Delft University of Technology
Master of Science Thesis in Embedded Systems

Deep Reinforcement Learning for Rapid Communication Network Recovery with Multiple Repair Crews

Ingimundur Vilhjálmsson

TU Delft Delft University of Technology

Deep Reinforcement Learning for Rapid Communication Network Recovery with Multiple Repair Crews

Master of Science Thesis in Embedded Systems

Embedded and Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Ingimundur Vilhjálmsson
I.Vilhjalmsson@student.tudelft.nl
ingimundurvi@gmail.com

13 September, 2021

Author
Ingimundur Vilhjálmsson (I.Vilhjalmsson@student.tudelft.nl)
(ingimundurvi@gmail.com)
Title
Deep Reinforcement Learning for Rapid Communication Network Recovery with Multiple Repair Crews
MSc Presentation Date
27 September, 2021

Graduation Committee

Dr.ir. Fernando A. Kuipers	Delft University of Technology
Dr.ir. Johan Pouwelse	Delft University of Technology
Ir. Jorik Oostenbrink	Delft University of Technology

Abstract

Natural disasters can destroy communication network components, potentially leading to severe losses in connectivity. During those devastating events, network connectivity is crucial for rescue teams as well as anyone in need of assistance. Therefore, swift network restoration following a disaster is vital. However, post-disaster network recovery efforts have been proven to be too slow in the past.

Rapidly deployable networks (RDN) are communication networks that can be configured as a wireless mesh network and can be integrated into an existing communication network. As the name suggests, RDNs have a quick setup time and are highly transportable. The technologies behind RDNs for communication networks have received considerable advancements in recent times. Nonetheless, the deployment strategy of such a network remains open.

The existing solutions on rapid post-disaster network recovery are built in an inflexible way. First, each of them is designed around a specific problem. Making slight modifications to the problem greatly increases the complexity of the algorithm and can require major design changes to the system. Second, the proposed solutions are unable to adapt to unexpected circumstances, such as repair times taking longer than anticipated. We propose an online network recovery approach to solve these flexibility issues.

With the optimization objective of maximizing a network's weighted connectivity while minimizing the overall recovery process duration, we design a Deep Reinforcement Learning (DRL) system to produce optimal RDN deployment decisions. Experiments show our Deep Q-network (DQN) algorithm outperform greedy and naive approaches on any disaster scenario.

Preface

This report presents my thesis project for the Master of Science in Embedded Systems at the Delft University of Technology. During my time with this project, I gained a vast amount of knowledge and experience in the fields of communication networks and machine learning. My research in these fields has revealed to me their immense capabilities as well as their future objectives.

I would like to express my deepest gratitude to Professor Fernando Kuipers for giving me the opportunity to work on this thesis. To my daily supervisor Jorik Oostenbrink, thanks for all the guidance and detailed feedback. I highly appreciate the support you gave me during the challenging times. I also want to thank Professor Johan Pouwelse for being on my graduation committee.

To my friends at TU Delft, thanks for all the wonderful memories. To Jacob, Kaustubh, Teodor, Yordan, Naveen, Thomas, Jón, and Nikoleta, I am grateful for the times we spent together at the university and in life. To Alejandro, Rafa, and Jacob T., thanks for keeping the spirit alive in Michiel when the pandemic hit. So many special memories were made and I will cherish them forever. To Avinab, Barry, Catalina, Eveliina, Iñigo, and Loucas, your warmth made me feel at home during my stay in Breestraat.

To mom and dad, none of this would have been possible without you. It is hard to describe how grateful I am for your support and belief in me. To my brothers, thanks for encouraging and motivating me to go the distance in all my endeavors.

Ingimundur Vilhjálmsson

Delft, The Netherlands
19th September 2021

Contents

Chapter 1

Introduction

The increasing dependence that society has on communication networks has reached nearly every person around the world. This dependence has been ingrained into most aspects of our lives including work, obtaining news, entertainment, and keeping in touch with people. More importantly, we rely on communication networks for contacting first responders during emergency situations.

