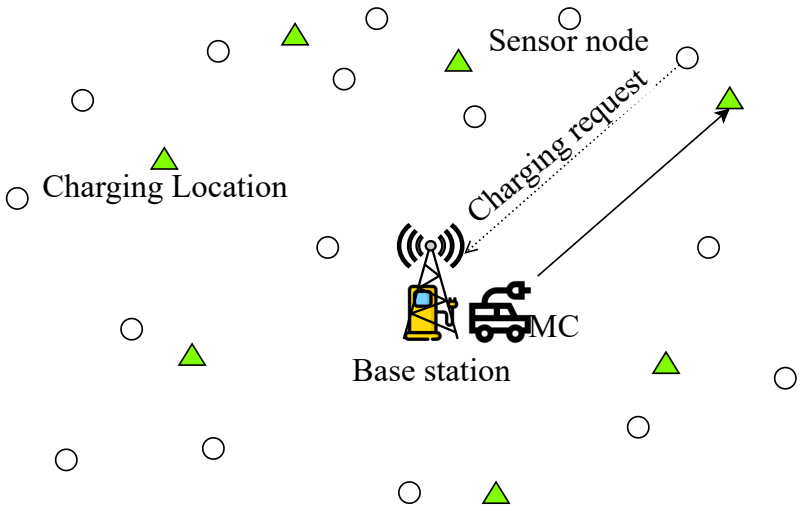

Figure 1.6: An illustration of on-demand charging

Table 1.1: A comparison between related works and this thesis

[Ref.]	Charging model		Factor optimization		MC's battery capacity		Strategy		Determinism	
	Point-to -point	Multi -point	Charging path	Charging time	Unlimited (Sufficient)	Limited	On -demand	Periodic	Offline	Online
[26]	✓		✓		✓			✓	✓	
[23] [29]–[31]	✓		✓		✓		✓			✓
[24]		✓		✓	✓			✓	✓	
[20][26]		✓	✓		✓			✓	✓	
[25]	✓		✓			✓		✓	✓	
Proposal		✓	✓			✓		✓		✓

- On-demand charging: the scheduler takes the sensors' charging requests as input and outputs the charging scheme. Specifically, a sensor with below-threshold energy sends a charging request to BS, and then MCs are arranged to charge that sensor (see Fig 1.6). In [29], the authors came up with an algorithm that dynamically determines alert thresholds for the sensor's energy. In [30], the authors utilized gravitational search to optimize the queue of charging requests. In [31], the authors model the charging problem as an MDP. They established an advanced deep reinforcement learning algorithm to solve the addressed MDP to obtain an optimized charging path under the constraints of MC's energy, sensors' energy, and time windows. Table 1.1 sums up the recent literature in WRSNs.

Although researchers have made substantial progress, these works have some common limitations. 1) Most works concentrated on small-sized networks with an insignificant number of sensors. They preferred the point-to-point charging model in these works; however, it only works well in minor-scale networks and proves inefficient in larger-size networks. 2) Most existing works made impractical assumptions, such as the MC has an unlimited battery capacity. 3) For on-demand methods, efficiency of the charging algorithm vastly depends on selecting the energy threshold for the charging request. If low-threshold is determined, MCs will have a short time to fulfill charging requests from critical nodes. Furthermore, if many requests are coming at a time, picking a path to satisfy these requests may become infeasible [23], and the network is in great peril. In contrast, a high threshold may lead to an ineffective charging scheme since more requests come at a time may occur [23].

Recent years have witnessed the evolution of Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) which has achieved outstanding performance in numerous tasks [32], [33]. An emerging subfield of AI/ML, Reinforcement Learning (RL), also plays an important role in this evolution [34]. RL aims to create interactive agents that maximize cumulative reward regarding

a decision-making problem. The combination of RL and DL sets up Deep Reinforcement Learning (DRL), which is a promising approach to Artificial General Intelligence problems [35]. It was presented in [36] that a deployed AI can have the same performance as an average human in Atari 2600, which created a resounding success aware by the community of AI/ML at that time.

Inspired by these phenomenons, this thesis aims to solve the addressed problem by leveraging DRL. Some motivations behind this innovational approach are: 1) The charging scheme optimization problem can be treated as a decision-making problem, in which the MC is the agent, and the network is the environment. The MC aims to determine the optimal charging scheme, similar to finding the optimal policy for the agent in this decision-making problem. 2) Since fixing the charging request threshold may cause the problem of charging congestion, DRL provides an effective solution to overcome this limitation. The agent trained by DRL can autonomously exploit the network's structure to determine subsequent action. 3) A generalizable agent is robust to the network changes; therefore, it can reasonably make future decisions based on the past experiences it collected.

1.5 Contributions and organization of the thesis

Overall, the main contributions of the thesis are summarized as follows:

- Studies the concept of WRSNs, charging schedule optimization problem, a wide spectrum of charging schemes: point-to-point, multi-point, offline, and online.
- Proposes an MDP model, which mathematically formulates the multi-point charging schedule optimization problem.
- Proposes a hybrid algorithm of Graph Neural Network (GNN) and Deep Reinforcement Learning (DRL). GNN provides an efficient mechanism to encode the state of the network, while the optimal charging location is determined by DRL.
- Finally, extensive experiments are conducted in various scenarios. The practical outcomes suggest that the proposed algorithm outperforms other state-of-the-art approaches.

The structure of this thesis is as follows:

- Chapter 1 introduces some basic concepts of WSNs, WRSNs, charging scheme optimization problem and its related works.
- Chapter 2 introduces some fundamental theories about Reinforcement Learning, Deep Reinforcement Learning, and Graph Neural Network.

- Chapter 3 establishes the network model and provides a mathematical formulation (MDP) for the charging scheme optimization problem.
- Chapter 4 presents the proposed algorithm.
- Chapter 5 conducts extensive experiments and analysis.

This chapter provides the basic theoretical foundation for the thesis. First, Reinforcement Learning is introduced along with Q-Learning and Deep Q-Learning. The last part of this chapter presents the background for Graph Neural Networks and especially focuses on GraphSAGE.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a sub-field of Machine Learning (ML), alongside supervised and unsupervised learning. RL considers the interactions between an agent and an environment. The concept of RL is to seek action for the agent so that the reward received by the environment is maximum [35]. Overall, ML can be classified as in Fig 2.1.

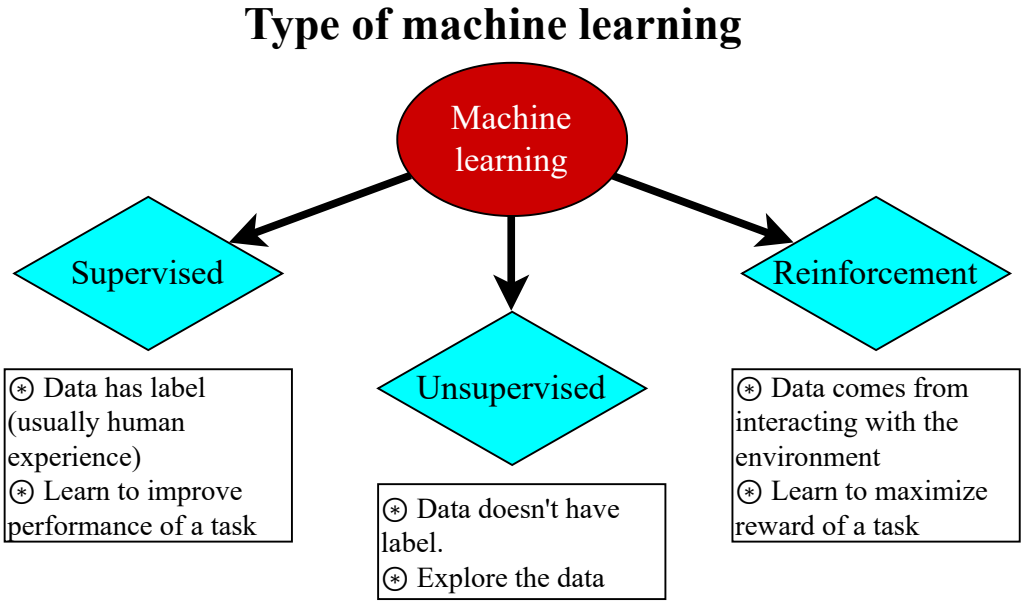

Figure 2.1: Types of ML

2.1.1 Markov Decision Process

Traditionally, RL problem is modelled as a Markov Decision Process (MDP) [37], which concerns the agent-environment relation at each *timestep* t. In general, MDP is a triplet $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}_0)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions the agent can take. $\mathcal{P}_0$ is known as *transition probability kernel* [34]. The concept of $\mathcal{P}_0$ is that given a state s and agent chose action a, then $\mathcal{P}_0(\cdot|x, a)$ is a probability map from $\mathcal{S} \times \mathbb{R} \to \mathbb{R}$. For example:

$$\mathcal{P}_0(y, R) = \Pr(y, R|S_t = x, A_t = a) \tag{2.1}$$

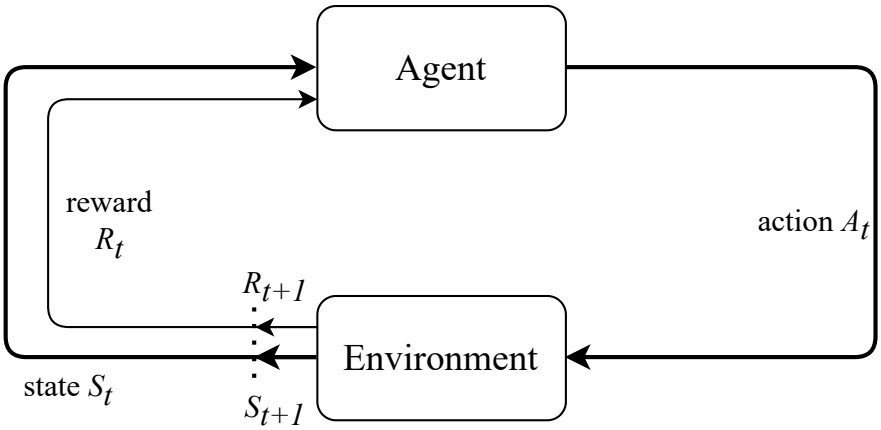

Figure 2.2: A general Markov Decision Process consists of an agent interacting with an environment. At each time t, the agent perceives the *state* S_t of the environment and takes an *action* A_t. The environment moves from state S_t to S_{t+1}, rises a reward of R_{t+1} to the agent. This interaction process continues in loop: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots$

shows the probability of transitioning from state $S_t = x$ by action $A_t = a$ to state $S_{t+1} = y$ and yield an *intermediate reward* $R_{t+1} = R$. The *return* is defined as follows:

$$\mathcal{R} = \sum_{t=0}^{+\infty} \gamma^t R_{t+1} \tag{2.2}$$

where $\gamma \in [0, 1]$ is *discount factor*. If $\gamma = 0$ then $\mathcal{R} = R_0$, the agent is near-sighted and only considers short-term reward. On the other hand, when γ grows, the agent is more far-sighted since it considers long-term rewards. Since R_t follows from a probability map, it is stochastic. The *expected return* is considered as:

$$\mathbb{E}[\mathcal{R}] = \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t R_{t+1}\right] \tag{2.3}$$

The objective of MDP is to maximize expected return, i.e. (2.3). In order to solve an MDP, the most common methods are to estimate the expected return and find the optimal behavior for the agent. For each state x, the *optimal value* at this state is denoted as $V^*(x) = \mathbb{E}[\mathcal{R}|S_0 = x]$. $V^* : \mathcal{S} \to \mathbb{R}$ is so-called the *optimal value function*. An *optimal* behaviour must achieve the optimal value for all state $x \in \mathcal{S}$, i.e. the optimal value function [34]. The mentioned behavior is well-known as *policy*. There are two types of policy:

- Deterministic stationary policies are policies that maps directly from a state x to an action a, for example, suppose $\pi : \mathcal{S} \to \mathcal{A}$ is a deterministic stationary policy, then, the agent always selects action $A_t = \pi(x) = a$ regarding to $S_t = x$.
- Stochastic stationary policies, on the other hand, are probability distributions

over $\mathcal{A}$. If π is a stochastic stationary policy and given $S_t = x$, the selected action A_t will follow as:

$$A_t \sim \pi(\cdot|S_t = x) \tag{2.4}$$

Underlying π, the *value function* can be defined as follows:

$$V^{\pi}(x) = \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t R_{t+1} | S_0 = x\right] \tag{2.5}$$

So far, the value function only shows the expected return of a state x. This estimation sometimes is not enough information for the agent to select an action a. Analogously, the *action-value function* can be defined as follows:

$$Q^{\pi}(x, a) = \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t R_{t+1} | S_0 = x, A_0 = a\right] \tag{2.6}$$

Let Q^* be the optimal action-value function. According to [35], the following relation holds:

$$V^*(x) = \sup_{a \in \mathcal{A}} Q^*(x, a), \forall x \in \mathcal{S} \tag{2.7}$$

The relation (2.7) infers that knowing Q^* is sufficient for always getting the maximum expected return for any state x, since the agent only needs to select the action a yield the maximum value of $Q^*(x, a)$. Q-Learning is an algorithm based on optimizing Q^{π} and produces a near-optimal policy $\widetilde{\pi}$.