It is well known that events such as earthquakes, hurricanes, and other natural disasters can cause severe damage to communication networks. During those critical times, people may not be able to call for help until the network has been restored. Repair operations are, however, often extremely time consuming and can range from weeks to months [25]. Besides the economic impact, this amount of time is too long to endure for people that get trapped and are in need of food and water [5]. It is therefore essential to find an effective way to quickly recover a damaged communication network following a major disaster.

1.1 Rapidly Deployable Networks

From the need for fast network establishment and recovery emerged the concept of Rapidly Deployable Networks (RDN). The origin of RDNs goes back to 1998, when Evans et al. proposed the concept as a means for maintaining connectivity between troops traversing through unknown locations [7]. This concept has been the subject of many researches over the years, and still is to this day. Recently, much of the published work in the field of RDNs has specifically been on methods pertaining to the deployment of network components for post-disaster scenarios [17] [25].

An RDN is an adaptive and scalable network that can be set up as a wireless mesh network (WMN) or as an extension to an existing network. Typical applications of RDNs make use of transportable network components such as Cells On Wheels (COW) and the Movable and Deployable Resource Unit (MDRU). As these mobile cell site units offer the same services as existing permanent network components, they can be utilized as replacements for damaged components. In addition, integrating these units into an existing network is a swift and seamless process, and far less time-consuming than performing network repairs.

1.2 Existing Solutions

In the field of rapid post-disaster network recovery, numerous approaches with various optimization objectives have been proposed. For instance, some studies focus on maximizing throughput over time, while others consider reachability or other metrics. Still, most of the existing approaches share a similar repair resource allocation mechanism to achieve their recovery goal. That is, these approaches tackle the network recovery problem in a so called progressive manner.

Since there is a limited amount of repair resources available at a time, it is not possible to repair the whole network at once. Thus, progressive network recovery strategies slice the total repair time into stages. In each stage, an operator allocates the available repair resources to a number of components. The problem is then to find the repair order that fulfills the optimization objective.

There exist many variations in the application of the existing progressive network recovery approaches. In other words, the existing solutions are typically designed around a specific issue, in addition to their optimization objective. Some approaches consider, for example, partial repair, uncertainty, or data centers. However, the existing approaches usually focus on a just a single variant. As a consequence, these solutions suffer from the lack of flexibility.

First, many of the proposed solutions assume full knowledge on aspects such as network damage, demand, and repair times. In real-life, however, operators are not guaranteed to have full knowledge on numerous parameters. Furthermore, the parameters considered in the optimization objective of an approach may turn out to be different in the real world. This raises questions on the practicality of many of these studies.

Second, the proposed approaches are designed to consider only a limited set of factors relevant to network recovery. Some of these factors include travel time, repair time, uncertainty, and more. Although incorporating more factors into the model could lead to better results, such changes would often require major or complete changes to the proposed algorithms.

Last, the stage-like manner of these solutions prevents repair resources from being immediately assigned to their next repair site, leading to inefficient time management. That is, even though repair resources can become available before the next stage, they can only get allocated at the beginning of each stage.

1.3 A Reinforcement Learning Based Approach

To address the main issue of inflexibility, we present an online and adaptable approach that is also capable of considering multiple factors relevant to network recovery. Our approach involves training a reinforcement learning (RL) agent to learn how to recover a network optimally. The use of RL for our problem is appealing because it (1) allows for one to easily make modifications to the underlying model, and (2) can adapt to unforeseen conditions.

In our network environment model, we consider damage uncertainty, component repair times, repair crew travel speeds, crew locations, and the remaining times of each crew's assignment. Hence, these parameters are taken into account in the agent's decision on where to assign the repair crews. As we will explain later, the agent learns to optimally recover the network through simulations of interactions with the environment. Furthermore, we design our RL

based approach to be adaptable to unexpected situations.

1.4 Contributions

The motivation and objective of this thesis is to bring new ideas into the field of rapid post-disaster network recovery. In our opinion, the existing solutions are inflexible with respect to design and to the uncertainties present in real-life. It turns out that adding or changing a parameter in these algorithms requires their design to be completely modified. That is the main reason why there exist so many papers on the topic in the first place.