2.1.2 Q-Learning

At each timestep t, Q-Learning estimate $Q^*(x, a)$ by quantity $Q_t(x, a)$, the updates rules based on Bellman equation [37]:

$$Q^*(x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{S}} \Pr(y|S_0 = x, A_0 = a) \sup_{a' \in \mathcal{A}} Q^*(y, a'), \forall (x, a) \in \mathcal{S} \times \mathcal{A} \tag{2.8}$$

where $r(x, a) = \mathbb{E}[R_{t+1}|S_t = x, A_t = a]$ is the expected intermediate reward regard to (x, a). Suppose the agent observes a transition tuple $(S_t, A_t, R_{t+1}, S_{t+1})$. The update rules at each timestep t are [35]:

$$\delta_{t+1} = R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q_t(S_{t+1}, a') - Q_t(S_t, A_t) \tag{2.9}$$

$$Q_{t+1}(x,a) = \begin{cases} Q_t(x,a) + \alpha_t \delta_{t+1}, & \text{if } S_t = x, A_t = a \\ Q_t(x,a), & \text{otherwise} \end{cases} \tag{2.10}$$

Unfortunately, (2.9) and (2.10) only works for finite MDPs, which applies when $\mathcal{S}$ and $\mathcal{A}$ are both finite. In case the MDP is not finite, the following estimation is introduced: $Q_\theta = \theta^\intercal \phi$, where $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ is known as *feature extraction* [34] and θ is a d-dimension vector parameter. Feature extraction is a method to estimate an infinite set (in this case: $\mathcal{S} \times \mathcal{A}$ is infinite). Each feature ϕ_i of ϕ is called *basis function*. There are many basis functions in practice, such as wavelet, Fourier and polynomial, etc. [38]. The update rule (2.9) stays the same; however, the second rule needs to modify as [34]:

$$\theta_{t+1} = \theta_t + \alpha_t \delta_{t+1} \nabla_\theta Q_{\theta_t}(S_t, A_t) \tag{2.11}$$

2.2 Deep Q-Learning

Deep Q-Learning (DQL) was introduced in [36], their work used deep neural networks techniques, which is one of the most prominent areas in recent years. Thanks to deep neural networks ([33], [39]), agents in DQL can extract much more abstract levels of data, this result in a more robust learning capability for deep Q-network agent. In DQL, a neural network θ is used to approximate the action-value function Q. In the original paper, the algorithm was designed as a model-free algorithm.

2.2.1 Key components

The authors initially proposed three key components:

- Replay memory D: at each timestep t, $D = \{E_1, E_2, \ldots, E_t\}$ stores previous interactions between agent and environment. $E_t = \langle S_t, A_t, S_{t+1}, R_t \rangle$ is the transition in timestep t. Normally, the replay memory has a limited size of $\mathcal{M}_\mathcal{D}$. This component is used for experience replaying [40]. In detail, this technique samples a minibatch $\langle S, A, S', R \rangle \sim U(D)$, where U is a uniform distribution.
- Target network $\hat{Q}$: This component generates the target action-value Q for each training minibatch.
- Policy network Q: is used for selecting action at each timestep t. The agent follows by a ϵ-greedy policy [35], i.e., chooses the action that maximizes the approximate action-value function with probability $1-\epsilon$ and chooses a random action with probability ϵ.

2.2.2 Update rules in Deep Q-Learning

The neural network structure of the target network is the same as the policy network. The policy network is updated by backpropagation [32] with the sampled minibatch at each timestep t. In detail, the gradient descent process is to minimize the following loss [36]:

$$L_t(\theta_t) = \mathbb{E}_{\langle S,A,S',R\rangle \sim U(D)} \left[(R + \gamma \max_{A'} \hat{Q}_t(S', A') - Q(S, A, \theta_t)) \right] \tag{2.12}$$

where $\hat{Q}_t, Q$ are a current approximation for the target network and policy network, respectively. On the other hand, the target network is periodically updated after $\mathcal{M}_C$ timestep by cloning exactly the policy network, i.e.:

$$\hat{Q}_j = Q_{k\mathcal{M}_C}, \forall j = \overline{k\mathcal{M}_C, (k+1)\mathcal{M}_C - 1}, \forall k \in \mathbb{N} \tag{2.13}$$

2.3 Graph neural network

Graph neural network (GNN) is a neural network that can learn data structures in a graph [41]. After learning, a GNN can be used to predict various data structures on the graph, such as graph-level, edge-level, and node-level [42].

Consider a graph as $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of edges. For convention, denote $E(v)$ as the set of neighboring nodes of v. Additionally, each node is sometimes abstraction of a real-life object. For instance, in a social network graph, each node represents a person's profile and captures personal information like age, place of birth, gender, etc.; each piece of information is a feature. Suppose v_i is represented by a X-dimensional vector $(v_i^1, v_i^2, \ldots, v_i^X)$. In GNN, the ultimate end is to find a neural network $\mathcal{G} : \mathbb{R}^X \to \mathbb{R}^D$ that maps each node to its latent characterization. $\mathcal{G}$ is also known as node embedding function [43].

2.3.1 Learning and inference in GNN

An ideal GNN can retain inter-dependencies between nodes in the original graph. Furthermore, if u, v be two nodes in G, then, the similarity of $\mathcal{G}(u)$ and $\mathcal{G}(v)$ should approximate the similarity of u and v. There are three important criteria based on these intuitions:

- Neighborhood dependencies: a GNN algorithm must gather all its neighboring node embeddings (messages) [44].

- Aggregation: an aggregate function (must be an order-invariant function) is used, for example, average, sum, ..., to aggregate neighbors of a node.
- Computation: a forward pass rule for the GNN must be defined. This rule will determine what information will be propagated from a layer to its successive layer. Generally, the formula for this propagation is:

$$l^k(v) = \sigma\left(W_k \mathcal{F}\{l^{k-1}(u) | u \in E(v)\} + B_k l^{k-1}(v)\right) \tag{2.14}$$

where σ is activation function in layer k; l^k is the latent representation in k-th layer; $\mathcal{F}$ is mentioned aggregate function; W_k, B_k are weight and bias in this layer, respectively.

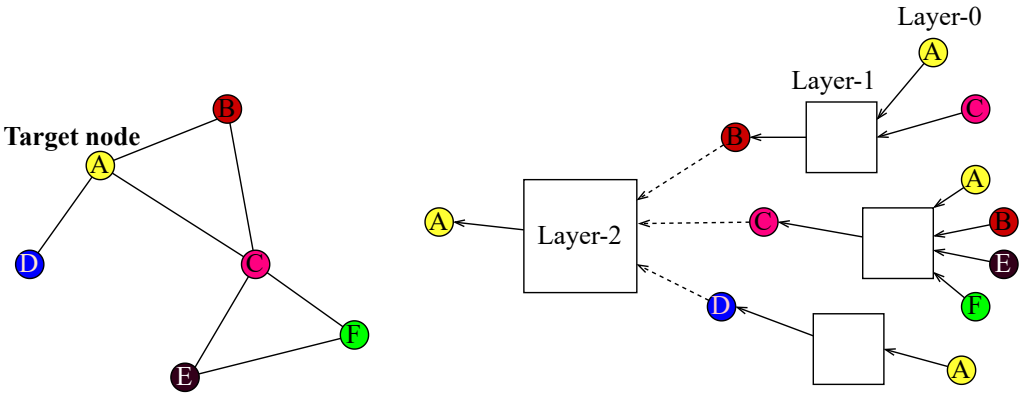

Figure 2.3: An example of a GNN

The type of interference in a GNN heavily depends on the type of learning. Learning can be either unsupervised or supervised. In unsupervised, a GNN tries to learn about the hidden structure of the initial graph, like the similarity of nodes or random walk. In supervised learning, a GNN tries to predict unseen data in the future. Fig 2.3 gives information about an example of a GNN.

2.3.2 GraphSAGE

The majority drawback of most GNN approaches is that they usually assume every node in the graph must be present during learning, which makes a large-scale graph tough to learn. GraphSAGE was proposed to sort this issue out [45]. The solution to this problem is to learn based on induction. GraphSAGE is supervised learning; it utilizes graph information to produce low-dimensional node embeddings and efficiently predicts future data.

Compared to existing works, the novel idea behind GraphSAGE is that it learns parameters of aggregate functions in GNN. GraphSAGE applies some modifications in neighborhood dependencies: a subset of neighboring vertices is used in-

stead of the full neighboring set. In computation, a substituted operation for (2.14) is as follows:

$$l^k(v) = \sigma\left(\left[W_k\mathcal{F}\{l^{k-1}(u)|u \in E(v)\}, B_k l^{k-1}(v)\right]\right) \tag{2.15}$$

where the square bracket in the above equation is denoted for concatenated function. In order to learn parameters of GraphSAGE $\mathcal{G}$, a loss function is introduced:

$$\mathcal{L}(\mathcal{G}(v)) = -\log(\sigma\left(\langle\mathcal{G}(v), \mathcal{G}(u)\rangle\right)) - \mathcal{Q}\mathbb{E}_{u_n\sim P_n(u)}\log(\sigma\left(-\langle\mathcal{G}(v), \mathcal{G}(u_n)\rangle\right)) \tag{2.16}$$

In (2.16), u is a nearby node of v picked by fix-size random walk, the author use sigmoid [46] for σ, P_n is negative sampling distribution and $\mathcal{Q}$ is number of samples . This loss function is used to orient proximity nodes to have analogous embeddings. Also, (2.16) is utilized to distance nodes that are totally different. The authors considered three aggregate architectures:

- Mean: this function is almost the same as proposed in GCN [47]. In the formula, this function can be described as:

$$\mathcal{F} = \sum_{u\in E(v)} \frac{l^{k-1}(u)}{|E(v)|} \tag{2.17}$$

- LSTM [48]: since LSTM is not satisfied the symmetric property of aggregate function, the author resolved this problem by applying LSTM to a random shuffle of an order set:

$$\mathcal{F} = \text{LSTM}\left(\left[l^{k-1}(u), u \in \pi(E(v))\right]\right) \tag{2.18}$$

- Pool: this function combines a learnable W_{pool} and max-pooling [49] to aggregate information of neighboring set:

$$\mathcal{F} = \max\left(\{W_{\text{pool}}l^{k-1}(u) + b, u \in (E(v)\}\right) \tag{2.19}$$

CHAPTER 3. NETWORK MODEL

The network model is presented first in this chapter, followed by its mathematical formulation. Consider a Wireless Rechargeable Sensor Network (WRSN) $\mathcal{N}$ deployed a Region of Interest (RoI). As presented in Chapter 1, there are four main components in this problem:

- The set of sensors: $\mathbf{SN} = \{SN_1, SN_2, \ldots, SN_n\}$. For each sensor SN_i, some fundamental information is known, such as location (x_{SN_i}, y_{SN_i}) and its consumption rate p_i.
- The set of Charging Locations (CLs): $\mathbf{CL} = \{CL_1, CL_2, \ldots, CL_m\}$. This thesis extends the idea in [18] to find the precise position (x_{CL_j}, y_{CL_j}) for each CL_j.
- Base station (BS): the center of the transmitting process. All sensing data will be transferred to this termination to serve end-users. The location of BS is $(x_{\text{BS}}, y_{\text{BS}})$. Conventionally, denote CL_0 as the BS.
- Mobile charger (MC): can charge nearby sensors by staying at a particular CL. In the proposed network model, suppose the service station, i.e., the station that helps recharge the MC is overlapped with the BS.