With this thesis, our contributions to the area of rapid post-disaster network recovery are as follows:

- A simulation environment that incorporates the essential parameters of network recovery: time, distances, locations, repair crews, different network component types and their states. These parameters enable for interactions one would need in real-life network recovery scenarios.
- Two RL algorithms that are trained to make optimal repair order of a network subject to uncertain damage extent. Further, these algorithms are flexible to parameter variations. They are also flexible in the way that changing or adding parameters to the model is simple.
- We demonstrate our approach outperforming naive baseline algorithms in all of our experiments.

Chapter 2

Related Work

This chapter provides an insight into the related work. As the subject of RDNs is broad, the chapter is organized into sections based on each item involved in RDNs.

2.1 Mobile Cell Sites

While this thesis focuses on the deployment strategy of RDNs, it is important to understand the hardware used in recovering a network. Since we assume that only network nodes are prone to failures, we consider existing work on mobile cell sites. Although mobile cell sites are commonly utilized in temporary network expansion situations where there is increased traffic, their function in the context of this thesis is to serve as replacements to damaged nodes.

2.1.1 Movable and Deployable ICT Resource Units

Following the immense network destruction caused by the Great East Japan earthquake in 2011, substantial research has been put into the technology of Movable and Deployable ICT Resource Units (MDRU). The MDRU first emerged in 2013 and was developed by a collaboration between Nippon Telegraph and Telephone (NTT), Tohoku University, and Fujitsu. In their initial article, Sakano et al. present the architecture of the device and demonstrate that it is applicable to large scale network failure scenarios [27].

In essence, an MDRU is a transportable ICT node in the form of a container that houses communications equipment, servers, storage, power supplies, and cooling systems [27][30]. The deployment scheme of an MDRU revolves around setting up and connecting access points (AP) to gateways that are wirelessly linked to the MDRU. Afterwards, the APs provide voice services and Wi-Fi Internet access to the public. The authors note that the total MDRU installation process takes around 40-140 minutes [28]. Furthermore, the MDRU is shown to satisfy six key requirements from a user's perspective: transportability, quick installation, carrier-free usability, provide essential services, high capacity, and large coverage [29]. It is worth noting that backhaul is achieved via existing optical fibers, but can also be done with satellite links. In addition, the MDRU can run for five days without outside interaction.

2.1.2 Cell on Wheels and Cell on Light Trucks

A Cell on Wheels (COW) is a fully fledged mobile communication system that is transported by vehicles, usually as a trailer [38]. Typically, COWs are deployed to provide extra capacity at public events where existing infrastructure will likely not be able to handle the network traffic. However, COWs have also been used in post-disaster situations [37]. Similar to the MDRU, the backhaul to the network for COWs can be done through existing cables or via wireless standards, including satellites. The main difference between COWs and MDRUs lies in their setup procedure. COWs are simpler to setup since they are single systems that users directly connect to, while MDRUs require the placement of APs at locations around them for users to connect to. In fact, the COW manufacturer Solaris Technologies Services notes that the operational setup time of its products ranges from 10 to 40 minutes, depending on the model [31].

There is no significant technical difference between COWs and Cells on Light Trucks (COLT), except that the cellular system is fixed on the COLT. However, there are trade-offs between these two designs, primarily in the aspect of transportability. A COLT can be more easily relocated as it is physically built to the vehicle, whereas the COW system is on a trailer and thus can be detached from the vehicle. The benefit of the COW design is that it allows the deployment crew to stay on the move after setting it up. Nevertheless, both of these technologies have been tested and proven to work in extreme weather conditions, from hot and humid weather to knee-deep snow levels, and even in areas struck by hurricanes [20].

2.2 Recovery Strategies

The task of rapidly deploying repair resources for network recovery has been the focus of numerous researches. While various strategies have been considered, it is clear that the style of progressive recovery approaches is the most popular.

2.2.1 Progressive Network Recovery Approaches

Since repair resources are limited in amount, progressive network recovery strategies slice the recovery process into stages. In each stage, the available resources are assessed such that allocations can be made according to a recovery objective.

In the first-ever progressive recovery based approach, Wang et al. consider the maximization of network capacity in terms of the total possible flow between each source and destination node pairs as their objective [41]. Assuming that only links need repair, the solution looks to find the best possible subset of links to repair during each stage. The effectiveness of a recovery solution is evaluated with the proposed metric of the Accumulative Weighted Total (AWT) flow over all stages. To achieve this, methods based on Mixed Integer Linear Programming (MILP) were designed. Proposed are three variants that differ in the importance placed on stages during the recovery. That is, whether to strive for gains in the early stages or in later stages. There are significant drawbacks to this approach overall. First, full knowledge of the damages to the components of the network is required. Second, it assumes exactly how much resources it costs to repair a link, making it inflexible to the deviations present in real life.

Tootaghaj et al. propose a more realistic approach for progressive network

recovery by incorporating the aspect of partial state knowledge [35]. This approach uses a probabilistic estimate for node and link failures. That is, the failure probability of elements with an unknown status get approximated using various methods. These nodes are designated to a so called *gray* area, while nodes known to work belong to a *green* area and those not working are placed in a *red* area. The objective is then to minimize the Expected Recovery Cost (ERC), which corresponds to the cost of repairing nodes and links, while satisfying demand under a specific capacity constraint. Nodes are allowed to monitor or discover the status of other components within a certain hop count, hence gaining better information on the *gray* area. The authors note, however, that the impact of the hop count limit varies greatly with the network topology size. In addition to this issue not being clarified further, it is hard to deduce the effectiveness of this approach for large networks from the experiments that are made. Nevertheless, as discussed later, our work borrows their method for approximating component failure probabilities.

In the context of network virtualization, Pourvali et al. developed progressive recovery approaches that aim for repair resource allocations to minimize disruption [23][24]. These approaches include a uniform placement of resources between damaged components, random placements of resources, and resource prioritization on node degree. As these approaches operate under the assumption that repair resources are limited and cannot be endlessly reallocated, the problem is then of partial network recovery. Furthermore, the components used in this work can provide a partial or degraded level of service despite being damaged. Experiments showed the uniform placement scheme producing fast recovery in the initial stages. But, the node degree algorithms lead to a higher level of recovery, after the resources have been fully utilized. These approaches lack flexibility to deal with unexpected events that occur in real life, e.g. repairs taking longer than anticipated.

Ferdousi et al. explore how progressive recoveries of networks coupled with datacenters can be optimized [8]. Because network recovery and datacenter recovery are interdependent, the authors look into maximizing the cumulative weighted content reachability to users at each stage. This optimization is formulated as an Integer Linear Program (ILP). The algorithm designed to solve this problem evaluates potential reachability gains for repairing certain components and chooses the order that yields the highest gains earlier. In addition to this, the algorithm is resource-aware (limited number of resource reallocations) and considers the partial recovery of components. That is, damaged components can still operate at partial capacity.

Ciavarella et al. present a progressive approach for minimizing the duration of the network recovery process [4]. Because a disaster can impact a network such that the damage extent is unknown, it is necessary for operators to perform monitoring and inspections to see which components were affected. Hence, the authors propose a damage assessment program with that purpose along with restoring components; the overall process covers monitor placement, network probing and repair performed in a progressive manner. Comparing its performance to their other, non-progressive solution [2], the authors show substantial improvements in recovery time and network flow throughout the process. However, optimizing recovery time is not the main objective of their non-progressive approach.

Zhao et al. propose a progressive solution for finding the optimal recovery

schedule for Cyber Physical Systems (CPS) in post-disaster situations [45]. In these networks, there is interdependency between components such that the order of their repair matters. Additionally, components can differ in the repair resources required. The proposed solution, while considering these two factors, has the objective of maximizing the system utility over all repaired components through the overall repair process. Similar to other works, this proposed approach uses an ILP algorithm. Experiments show that their ILP based solution becomes too computationally intensive for cases where over 200 components fail.