Let the length of the period be $\mathcal{T}$. This problem aims to output a charging scheme for MC to charge sensors in $\mathcal{N}$ with the time constraint bounded by $\mathcal{T}$. This charging scheme is a sequence of *charging actions*. For simplicity, the i-th charging action is defined by (CL_{π_i}, τ_i), where $\pi_i \in \{0, 1, 2, \ldots, m\}$ indicates the correspond CL the MC stays; $\tau_i \in [0, \mathcal{T}]$ indicates how long the MC spends on charging at CL_{π_i}.

3.1 Energy model

The energy consumption p_i is based on the research presented in [17]. Two main tasks are held for each sensor: sensing the surrounding environment and conveying collected information to the BS. Let ϵ_{elec} be the consumption rate for each bit by the physical circuits. Let $\epsilon_{\text{mp}}, \epsilon_{\text{fs}}$ be the amplify energy multi-path communication and free space, respectively. The needed energy to convey b bits is calculated as follows:

$$e_r = b\epsilon_{\text{elec}} \tag{3.1}$$

$$e_t = \begin{cases} b\left(\epsilon_{\text{elec}} + \epsilon_{\text{fs}} d^2\right), & \text{if } d < d_0 \\ b\left(\epsilon_{\text{elec}} + \epsilon_{\text{mp}} d^4\right), & \text{otherwise} \end{cases} \tag{3.2}$$

where e_r, e_t are the needed energy for received and transmitted operations, respectively. d is the Euclidean distance between two communicating sensors that are being considered. d_0 is a technical threshold for a transmitter and a receiver. Based on this energy model, with knowledge of how many bits each sensor receives and transmits, the BS can calculate the average energy consumption over a period. For simulation, this mean estimation is used as an input p_i for each $SN \in \mathbf{SN}$. Additionally, this thesis adopts the work in [50], all sensors have a minimum $e_{\min}$ and maximum $e_{\max}$ capacity. The energy e_i of S_i must stay between $e_{\min}$ and $e_{\max}$ during the period.

3.2 Charging model

The charging scheme during the period of $\mathcal{T}$ is divided into many *charging cycles*. Each charging cycle begins when the MC leaves the BS and visit a subset of **SN** in an order and ends when the MC returns to the BS. In the k-th charging cycle, the actions taken by the MC can be described as the sequence: $(CL_{\pi_{k,1}}, \tau_{k,1}) \rightarrow (CL_{\pi_{k,2}}, \tau_{k,2}) \rightarrow (CL_{\pi_{k,3}}, \tau_{k,3}) \rightarrow \cdots \rightarrow (CL_{\pi_{k,t_k}}, \tau_{k,t_k})$. By definition, the following must hold: $CL_{\pi_{k,t_k}} = CL_0$. The total time the MC spends at this charging cycle consists of two parts:

- The time spent on charging: $\mathcal{T}_{k,\text{charge}} = \sum_{i=1}^{t_k} \tau_i$.
- The time spent on travelling: $\mathcal{T}_{k,\text{travel}} = \sum_{i=1}^{t_k-1} \frac{d_{CL_{\pi_{k,i}}, CL_{\pi_{k,i+1}}}}{V}$.

The time spent for all charging cycles must add up to $\mathcal{T}$, i.e.:

$$\mathcal{T} = \sum_{k} \left(\mathcal{T}_{k,\text{charge}} + \mathcal{T}_{k,\text{travel}} \right) \tag{3.3}$$

This work extends the multi-point charging model in [20]. The main difference between a multi-point charging model and a point-to-point one is that at each time, the MC can provide multiple sensors simultaneously, whereas, in the latter one, the MC can only charge one sensor at each time. When staying at D_i, the per-second energy that each $S_j \in \mathcal{S}$ receives from the MC will be:

$$\rho_j^i = \frac{G_t G_r \mu}{L_p} \left(\frac{\lambda}{4\pi (d_{CL_i, SN_j} + \beta)^2} \right) \tag{3.4}$$

The equation (3.5) is also known as Friis's equation [51]. This can be rewritten as:

$$\rho_j^i = \frac{\alpha}{(d_{CL_i, SN_j} + \beta)^2} \tag{3.5}$$

where α, β are technical constants.

To sum up, the charging scheme optimization problem can be treated as a decision-making problem, where the MC (agent) interacts with the WRSN (environment). The MC can observe the state of the WRSN through various information, such as its current position, its charging rate to the sensors, and the sensors' remaining energy, etc. Based on these observations, the MC can choose the next CL to minimize the number of energy-depleted sensors. Naturally, this thesis proposes a novel Markov Decision Process (MDP) model to represent the interaction between the MC and the WRSN. This MDP model aims to construct a reward function with the same objective as the concerned problem. The output of the MDP is the optimal policy indicating the corresponding charging scheme should the MC take. The rest of this thesis uses the terms: MC and agent; WRSN and environment interchangeably.

3.3 MDP formulation

Table 3.1 denotes the abbreviations of the notions mentioned in this problem. This work formulates the problem as an MDP (denoted as $\mathcal{M}_{\text{WRSN}}$). Consider that the environment in $\mathcal{M}_{\text{WRSN}}$ is the network itself, and the interacting agent is MC. The agent perceives some information from the network, such as the location of sensors, the location of itself, the remaining energy of sensors, etc., and combines this knowledge into a full state representation. However, this representation is not taken directly into the learning process but extracted into an embedded status learned from a Graph Neural Network (GNN). From these embedding features, the agent conducts the next action, which is the next charging location. In detail, the components of $\mathcal{M}_{\text{WRSN}}$ can be described as follows:

- *State*: Each timestep, the agent observes the state as:

$$\hat{S}_t = \begin{bmatrix} e_1^t & e_2^t & \dots & e_n^t \\ d_{CL_j,SN_1} & d_{CL_j,SN_2} & \dots & d_{CL_j,SN_n} \end{bmatrix} \tag{3.6}$$

  A state graph can represent the transitions between states. To learn this graph, a GraphSAGE model is utilized and transforms $\hat{S}_t$ into embedding features S_t. These processes will be discussed in the next chapter.

- *Action*: The action of MC is represented as the next CL (or BS), i.e.:

$$A_t = \text{index of next location} \tag{3.7}$$

  A_t is a member of $\{0, 1, 2, \dots, m\}$. Note that this is just the abstraction of the actual action. Besides determining the next location, a practical action also

Table 3.1: List of Notations

Notation	Description
$\mathcal{N}$	The WRSN that are being considered
$\mathbf{CL} = \{CL_1, CL_2, \ldots, CL_m\}$	The set of charging locations
$\mathbf{BS} = CL_0$	The BS
$\mathbf{SN} = \{SN_1, SN_2, \ldots, SN_n\}$	The set of all sensors
$\mathcal{D}_i$	The set of sensors that MC can charge while staying at CL_i
$d_{A,B}$	The Euclidean distance between two locations A and B
r_c	The sensor's communication radius
$e^{\max}$	The sensor's maximum energy capacity
$e^{\min}$	The sensor's minimum energy capacity
e_j^t	The remaining energy of sensor SN_j at timestep t
$e_j^{\text{init}}, e_j^{\text{depot}}$	The residual energy of SN_j at the beginning and finishing time of a charging cycle
p_j	The average energy consumption rate of SN_j
ρ_j^i	The per-second energy rate that sensor SN_j receives from the MC when stopping at D_i
$E_{\text{MC}}^{\max}$	The maximum energy capacity of the MC
E_{MC}^t	The current energy of MC at timestep t
P_M	The per-second energy consumption rate of the MC when traveling
σ	The per-second energy that MC receives when charging at the depot
V	The MC's velocity
$\mathcal{T}$	Total time of the charging period
$\mathcal{T}_{\text{charge}}$	Total charging time of the MC
$\mathcal{T}_{\text{travel}}$	Total travelling time of the MC
E_{travel}	Total energy that the MC consumes for traveling
E_{charge}	Total energy that the MC consumes for charging

requires how long the MC will stay at that location. This work adapts a full charging strategy. More specific, denote τ_t as the time MC stays when taking action A_t, then the time interval MC will stay at CL_{A_t} can be defined as:

$$\tau_t = \begin{cases} \max_{j \in \mathcal{D}_{A_t}} \{\dfrac{e^{\max} - e_j^t}{\rho_j^{A_t} - p_j}\}, & \text{if } A_t \geq 1 \\ \dfrac{E_{\text{MC}}^{\max} - E_{\text{MC}}^t}{\sigma}, & \text{otherwise} \end{cases} \tag{3.8}$$

The first case is when the following location is a CL ($A_t \geq 1$), the agent charges the first nearby sensor to its max capacity. The second case is when the agent heads back to the BS, it will wait until its battery reaches maximum capacity.

- *Objective*: In order to design a good reward function, this work proposes an objective function that shows how the current situation of $\mathcal{N}$ might favor the MC. This objective has the form of:

$$F_t = \frac{\min_{i \in \{1,2,\ldots,n\}} e_i^t}{\max_{i \in \{1,2,\ldots,n\}} e_i^t} \tag{3.9}$$

  It is observable that $0 < F_t \leq 1, \forall t$. and the greater F_t is, the lower gap of network energy is. Moreover, moving from a low-value objective to a high-value one means the agent has found the sensor that currently has the minimum remaining energy, thus reducing the chance of node failure.

- *Reward*: The reward is defined as the difference between two states obtained by a transition, i.e.:

$$R_{t+1} = F_{t+1} - F_t \tag{3.10}$$

- *State transition*: As discussed previously, the environment in $\mathcal{M}_{\text{WRSN}}$ is deterministic. Therefore, the probability transition kernel is just 1 for the next state and 0 for other states. The environment can calculate exactly $\hat{S}_{t+1}$ from $\hat{S}_t$ with the assumption of A_t. The reward process is also deterministic. However, the formulation of (3.6) suggests that this state space is infinite, since each e_i is a continuous variable between $e^{\min}$ and $e^{\max}$.

- *Episode*: an episode is defined as when the MC does not have sufficient energy for the following action. This is because the network structure no longer exists when the MC runs out of energy. When the MC lacks the resources to visit the following charging location, it is forced to return to the BS and recharge to its total capacity. The new episode begins when it finishes charging.

CHAPTER 4. A HYBRID OF GRAPHSAGE AND DEEP Q-LEARNING ALGORITHM

This chapter details the proposed algorithm step-by-step. Overall, the algorithm consists of two key ideas: learning the embedding representation of the state space and training the agent to obtain the optimal policy for the formulated model.

As indicated in Chapter 3, the Markov Decision Process's (MDP) state space is an infinite high-dimensional space. This large-scale space makes the agent's learning process difficult and can lead to the underfitting problem. This proposal transforms the original state space into a more feasible, lower-dimensional space to overcome this issue. The proposed algorithm first constructs the state graph using discretize method to retain the connection between state transitions. Next, this thesis utilizes a state-of-the-art Graph Neural Network, namely GraphSAGE, to learn the representation of this state graph. After training the GraphSAGE model, an effective embedding mechanism is obtained to reduce the high-dimensional problem.

This work utilizes the method of Deep Q-Learning (DQL) to train the agent, which was first introduced in [36] and showed fast convergence to the optimal policy. DQL provides a reasonable approach to training the agent when the set of states is infinite. Furthermore, DQL is a model-free algorithm with low time complexity. This enables the calculation to be prompt and, therefore, suitable for practical deployment. Fig 4.1 illustrates the top-level design of the algorithm.