All the mentioned works were specially designed around a single issue and are not easily modifiable to fit slightly different problems. Furthermore, they all optimize for specific parameter values and are thus not flexible to unexpected situations. The requirement for full knowledge on the damage extent is prevalent in the existing approaches. Lastly, scalability is a key aspect that is not addressed widely. In fact, only Zhao et al. directly discussed this aspect after discovering flaws in their work when recovering from large-scale failures.

2.3 Machine Learning

Recently there have emerged proposals of approaches utilizing machine learning for optimizing the network recovery process. These proposals show that machine learning, specifically reinforcement learning, is viable for graph based problems.

Within the scope of infrastructure networks, Sun and Zhang demonstrate that deep reinforcement learning can be utilized to find effective recovery solutions [32]. The authors propose a framework for Agent Based Modeling (ABM) and a Deep Q-Network (DQN). As an infrastructure network is composed of different yet interdependent components, such as transportation and power systems, the DQN can and will learn about these relationships during training. The focus of the proposed system is to find the action that improves the network functionality the most for a given state. Simulation experiments were carried out on a large network and showed the learner converging towards an optimal solution. However, the authors only compare the performance of their system to naive random algorithms.

Ishigaki et al. also explore the recovery of dependent network layers, but with regards to Virtual Network Functions (VNF) [12]. Similar to many of the existing network recovery strategies, the authors propose a progressive approach that aims to maximize accumulated utility through the recovery effort; the same objective as in [41], except with the interdependence of nodes in different layers taken into account. Furthermore, the employment of an RL algorithm sets this recovery approach apart from the other progressive solutions. From their results, the authors posit that RL coupled with a heuristic algorithm makes for superior means to solve optimization problems over the ones utilizing only either. However, the authors do not demonstrate a capability to adapt to unexpected situations in their approach. Also, it is unclear how computationally expensive the utility and demand functions in their heuristic algorithm are. This information is critical for understanding the scalability of their approach.

Chapter 3

Problem Definition

This chapter defines the problem of rapidly deploying repair crews for post-disaster communication network recovery. Specifically, we focus on recovering networks damaged by devastating earthquakes. In such events, nodes are vulnerable to damages, while optical fiber links are robust [27]. To recover the communication network, an operator sends repair crews to damaged nodes in an online manner.

Problem 1. Minimizing a Network's 1-WATTR: In our problem, we consider an undirected graph $G = (V, E)$ composed of nodes $v \in V$ and links $e \in E$ to represent a real-world communication network. We also consider repair crews $r \in R$ that travel to failed nodes and recover them. The problem is then to make repair crew deployments that minimize $G's$ 1-WATTR over the recovery process. In other words, our optimization objective is to minimize the area underneath the curve of the network's 1-WATTR over time.

The metric Weighted Average Two-Terminal Reliability (WATTR) is the weighted ratio of connected node pairs and the total amount of node pairs in a graph. WATTR can be used to quantify the proportion of the population connected to the network [22]. Thus, we let each node have its own scalar weight $v_w \in \mathbb{R}_{\geq 0}$. The WATTR of a graph is formally defined as follows:

Definition 1. *Weighted Average Two-Terminal Reliability (WATTR)*

Let

$$I(v, x) = \begin{cases} 1 & \text{if node } v \text{ is connected to node } x \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Then

$$WATTR := \frac{1}{W} \sum_{v \in V} \sum_{x \in V - \{v\}} v_w \cdot x_w \cdot I(v, x) \tag{3.2}$$

where

$$W := \sum_{v \in V} \sum_{x \in V - \{v\}} v_w \cdot x_w \tag{3.3}$$

Since we consider an undirected graph in our problem, the computation of the WATTR can be optimized by iterating through each connected component