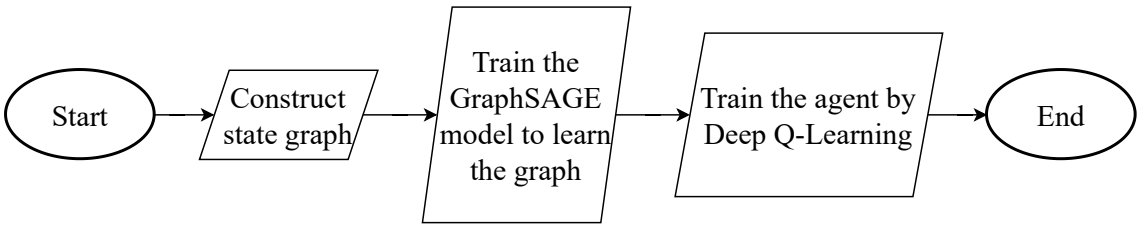

Figure 4.1: Overview of the proposed algorithm

4.1 State representation learning process

4.1.1 State graph construction

Consider time relative to the start of the charging schedule. First of all, the whole period $\mathcal{T}$ is divided by unit time interval Δ. In total, there are $\frac{\mathcal{T}}{\Delta}$ intervals and the elapsed time t is considered as $k\Delta$ with $k = \lfloor \frac{t}{\Delta} \rfloor$ (i.e. round t to the greatest time interval that not exceeds k). For each $CL_i \in \mathbf{CL}$, a cluster $\mathcal{C}_i$ is constructed, this cluster contains all status of CL_i during the charging period. Nodes inside this cluster have label i. More specific, $\mathcal{C}_i = \{C_i^0, C_i^1, C_i^2, \ldots, C_i^{\frac{T}{\Delta}}\}$. $C_i^t, j = \overline{0, \lfloor \frac{\mathcal{T}}{\Delta} \rfloor}$

represents the state of the network when MC are at CL_i at anytime in the interval $[t\Delta, (t+1)\Delta]$. The full state representation is used (3.6) here:

$$C_i^t = \begin{bmatrix} e_1^{t\Delta} & e_2^{t\Delta} & \dots & e_n^{t\Delta} \\ d_{CL_i,SN_1} & d_{CL_i,SN_2} & \dots & d_{CL_i,SN_n} \end{bmatrix} \tag{4.1}$$

To sum up, nodes in the constructed graph denote as C_i^t, with $i = \overline{0,m}; t = \overline{0, \lfloor \frac{\mathcal{T}}{\Delta} \rfloor}$ (in total, $(m+1)(\lfloor \frac{\mathcal{T}}{\Delta} \rfloor + 1)$ nodes). Each node has a feature presentation of (4.1), and nodes in a cluster have the same labels as the cluster.

To create edges between nodes in this graph, this work leverages time and distance relations. There will be an edge from C_i^t to $C_j^{t'}$ if:

$$t\Delta + \frac{d_{CL_i,CL_j}}{V} \in \left[t'\Delta, (t'+1)\Delta\right] \tag{4.2}$$

The relation (4.2) shows that at time t MC starts from CL_i and arrives at CL_j at approximate t'. An example of constructing edge from C_0^0 to other vertices is described as in Fig 4.2. These generated edges conceptually show the transition of states in the MDP regards to actions taken.

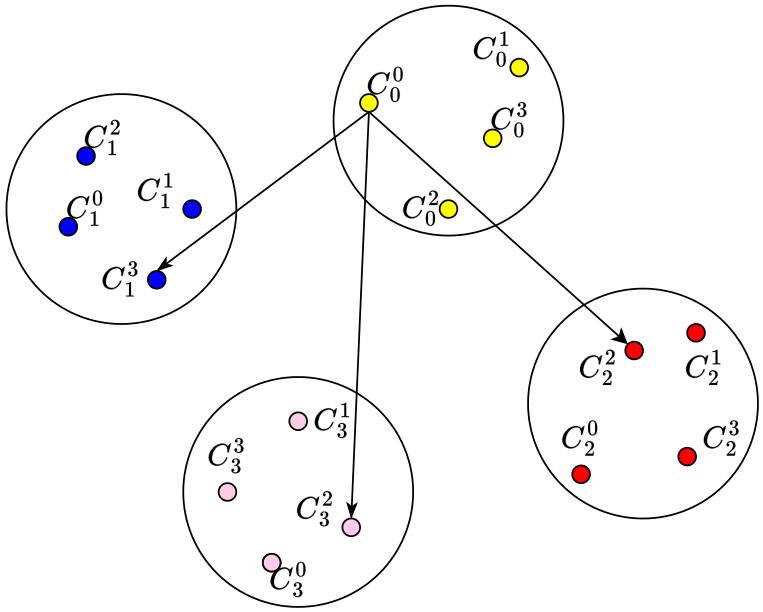

Figure 4.2: An example of constructing edges from C_0^0. In this example, suppose it takes 3 seconds to travel from CL_0 to CL_1, 2 seconds from CL_0 to CL_2, and 2 seconds from CL_0 to CL_3

4.1.2 A GraphSAGE-based algorithm for embedding state

This thesis leverages GraphSAGE from [45] to learn the representation of the graph. By constructing the state graph with the previous method, a supervised data-

set **SG** is obtained. Let the number of nodes in this dataset be $|V|$, the number of edges be $|E|$, and the number of features be $|F|$. This dataset consists of:

- Feature matrix: a $|V| \times |F|$ matrix shows initial representations of all nodes C_i^t.
- Edge index matrix: a $2 \times |E|$ matrix, where each column indicates edge from vertex i to j.
- Label: a vector with the number of features equal to $|V|$, each feature shows the corresponding label to node i.

SG is learnt directly by a GraphSAGE model, namely $\mathcal{G}$. After training, $\mathcal{G} : |F| :\rightarrow |F'|$, where $|F'| \ll |F|$, can be used as a predictor that takes full state representation as an input and outputs an abstract vector. The training process can be briefly depicted as in Algorithm 1.

Algorithm 1: StateGraphLearning($\mathcal{N}, \Delta, \mathcal{T}$)

Input: WRSN $\mathcal{N}$, unit time interval Δ, period time $\mathcal{T}$
Output: Learnt model $\mathcal{G}$ from the dataset

$T \leftarrow \frac{\mathcal{T}}{\Delta}$; `// Calculate number of unit intervals`
Initialize **SG**; `// Initialize the graph`
for $i \leftarrow 0$ *to* m **do**
  for $t \leftarrow 0$ *to* T **do**
    Get C_i^t by (4.1) Add vertex C_i^t to **SG**; `// Incrementally generate vertices set`
  end
end
for $i \leftarrow 0$ *to* m **do**
  for $t \leftarrow 0$ *to* T **do**
    for $j \leftarrow 0$ *to* m **do**
      if $j \neq i$ **then**
        time $\leftarrow \frac{d_{CL_i, CL_j}}{\overrightarrow{V}}$ $t' = \lfloor \frac{t\Delta + \text{time}}{\Delta} \rfloor$
        Add edge $\overrightarrow{C_i^t, C_j^{t'}}$ to **SG**; `// Incrementally generate edges set`
      end
    end
  end
end
$\mathcal{G} \leftarrow$GraphSAGE[45](**SG**); `// Utilize GraphSAGE to learn this graph`
return $\mathcal{G}$

4.2 A Deep Q-Learning-based algorithm

The rest of this proposal focuses on how to solve this MDP. Although this thesis has transformed to a smaller state space $|F| :\rightarrow |F'|$, the MDP remains infinite. To handle infinite state space, Deep Q-Learning, which was first introduced in [36], is utilized. A neural network is known for its capability of fitting any continuous functions [52]. This suggests choosing robust neural networks to fit the Q function. This work considers Recurrent Neural Network (RNN) for designing neural architecture. Each time the agent selects an action and executes to the environment, its state changes. These transformations can be thought of as a sequence of states. RNN can help find the long-term relations between hidden states and output actions, especially when the goal of the control problem in RL is to find optimal action responding to each state. GRU [53] architecture is leveraged in this work.

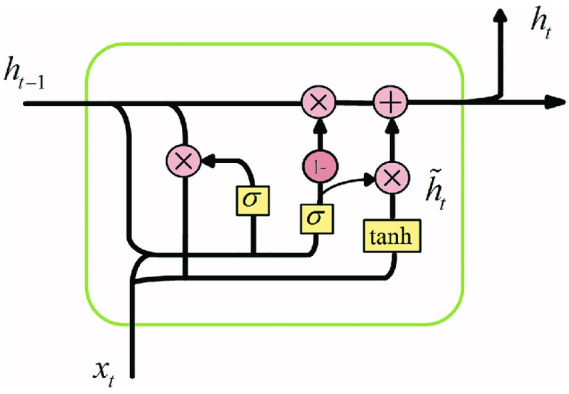

Figure 4.3: The structure of a cell in GRU [54]

Each cell in the RNN takes the embedding representation of the current state S_t and outputs the action-value function of this state. For example, the illustration in Fig 4.3 implies that $x_t = S_t$ and $C_t = (Q(S_t, 0), Q(S_t, 1), \ldots, Q(S_t, m))$. The main reason for choosing GRU as the neural network structure is because it encourages learning connections between states.

The proposed algorithm is based on the framework of Deep Q-Learning. In this problem, the energy of each sensor gradually decreases, and eventually, every sensor node can be energy-depleted if not receiving any charging. Therefore, a permutational exploration is much more suitable than the original ϵ-greedy in [36]. A list named *preference list* (denote by **PL**) keeps track of the unvisited CLs. For each exploration, a random CL is selected among the list. The list resets after it becomes empty or the episode ends. The full procedure of updating **PL** is described in Algorithm 2.

Algorithm 3, which illustrates the forecast for the spent energy of the next action, is processed before selecting any action. This procedure is to determine the termin-

Algorithm 2: UpdatePL(**PL**, A_t, terminate)

Input: Current preference list **PL**, current action A_t and terminate status
Output: Updated preference list

PL $\leftarrow$ **PL** $\setminus \{A_t\}$
if ***PL*** $= \emptyset \vee$ *terminate* **then**
  PL $\leftarrow \{1, 2, \ldots, m\}$
end
return PL

ation of the episode by checking whether the MC's energy is sufficient for the next action or not. The update rules for the policy and target networks of the MC are extended from [36], as in Algorithm 4. Finally, combine all these sub-procedures to obtain the proposed algorithm, as in Algorithm 5.

Algorithm 3: CheckSpareEnergy(A_{t-1}, A_t)

Input: Current action A_t and previous action A_{t-1}
Output: Whether the MC has adequate energy or not

Calculate charging time τ_t by (3.8)
$E_{\text{charge}} \leftarrow \sum_{j \in \mathcal{D}_i} \min\{\tau_i \rho_i^j; e^{\max} - e_j\}$
$E_{\text{travel}} \leftarrow P_M \times \frac{d_{CL_{A_{t-1}}, CL_{A_t}}}{V}$
return $E_{\text{MC}} > E_{\text{charge}} + E_{\text{travel}}$

Algorithm 4: OneBatchLearning(**RM**, $\Theta_{\text{policy}}, \Theta_{\text{target}}$)

Input: Replay memory **RM**, policy and target networks of the MC
Output: Update policy and target networks

Sample a batch $\mathcal{B}$ from **RM**
for $(S, A, R, S', terminate)$ *in* $\mathcal{B}$ **do**
  if ***terminate*** **then**
    $y \leftarrow R$
  else
    $y \leftarrow R + \gamma \times \max_{a' \in \{0,1,2,\ldots,m\}} \Theta_{\text{target}}(S', a')$
  end
  Perform gradient descent on Θ_{policy} by MSE: $(y - \Theta_{\text{policy}}(S, A))^2$
end

Algorithm 5: Charging policy determination

Input: The WRSN $\mathcal{N}$, learnt model $\mathcal{G}$ and the neural network model Θ
Output: Output the list of action $A_0, A_1, \ldots$ to solve the proposed MDP

Initialize replay memory **RM**, preference list, policy network **PL**, Θ_{policy}, target network Θ_{target}
for $i \leftarrow 1$ *to* $\ldots$; `// Loop through number of episodes`
do
  $t \leftarrow 1$; `// Initialize timestep`
  terminate $\leftarrow$ False
  while *not terminate;* `// Checking if the episode terminates`
  do
    Observe the current state $\hat{S}_t$ from $\mathcal{N}$
    $S_t = \mathcal{G}(\hat{S}_t)$
    r = Random(); `// Generate a random number in` $[0, 1]$
    if $r < \epsilon$ **then**
      Select random action A_t from **PL**
    else
      Choose $A_t \leftarrow \text{argmax}_{a \in \textbf{PL}} \Theta_{\text{policy}}(S_t, a)$
    end
    if *not CheckSpareEnergy*(A_{t-1}, A_t) **then**
      $A_t \leftarrow 0$; `// The MC is forced to go back to the BS`
      terminate $\leftarrow$ True
    end
    Transition $\hat{S}_t$ to $\hat{S}_{t+1}$ by A_t and yield R_t ; `// Transition to next state`
    Store $(S_t, A_t, R_t, S_{t+1}, \text{terminate})$ to **RM**
    OneBatchLearning(**RM**, $\Theta_{\text{policy}}, \Theta_{\text{target}}$); `// Call one-batch update`
    if *End of a episode* **then**
      terminate $\leftarrow$ True
    else
      $t \leftarrow t + 1$
    end
    PL $\leftarrow$UpdatePL(**PL**, A_t, terminate); `// Update preference list`
  end
  if $i \mod \lambda = 0$ **then**
    $\Theta_{\text{target}} \leftarrow \Theta_{\text{policy}}$; `// Periodically update target network`
  end
end

CHAPTER 5. EXPERIMENTAL RESULTS

This chapter conducts thorough experiments. After that, to see the effectiveness of the proposed algorithm, the performance of this work is compared with the state-of-the-art approaches. The algorithm is called "Hybrid of **G**raph**SA**GE and **D**eep **Q**-**L**earning Algorithm", or **GSADQL**.

5.1 Simulation settings

The experiments are set up by utilizing Python programming language and proceed on Google Colab Pro's servers (2vCPU @ 2.2GHz, with P100 GPU and 24GB RAM). Pytorch framework [55] is utilized for designing neural networks. The energy model is extended from the work in [26], as described in Table 5.1. The concepts of the energy model are presented in Chapter 4. Furthermore, the parameters of GSADQL figures in Table 5.2.

Table 5.1: Base parameters of the energy model

Parameter	Value
$E_{\text{MC}}^{\max}$	108000 (J)
P_M	1 (J/s)
V	5 (m/s)
α	3600
β	30
$e^{\max}$	10800 (J)
$e^{\min} = 0.05\ e^{\max}$	540 (J)
$\mathcal{T}$	200000 (s)
σ	100 (J/s)

Table 5.2: Parameters of GSADQL

Parameter	Value
γ	0.95
Target Update	10
Number of episodes	50
Δ	40

This work considers networks with the number of sensors ranging from 200 to 400 deployed in a 2-D area. This area is obstacles-free and has a fixed size of $500 \times 500m^2$. The BS is located at coordinate $(250, 250)$. In the simulated process, the region is first split into 100 unit grids; each one is a $50 \times 50m^2$ zone. Then, for each zone, sensors' locations are generated by a uniform distribution. After creating the WRSN, the network operation is simulated based on [19] as well as the energy model presented in (3.2) to obtain sensors' energy parameters. These parameters will be written in each file.

For each number of sensor nodes varies from 200 to 400, a dataset is produced. Each dataset has 10 data instances. In total, there are 50 input files, and the performances of algorithms are weighted on these files. The format of an input file is as follows: `g_{number_of_sensors}_{tokens}`, in which prefix `g` is

abbreviation for grid, two suffices represents the number of sensors and the file index, respectively. The file index helps to distinguish instances among a dataset. For example, `g_300_8` is the eighth instance of 300 nodes dataset.

To measure the efficiency of the proposed algorithm, two benchmark baselines with a similar objective of minimizing nodes failure ratio are chosen:

- INMA: Invalid Node Minimized Algorithm [23], known as an on-demand, online, point-to-point charging algorithm. This algorithm handles requests from sensors that are lower than a threshold. Selection of the next sensor to be charged considers the remaining energy and the distance to MC.
- GR-GACS: GRreedy Genetic Algorithm-based Charging Schedule [18], an offline, multi-point charging algorithm. The algorithm first determines charging locations by a greedy mechanism. The second phase of this work utilizes evolutionary algorithms to optimize the node failure ratio.
- No charging: An upper bound for any charging algorithms indicates how many nodes will be deactivated if there is no maintenance.

The metric that is evaluated here is the node failure ratio. A scheduling algorithm that results in a lower node failure ratio proves to save more sensors; therefore, less network information is lost, and the lifetime is prolonged. The node failure ratio is the proportion of the number of energy-depleted sensors to the number of deployed sensors. For a reliable result, the period length is identical to all algorithms.

$$\text{The node failure ratio } (\%) = \frac{\text{Number of energy-depleted sensors in a period}}{\text{Number of deployed sensors}} \times 100\% \tag{5.1}$$

5.2 Training process

This section discusses the performance of the GraphSAGE model and the proposed algorithm over the datasets. First, a study on the state graph learning process by the GraphSAGE model is conducted and then taken into solving proposed MDP by GSADQL. Each experiment runs 10/20 times over an data instance.

5.2.1 Learning state representations in GraphSAGE

Overall, GraphSAGE models can learn the state graphs in all data instances. This part of the thesis illustrates one example of the graph learning process. Specifically, the training process for data instance `g_200_7` is considered, with the base settings in Table 5.2.

The ability to ultimately learn the state graph is shown in Fig 5.1. After about 70 epochs, the Negative LogNLL loss approaches 0, and accuracy acquires 100%.

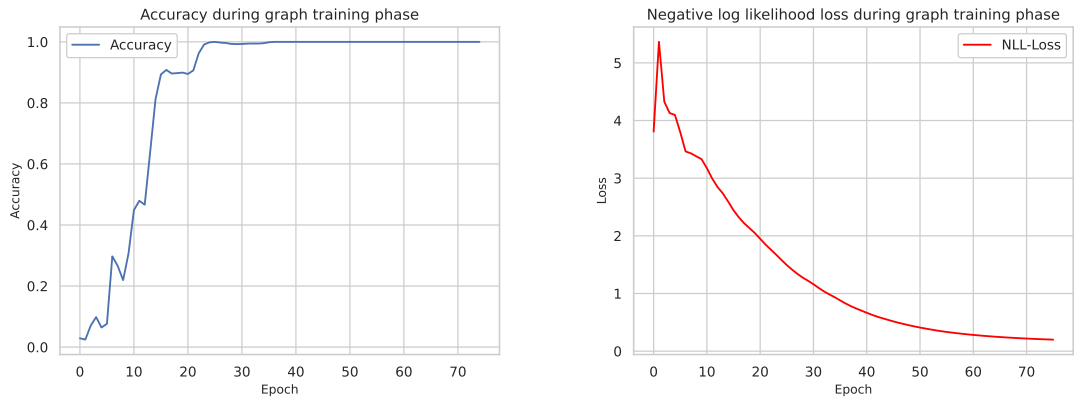

Figure 5.1: Training process of GraphSAGE model in `g_200_7`

This is the perfect score for the supervised task; it tells us that the GraphSAGE model absolutely learns the graph. Furthermore, in other data instances, this perfect score is also achieved. A natural question is whether the accuracy metric is inadequate and can not represent the model's generalization. However, this thesis is only concerned with accuracy and loss function because generalization is unnecessary. Each data instance has its own approximate model $\mathcal{G}$, and each learned $\mathcal{G}$ could be used for only one instance. Hence, as long as $\mathcal{G}$ can fit perfectly the state space of a problem, it is sufficient for further usage, i.e., predicting state representation for stated MDP problem.

5.2.2 Learning process of GSADQL

This part focuses on the learning process of GSADQL. In detail, this part wants to study the accumulative reward of the agent as well as loss function during the training process. Measurements are taken over the same data instance as Subsection 5.2.1. Again, the based parameters in Table 5.2 are used; however, $\mathcal{T}$ is adjusted to 500000. The purpose of this extension is that more episodes need to be observed, and the period length is proportional to the number of episodes.

Experiments are run on data instance `g_200_7` for 20 times. Fig 5.2 shows the loss curve obtained by training the agent. The metric used here is Huber loss. These curves are almost identical, begin from approximately 0.30 and converge to 0.00. This characteristic indicates the approximation in 2.12 approaches 0, and a near-optimal policy of formulated MDP problem is acquired, based on the theory of (2.8).

Since GSADQL is an online algorithm, it is not surprising that the number of episodes each run is slightly greater than 14. For each episode, the average accumulative rewards of 20 runs are recorded, and the result is illustrated in Fig 5.3. As can be seen from this figure, the return value tends to increase, which is the

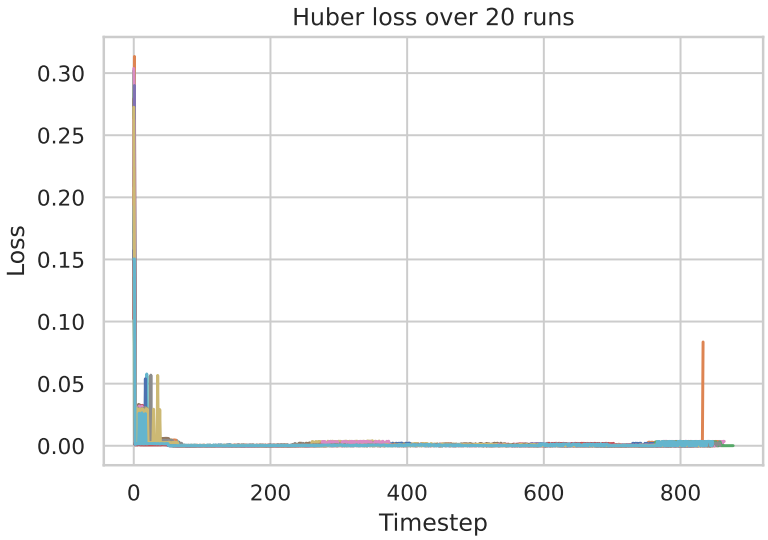

Figure 5.2: Loss function of training agent with data instance `g_200_7` over 20 runs

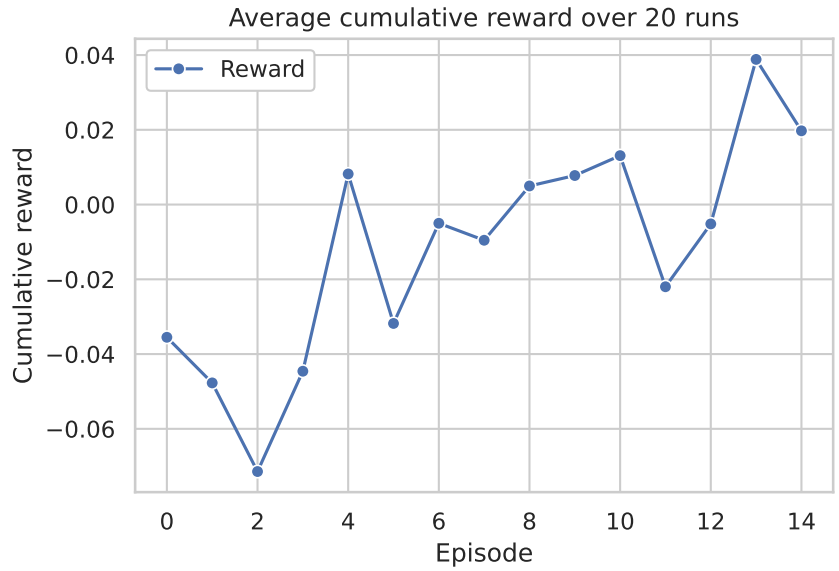

Figure 5.3: Average cumulative reward of agent with data instance `g_200_7` over 20 runs

ultimate goal in any RL problem. Nevertheless, the increasing trend is not monotone but oscillatory. This is because of the exploitation-exploration trade-off in GSADQL. Notably, in order to achieve a near-optimal policy, the agent has to explore the action space. The cumulative reward decreases when it chooses the wrong action regarding a state. Nevertheless, in the long run, the agent is able to learn the near-optimal policy and sustain accumulative rewards at a higher rate; for example, since 4^{th} episode, the average accumulative reward is greater than the beginning.

5.3 Empirical results

This section shows the results of conducted experiments. First, a study of an algorithm hyperparameter is presented. Next, comparisons with other benchmarks

highlight the proposal's effectiveness, including statistical analysis and qualitative results in several scenarios.

5.3.1 Impact of Δ on the performance of the proposed algorithm

Recall that the size of the constructed graph in Subsection 4.1.1 is $\mathcal{O}((m\frac{T}{\Delta})^2)$, i.e., the number of edges and Δ^2 are in an inverse proportion. Thus, when Δ increases, the graph will become significantly smaller in terms of size, and therefore, computation is much more efficient. However, reducing the size of graphs could lead to inaccurate representations of the state graph. Firstly, the performance by varying Δ from 20 to 100 can be observable in the Box plot 5.4: This figure shows

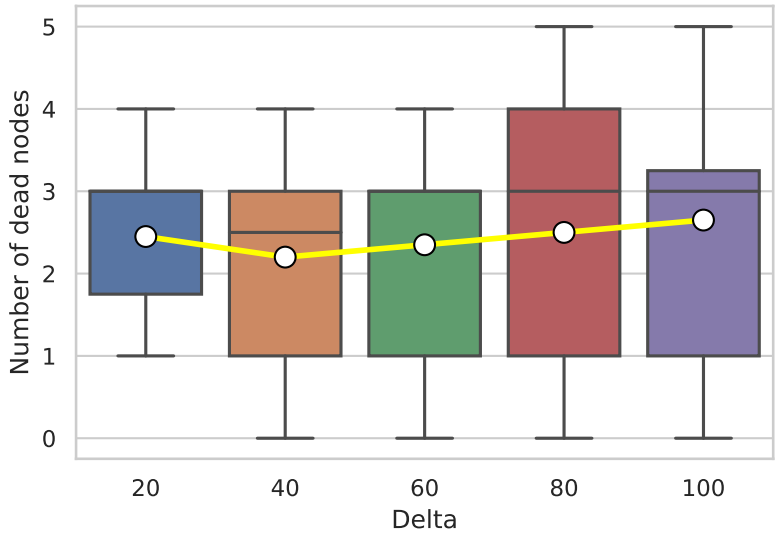

Figure 5.4: Impact of Δ on the proposed algorithm

the number of dead nodes obtained by the proposed algorithm when Δ changes as in the x-axis. Each box shows the distribution of the metric. There are three lines in each box, which represent the first, second (median), and the third quartile of the distribution; there are two lines outside the box, which illustrate the min and max of the results. The white dot shows the average number of dead nodes by 20 runs.

This figure tells that when Δ grows, the variation of the result increases. This is because of the greater Δ, the smaller size of the state graph. Therefore, when the size reduces, the search space of the problem shrinks, which leads to insufficient information for the agent to find the optimal solution. Eventually, this results in the fact that the learning process of the agent becomes fragile and unstable, which can be seen in larger gaps between the first and third quartiles when Δ is greater. Followed by this intuition, Δ should be small enough so that the state graph can retain rich information for the agent to learn.

The mean number of dead nodes bottoms out at 2.2 when $\Delta = 40$, followed by

an increase from 2.2 to 2.65 when Δ grows from 40 to 100. This can be explained as follows when Δ is too small, the graph becomes too complex, and therefore, the agent is unable to find an optimal solution with a learning mechanism of an incremental update, therefore, the metric when $\Delta = 20$ is less than when $\Delta = 40$. When Δ is large, the learning process of the agent is much more fragile and which could lead to more number of dead nodes.

To sum up, Δ should be chosen in a reasonable method. The above result suggests that $\Delta = 40$ should be picked for these datasets.

5.3.2 Statistical analysis

In order to collect credible comparisons, statistical analysis is a popular choice. It provides thorough examinations as well as calculations for comparing algorithms. Three statistical tests are conducted to weigh the proposed algorithm up against other methods. In particular, sample runs over the datasets by GSADQL, INMA, and GR-GACS are combined, then the off-the-shelf Friedman [56], Wilcoxon [57], and contrast estimation [58] tests are used to evaluate the obtained samples.

Table 5.3: Average Rankings of the algorithms (Friedman)

	Mean Rank
GSADQL	1.53
INMA	2.53
GR-GACS	1.95

Table 5.4: Test Statistics (Friedman)

N	500
Z	269.508
df	2
p-value	2.9997e-59

Friedman test is widely utilized to identify distinctions among treatments by a variety of methods. The null hypothesis H_0 in Friedman test is the treatments of all methods are identical. Per Table 5.3, the mean ranks of the three algorithms are significantly different: GSADQL has the best rank of 1.53, the rank of GR-GACS is the runner-up with a value of 1.95 and finally, INMA has a rank of 2.53. Table 5.4 shows that p-value is 2.9997e-59, the degree of freedom is 2 and Z (chi-square value) is 269.508. With a degree of significance $\alpha = 0.05$, the null hypothesis can be rejected since 2.9997e-59 $\lll$ 0.05. In other words, it is justifiable to say that there are differences among the results provided by three algorithms with confidence of 95%, which is normally known as an acceptable level.

Table 5.5: Ranks (Wilcoxon Signed Ranks Test)

		N	Mean Rank	Sum of Ranks
INMA - GSADQL	Negative Ranks	99	163.11	16148.00
	Positive Ranks	354	244.87	86683.00
	Ties	47		
	Total	500		
GR-GACS - GSADQL	Negative Ranks	114	188.81	21524.50
	Positive Ranks	333	236.05	78603.50
	Ties	53		
	Total	500		
GR-GACS - INMA	Negative Ranks	370	230.36	85235.00
	Positive Ranks	100	254.50	25450.00
	Ties	30		
	Total	500		

Table 5.6: Test Statistics (Wilcoxon Signed Ranks Test)

	INMA - GSADQL	GR-GACS - GSADQL	GR-GACS - INMA
Z	-12.698[a]	-10.497[a]	-11.107[b]
p-value (2-tailed)	6.0333e-37	8.9614e-26	1.15739e-28

[a] Based on negative ranks
[b]Based on positive ranks

The second test this thesis takes account into is Wilcoxon signed-rank test. The purpose of this mechanism is to show comparisons between pairs of competitors. For each comparison, negative ranks show how many times the first algorithm produced worse results while positive ones indicate occasions the first algorithm produced better results. According to Table 5.5 and 5.6, the following results are obtained:

- INMA - GSADQL: 99 times INMA provides better results while 354 times GSADQL have superior outcomes. There are 47 times both algorithms output the same number of dead nodes. In short, GSADQL often produces upper results. The p-value of this comparison is 6.0333e-37, which is insignificant.
- GR-GACS - GSADQL: There are 114 times metric by GR-GACS that is better than GSADQL. On the other hand, 333 times GSADQL surpasses GR-GACS. They are tied on 53 occasions. A conclusion is followed that GSADQL often produces lower node failure ratios. In this case, the p-value is 8.9614e-26, which is relatively small.
- GR-GACS - INMA: GR-GACS seems to outperform INMA in the given data-sets. On 370 data instances, GR-GACS determines fairer objectives while

there are 100 times INMA provides better outcomes. On 30 occasions, they draw. The p-value of this statistical test is 1.15739e-28, which is also insignificant.

Since the obtained p-values of all tests are way below $\alpha = 0.05$, thus, it is reasonable to decline null hypotheses. This implies that GSADQL has the best results among the three algorithms with a confidence interval of 0.95, also, GR-GACS is the second-best while INMA is at the lowest position.

	GSADQL	INMA	GR-GACS
GSADQL	0.000	-2.333	-1.667
INMA	2.333	0.000	0.6667
GR-GACS	1.667	-0.6667	0.000

Table 5.7: Contrast Estimation

Finally, this thesis discusses the result of contrast estimation. This test measures the extent between couples of mentioned methods. Table 5.7 shows that the contrast estimations of GSADQL are negative (-2.333 compared to INMA and -1.667 compared to GR-GACS). This test tells that most of the time, GSADQL achieves the best result among the three algorithms.

5.3.3 Comparison with other benchmarks

The rest of this thesis shows the results of comparing the proposed algorithm with the others. To examine the performance of each algorithm, this thesis considers changing the most vital parameters to the network: number of nodes, battery capacity of MC, period length, charging rate of MC, and consumption rate of sensors. For each scenario, the metric node failure ratio is considered to measure the efficiency of each benchmark.

a, Impact of the number of nodes

This thesis first studies the impact of the number of nodes on the node failure ratio. The tendency that can be seen in this scenario (Fig 5.5) is that number of deactivated nodes is proportional to the number of dead nodes. This can be explained as follows, as the network grows in size, more packets need to be transferred. When more communication is needed, recall (3.1) and (3.2), the mean energy consumption rate increases, which results in more energy-depleted sensors. The number of dead nodes in case of no maintenance (shown by the line "No charging"), slightly rises from 16 to 32 between 200 and 400 nodes and the corresponding node failure ratio grows from 8% to 9%.

The demonstrated result by INMA is inferior to the other algorithms. In all

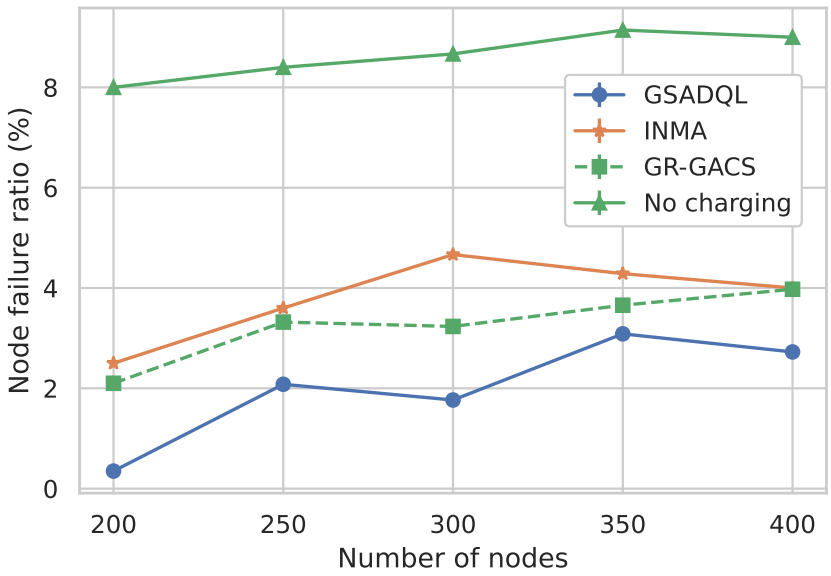

Figure 5.5: Impact of the number of nodes

cases, the node failure ratio outputted by INMA is the highest among comparing algorithms. On average, the performance of INMA is 1.80% and 0.05% less than GSADQL and GR-GACS, respectively. This shows the disadvantages of a point-to-point charging scheme, compare to a multi-point charging scheme. In the former scheme, the MC can only charge one sensor at a time, which leads to more visitations being required than in the latter scheme. Additionally, compare to GR-GACS, INMA adopts a fully-charging method, so the time spent at each location is longer, which means other sensors need to wait longer and as a consequence, more nodes die out while pending for replenishing energy.

When weight GR-GACS and GSADQL against each other, the online charging mechanism seems to accomplish better than the offline one. The main reason lies in the ability to adapt the network structure of a neural network. On the other hand, GR-GACS relies on a fixed strategy from the beginning and can not determine the changing structure of the network. Thus, in a long term, GSADQL identifies more perilous nodes than GR-GACS and results in smaller node failure ratio.

The proposed algorithm has the best result. In a 200-node instance, it has an impressive result of 0.035%, which outperforms other algorithms with a margin at least of 1.75%. As the nodes increase from 250 to 350, the gap with the second-best algorithm (GR-GACS) is narrower, it reduces from 1.24% to 0.57%. The gap bounces back to 1.25% in a 400-node instance. Generally, the average saving nodes ratio is 6.67% (approximate 20 sensors) when compared to no charging. This result suggests that GSADQL is effective in a highly-populated WRSN, thanks to the multi-point charging scheme and the ability to adapt the network structure.

b, Impact of the battery capacity of MC

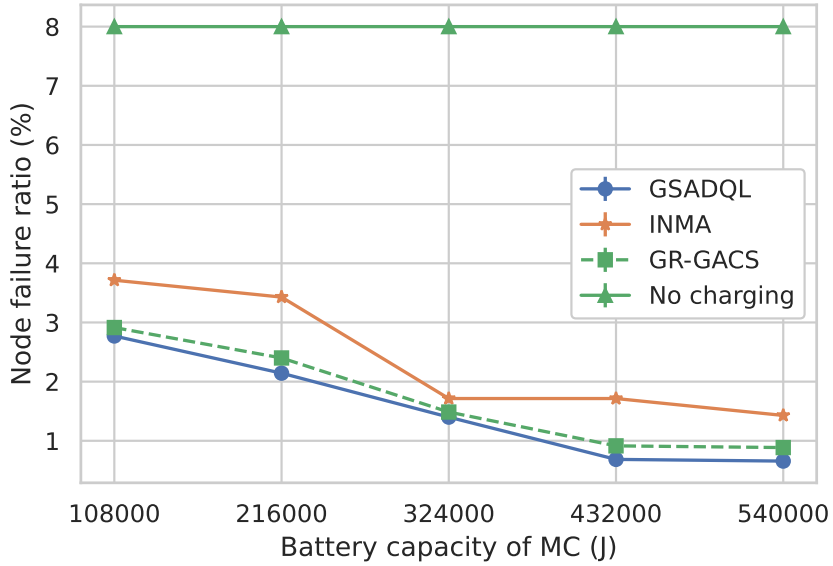

Figure 5.6: Impact of the battery capacity of MC

The battery capacity of MC is also vital to WRSN because it decides the quantities of transmitted energy to sensors. This section investigates the impact of this aspect. As the batter capacity of MC increases, the node failure ratios by all algorithms are opt to decrease. Clearly, this does not affect the number of dead nodes if the network is uncharged, therefore, all metrics have an upper bound constant line "No charging" with a value 8.00%.

In this scenario, INMA underperforms the other algorithms. The node failure ratio by INMA keeps a margin of 0.79% greater than the second-worst GR-GACS. On the occasion of 324000-jule-E_{MC} only, the margin closes to 0.23%. This tells us that in multi-point charging schemes like GR-GACS and GSADQL, the utilization of battery capacity is better than in the point-to-point charging scheme. Since the at a time, the MC can charge multiple nodes concurrently, the need for travelling is less, and thus, charging energy is much more effective. Additionally, there are fewer charging locations in a multi-point scheme than in a point-to-point one and these locations seem to stay closer to BS.

Comparing two multi-point charging algorithms, it is noticeable that GSADQL is better than GR-GACS, but it is an inconsiderable difference. GSADQL uses an adapting method to discover endangered sensors, while GR-GACS uses a fixing one. The first method can improve through time and that is why GSADQL seems to find more critical nodes and spend more proper charging energy for these energy-needed sensors.

Node failure ratio by GSADQL is the lowest, as shown in Fig 5.6. Specifically,

the gaps between measurements GSADQL and GR-GACS remain steadily at a rate of 0.18% when raising E_{MC} from 108000J to 540000J. One interesting point that can be seen from this figure is the node failure ratios of both algorithms do not change when E_{MC} varies from 432000J to 540000J. This point indicates that when the battery capacity is sufficiently large, two algorithms appear to converge to a fixed node failure ratio, i.e. large battery capacity will no longer have a strong impact on failure nodes.

c, Impact of the period length

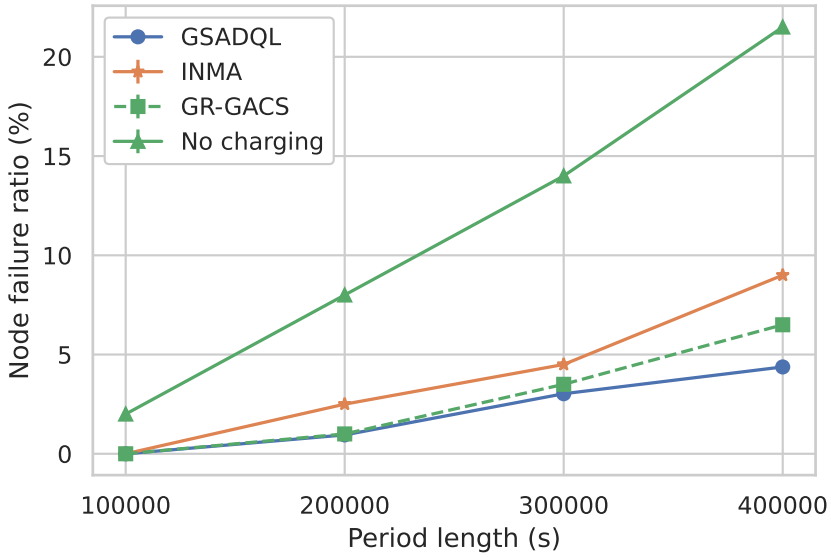

Figure 5.7: Impact of the period length

Next, this section looks into the performances of all algorithms when extending the period length, which are illustrated as in Fig 5.7. Apparently, when time is stretched, more nodes will have their energy decreased, which results in more number of deactivated sensors. All algorithms are beneath a linear-like line "No charging", with ratios stably going up from 2.4% to 21.5%. Similarly, node failure ratios by all algorithms level up when the period length extends and these incrementations are also linear-like.

Again, the node failure ratio by INMA is observable as the greatest. All algorithms output a node failure ratio of 0% in case the period length is 100000s. When the period length alters from 200000s to 300000s, INMA keeps differences of 1.51% and 1.25% with GSADQL and GR-GACS, respectively. When the period length ups to 400000s, the differences with GSADQL and GR-GACS peak at 4.63% and 2.5%, in the given order. Compared to GR-GACS and GSADQL, INMA only charges one sensor at a time and spends more energy on charging than the others. Through time, while more sensors deplete energy, these unproductive

actions made by the MC accumulate, and energy is not used for saving dying nodes.

GSADQL has a slightly better result than GR-GACS. Node failure ratios by GSADQL are less than GR-GACS only 0.05% when the period lasts 200000s and 0.48% when the period lasts 300000s. Up to 400000s, the gap is visible, at a ratio of 2.13%. As the period length is longer, GSADQL seems to save more energy-depleting sensors than GR-GACS. This happens in light of the precious knowledge learned during early episodes of the GSADQL agent. Because of the capability of exploiting the topology of the network, the intelligent agent proves to have a better strategy for scheduling the whole network in a long run than a fixed metaheuristic method. This statement is consolidated by comparing the node failure ratio by GSADQL to no charging plan, this rate increases from 2% to the peak ratio of 17.13% when the period extends from 100000s to 400000s.

d, Impact of the charging rate of MC

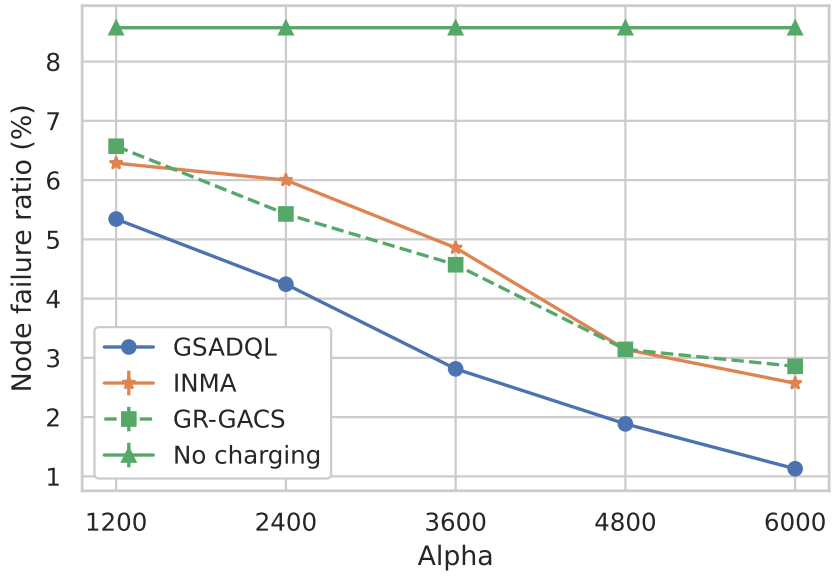

Figure 5.8: Impact of the charging rate α (Alpha) of MC

Another influence of the MC beside battery capacity is its charging rate. Recall the charging rate by (3.5), it is obvious that the gained energy rate of sensors is in proportion to α. As α grows, these gained rates also increase, consequently, the node failure ratios by all algorithms tend to drop. Similar to Section b, a constant line with values of 8.57% serves as an upper bound for all algorithms, because α does not affect the network if it receives no charging at all.

In this scenario (Fig 5.8), INMA is lower in terms of rank, compared to GR-GACS. When the value of α gets 1200 or 6000, the metrics obtained by INMA are insignificantly lower than GR-GACS, with gaps of 0.29% in both cases. When α increases from 2400 to 4800, there is also no noticeable difference between these

two algorithms, GR-GACs produce a node failure ratio 0.29% less than INMA. Therefore, this supports that with predetermined strategies, when growing linearly, the point-to-point charging scheme appears to utilize the charging rate better than multi-point, so it has overcome the mentioned disadvantages in previous sections. To clarify this, recall the equation (3.5) for received energy of sensors:

$$\rho_j^i = \frac{\alpha}{(d_{CL_i,SN_j} + \beta)^2} \tag{5.2}$$

Due to this, the gain charging utilities in the multi-point charging scheme are much smaller than in the point-to-point charging scheme, therefore, when growing linearly, the point-to-point charging scheme makes more use of α.

However, compare to GSADQL, the other algorithms underperform. This is because GR-GACS and INMA use a fixed mechanism that can only focus on critical sensors, while do not utilize the high charging rate to adjust charging time for other nodes. On the other hand, GSADQL uses an agent that is perceivable of changing environment, when the charging rate increases, the intelligent agent can also know how to balance between other nodes. Eventually, the exploitation of GSADQL is much more remarkable.

Once again, GSADQL outputs the best objective among the participants. The margins between GSADQL and INMA and GR-GACS are observable. On average, node failure ratio by GSADQL sustains a margin of 1.32% with GR-GACS and a margin of 1.48% with INMA. The result when α gets a value of 6000 is quite persuasive, only 1.12% of the network reaches its termination. In other words, GSADQL saves 7.44% when compared to no appearance of the MC.

e, Impact of the consumption rate of sensors

Finally, the last but not least important factor is considered, which is the consumption rate of sensors. Fig 5.9 illustrates the node failure ratios by the competitors, the x-axis represents the multipliers of original consumption rates. For example, a ratio of 0.5 indicates that the original consumption rates of all sensors are multiplied by 0.5. When the consumption rates grow, energy is faster in exhausting energy, which inevitably leads to more dead nodes. The upper bound is the linear line "No charging", ranging from 2.86% to 24.86%.

In contrast to other scenarios, GR-GACS produces the highest node failure ratio. When the consumption rate ratio gets a value of 0.5, all algorithms output 0 dead nodes. In the case of 1.0-ratio, GR-GACS is just slightly lower than INMA with a gap of 0.29%, while varying from 1.5 to 2.5, INMA performs better than GR-

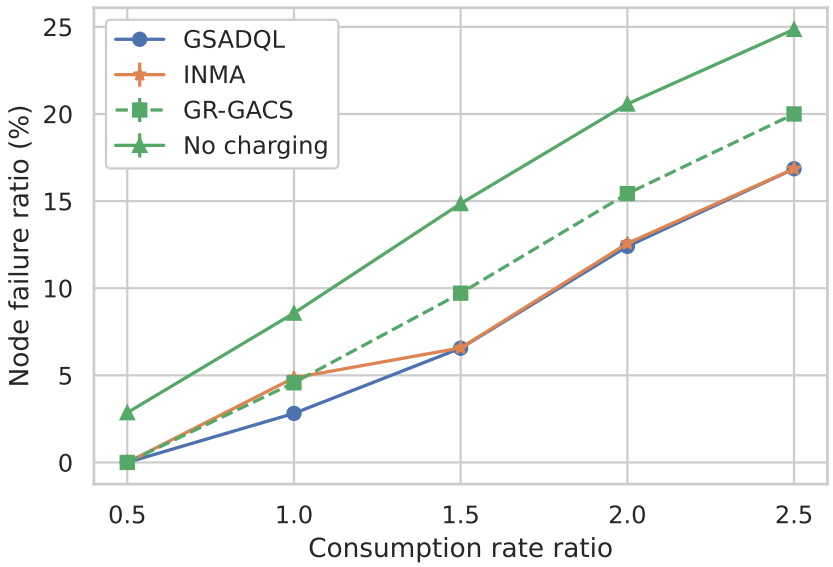

Figure 5.9: Impact of the consumption rate of sensors

GACS, which conserves a gap of 2.85%. In this scenario, online charging seems to be superior. This is because when consumption rates enlarge, sensors are dying out more quickly, and thus, charging requests are lining up more, and online charging schemes resolve this issue much more efficiently than offline ones.

Nevertheless, INMA still can not outperform GSADQL, although there is literally no difference between the two algorithms when the ratio alters from 1.5 to 2.5. The superiority of GSADQL to INMA in case of the 1.0-ratio shows that in a proper setting, the self-learning agent benefits more from the consumption rate because of a flexible critical-energy threshold rather than a fixed threshold. However, in a long run, when the consumption rate overflows, adapting strategy appears to converge to the predefined one because there is less time for the MC to handle.

GSADQL still has the highest rank in this particular scenario. When the ratio changes from 1.0 to 2.5, the node failure ratio by GSADQL maintains a margin of average 2.77% with GR-GACS, while growing from 1.5 to 2.5, a small difference of 0.18% is kept between GSADQL and INMA. In light of the ability to learn the structure of the network, this online and adapting method determines seriously endangered sensors and therefore, handles this high-consuming scenario in an acceptable result.

CONCLUSION

This thesis tackles the problem of minimizing the energy depletion in WRSN by leveraging an online multi-node charging scheme that optimizes the charging route. A novel MDP model is also proposed to give the foundation for solving the charging scheme optimization problem by RL methods. This thesis proposes a GraphSAGE and Deep Q-Learning hybrid algorithm, which finds the optimal policy for the modelled MDP. The results of thorough experiments suggest that the proposal outperforms other state-of-the-art methods which have the goal of minimizing the number of energy-depleted sensors.

However, there are still setbacks in this thesis. The proposed algorithm works well in the scenario of dense networks, however, performs much worse when the network is less populated. Additionally, the online algorithm desperately demand a long period length, and behaves badly in the opposite case. Although the assumptions made in this thesis are based on previous work, nevertheless, it is still not straightfoward to deploy such systems in practice.

In the future, the author will address and adjust the Markov model to study the scenarios when the network has multiple MCs and mobile sensor nodes.

Parts of this thesis were accepted as papers of 2020 IEEE Congress on Evolutionary Computation (CEC):

Huong, T. T., Binh, H. T. T., Le Nguyen, P., Long, D. C. T., & An, V. D. (2020, July). Optimizing charging locations and charging time for energy depletion avoidance in wireless rechargeable sensor networks. In 2020 IEEE Congress on Evolutionary Computation (CEC) (pp. 1-8). IEEE.

REFERENCE

[1] I Fakilidz *et al.*, "Wireless sensor network: A survey," *Computer Networks*, no. 38, pp. 393–422, 2002.

[2] D. Estrin, R. Govindan, J. Heidemann and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, 1999, pp. 263–270.

[3] J. M. Kahn, R. H. Katz and K. S. Pister, "Next century challenges: Mobile networking for "smart dust"," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, 1999, pp. 271–278.

[4] A. Chandrakasan, R. Amirtharajah, S. Cho *et al.*, "Design considerations for distributed microsensor systems," in *Proceedings of the IEEE 1999 Custom Integrated Circuits Conference (Cat. No. 99CH36327)*, IEEE, 1999, pp. 279–286.

[5] P. Bonnet, J. Gehrke and P. Seshadri, "Querying the physical world," *IEEE personal Communications*, vol. 7, no. 5, pp. 10–15, 2000.

[6] M. Suzuki, S. Saruwatari, N. Kurata and H. Morikawa, "A high-density earthquake monitoring system using wireless sensor networks," in *Proceedings of the 5th international conference on Embedded networked sensor systems*, 2007, pp. 373–374.

[7] E. M. Petriu, N. D. Georganas, D. C. Petriu, D. Makrakis and V. Z. Groza, "Sensor-based information appliances," *IEEE Instrumentation & Measurement Magazine*, vol. 3, no. 4, pp. 31–35, 2000.

[8] J. M. Rabaey, M. J. Ammer, J. L. Da Silva, D. Patel and S. Roundy, "Picoradio supports ad hoc ultra-low power wireless networking," *Computer*, vol. 33, no. 7, pp. 42–48, 2000.

[9] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," *Communications of the ACM*, vol. 43, no. 5, pp. 51–58, 2000.

[10] K. E. Ukhurebor, I. Odesanya, S. S. Tyokighir, R. G. Kerry, A. S. Olayinka and A. O. Bobadoye, "Wireless sensor networks: Applications and challenges," *Wireless Sensor Networks-Design, Deployment and Applications*, 2020.

[11] G. Anastasi, M. Conti, M. Di Francesco and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad hoc networks*, vol. 7, no. 3, pp. 537–568, 2009.

[12] H. Ba, I. Demirkol and W. Heinzelman, "Passive wake-up radios: From devices to applications," *Ad hoc networks*, vol. 11, no. 8, pp. 2605–2621, 2013.

[13] D. Tulone and S. Madden, "Paq: Time series forecasting for approximate query answering in sensor networks," in *European Workshop on Wireless Sensor Networks*, Springer, 2006, pp. 21–37.

[14] K. S. Adu-Manu, N. Adam, C. Tapparello, H. Ayatollahi and W. Heinzelman, "Energy-harvesting wireless sensor networks (eh-wsns) a review," *ACM Transactions on Sensor Networks (TOSN)*, vol. 14, no. 2, pp. 1–50, 2018.

[15] S. Khriji, D. El Houssaini, I. Kammoun and O. Kanoun, "Energy-efficient techniques in wireless sensor networks," in *Energy Harvesting for Wireless Sensor Networks: Technologies, Components and System Design*, De Gruyter Oldenbourg Berlin, Germany, 2018.

[16] F. Shen, W. Cui, W Ma, J. Huangfu and L. Ran, "Circuit analysis of wireless power transfer by "coupled magnetic resonance"," in *IET International Communication Conference on Wireless Mobile and Computing (CCWMC 2009)*, IET, 2009, pp. 602–605.

[17] W. B. Heinzelman, A. P. Chandrakasan and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on wireless communications*, vol. 1, no. 4, pp. 660–670, 2002.

[18] T. T. Huong, H. T. T. Binh, P. Le Nguyen, D. C. T. Long, V. D. An *et al.*, "Optimizing charging locations and charging time for energy depletion avoidance in wireless rechargeable sensor networks," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2020, pp. 1–8.

[19] T. T. Huong, P. Le Nguyen, H. T. T. Binh, K. Nguyenz, N. M. Hai *et al.*, "Genetic algorithm-based periodic charging scheme for energy depletion avoidance in wrsns," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2020, pp. 1–6.

[20] X. Yang, G. Han, L. Liu, A. Qian and W. Zhang, "Igrc: An improved grid-based joint routing and charging algorithm for wireless rechargeable sensor networks," *Future Generation Computer Systems*, vol. 92, pp. 837–845, 2019.

[21] B. Qureshi, S. A. Aziz, X. Wang *et al.*, "A state-of-the-art survey on wireless rechargeable sensor networks: Perspectives and challenges," *Wireless Networks*, pp. 1–25, 2022.

[22] C. Lin, G. Wu, M. S. Obaidat and C. W. Yu, "Clustering and splitting charging algorithms for large scaled wireless rechargeable sensor networks," *Journal of Systems and Software*, vol. 113, pp. 381–394, 2016.

[23] J. Zhu, Y. Feng, M. Liu, G. Chen and Y. Huang, "Adaptive online mobile charging for node failure avoidance in wireless rechargeable sensor networks," *Computer Communications*, vol. 126, pp. 28–37, 2018.

[24] C. Lin, Y. Zhou, F. Ma, J. Deng, L. Wang and G. Wu, "Minimizing charging delay for directional charging in wireless rechargeable sensor networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019, pp. 1819–1827.

[25] G. Jiang, S.-K. Lam, Y. Sun, L. Tu and J. Wu, "Joint charging tour planning and depot positioning for wireless sensor networks using mobile chargers," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2250–2266, 2017.

[26] Z. Lyu, Z. Wei, J. Pan *et al.*, "Periodic charging planning for a mobile wce in wireless rechargeable sensor networks based on hybrid pso and ga algorithm," *Applied Soft Computing*, vol. 75, pp. 388–403, 2019.

[27] Y. Ma, W. Liang and W. Xu, "Charging utility maximization in wireless rechargeable sensor networks by charging multiple sensors simultaneously," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1591–1604, 2018.

[28] L. He, Y. Gu, J. Pan and T. Zhu, "On-demand charging in wireless sensor networks: Theories and applications," in *2013 IEEE 10th international conference on mobile ad-hoc and sensor systems*, IEEE, 2013, pp. 28–36.

[29] C. Lin, Y. Sun, K. Wang, Z. Chen, B. Xu and G. Wu, "Double warning thresholds for preemptive charging scheduling in wireless rechargeable sensor networks," *Computer Networks*, vol. 148, pp. 72–87, 2019.

[30] A. Kaswan, A. Tomar and P. K. Jana, "An efficient scheduling scheme for mobile charger in on-demand wireless rechargeable sensor networks," *Journal of Network and Computer Applications*, vol. 114, pp. 123–134, 2018.

[31] X. Cao, W. Xu, X. Liu, J. Peng and T. Liu, "A deep reinforcement learning-based on-demand charging algorithm for wireless rechargeable sensor networks," *Ad Hoc Networks*, vol. 110, p. 102 278, 2021.

[32] I. Goodfellow, Y. Bengio and A. Courville, *6.5 back-propagation and other differentiation algorithms. deep learning*, 2016.

[33] Y. Bengio, *Learning deep architectures for AI*. Now Publishers Inc, 2009.

[34] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010.

[35] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[36] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[37] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[38] K. Itō, *Encyclopedic dictionary of mathematics*. MIT press, 1993, vol. 1.

[39] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[40] L. Lin, "Reinforcement learning for robots using neural networks. usa: Defense technical information center," DTIC Technical Report: ADA261434, Tech. Rep., 1993.

[41] S. F. G. M. T. AC, "Hagenbuchner m monfardini g the graph neural network model," *IEEE Trans. Neural Netw*, vol. 20, no. 1, p. 61, 2009.

[42] A. Menzli, *Graph Neural Network and Some of GNN Applications: Everything You Need to Know*, Dec. 2021. [Online]. Available: `neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications`.

[43] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*, PMLR, 2017, pp. 1263–1272.

[44] B. Sanchez-Lengeling, *A Gentle Introduction to Graph Neural Networks*, Sep. 2021. [Online]. Available: `https://distill.pub/2021/gnn-intro/`.

[45] W. Hamilton, Z. Ying and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[46] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *International workshop on artificial neural networks*, Springer, 1995, pp. 195–201.

[47] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[49] K. Yamaguchi, K. Sakamoto, T. Akabane and Y. Fujimoto, "A neural network for speaker-independent isolated word recognition.," in *ICSLP*, 1990.

[50] L. Xie, Y. Shi, Y. T. Hou and H. D. Sherali, "Making sensor networks immortal: An energy-renewal approach with wireless power transfer," *IEEE/ACM Transactions on networking*, vol. 20, no. 6, pp. 1748–1761, 2012.

[51] H. T. Friis, "A note on a simple transmission formula," *Proceedings of the IRE*, vol. 34, no. 5, pp. 254–256, 1946.

[52] K. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[53] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[54] H. Zhao, Z. Chen, H. Jiang, W. Jing, L. Sun and M. Feng, "Evaluation of three deep learning models for early crop classification using sentinel-1a imagery time series-a case study in zhanjiang, china," *Remote Sensing*, vol. 11, p. 2673, Nov. 2019. DOI: 10.3390/rs11222673.

[55] A. Paszke, S. Gross, F. Massa *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in.

[56] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

[57] W. J. Conover, *Practical nonparametric statistics*. john wiley & sons, 1999, vol. 350.

[58] G. Casella and R. L. Berger, "Statistical inference, cengage learning," *Kentucky*, 2001.