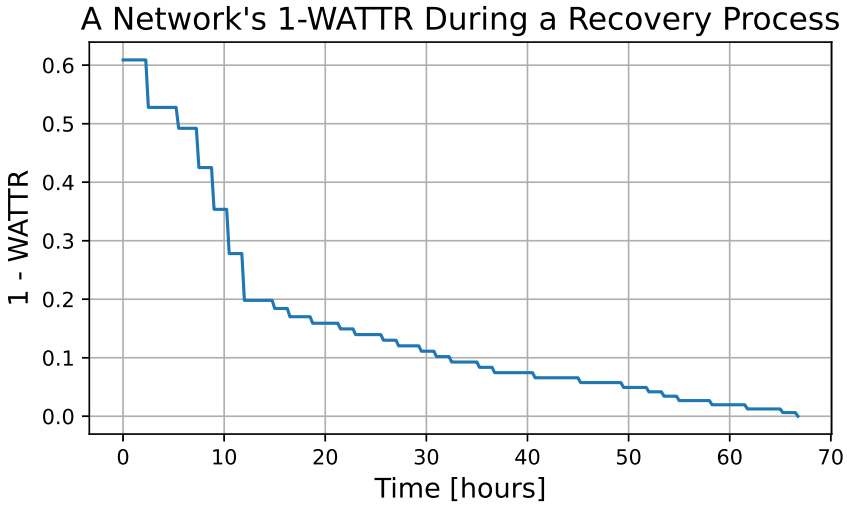


Figure 3.1: **Example of a network's 1-WATTR during post-disaster recovery operations.**

$c \in C$ rather than each node pair:

$$WATTR = \frac{1}{W} \sum_{c \in C} \sum_{v_i \in c} v_{w,i} * \left(\left(\sum_{v \in c} v_w \right) - v_{w,i} \right) \tag{3.4}$$

where v_i is a node in the connected component c, $v_{w,i}$ is its weight, and W is obtained from the graph prior to any damage:

$$W = \sum_{v_i \in V} v_{w,i} * \left(\left(\sum_{v \in V} v_w \right) - v_{w,i} \right) \tag{3.5}$$

Thus, a 1-WATTR of 0 signifies that all node pairs are connected, while a 1-WATTR of 1 indicates that no node pairs are connected.

An example of a 1-WATTR curve over the course of a network's recovery process is illustrated in Figure 3.1.

Motivation for minimizing 1-WATTR: It might seem simpler to maximize the area underneath the WATTR curve, rather than to minimize the area underneath the 1-WATTR curve. Although they mean the same thing, the former case is harder to implement. Since we aim to optimize the area under the curve from when recovery begins until it finishes, maximizing the WATTR area would lead to inefficient repair orders that extend the total recovery time.

Flexible modelling: As explained in the next chapter, the communication network and repair crews are modelled with specific properties. This is done to enable the simulation of network recovery processes and the utilization of an RL agent. Moreover, this type of approach is flexible in terms of modelling. That is, an RL agent can learn to optimally deploy crews to nodes no matter how they are modelled.

Chapter 4

Approach

Our approach to solve the problem defined in chapter 3 involves two systems: a network recovery simulator and an RL agent. Although these systems are distinct, they are dependent on each other. In essence, our agent's purpose is to assign crews to nodes, while the simulator's role is to simulate their effects. Since the agent assumes the role of a network operator, it has access to the state of the crews and the communication network. This information is maintained and provided by the simulator. Specifically, the state of the communication network is managed using the NetworkX Python library [21].

In this chapter, we first present the design of the simulator and its operating procedure, followed by details on our RL agent and how we train it to make intelligent crew-to-node assignments.

4.1 Network Recovery Simulator

Because of the interactions between repair crews and the communication network in our problem, we must model their relevant characteristics. We model the nodes $v \in V$ of our communication network as follows:

- Type $v_t \in \{\text{Base Station, Core Node}\}$: a node is either a base station or a core node. In the context of our work, core nodes represent switching centers, while base stations provide users with connections to the network.
- Weight $v_w \in \mathbb{R}_{\geq 0}$: a base station is weighted to reflect the population it serves. Core nodes have a weight of zero.
- Location $v_l = (\text{Latitude, Longitude})$: every node has specific GPS coordinates corresponding to its location.
- Status $v_s \in [0,\ 1] \vee 2$: the probability of the node being damaged, where values between 0 and 1 indicate the uncertainty of a node's functionality, and $v_s = 2$ indicates that the node is reserved for repair.

Besides the nodes, we model repair crews in the following way:

- $r_l = (\text{Latitude, Longitude})$: the location of a crew.
- $r_s \in \{Free,\ Busy\}$: a crew is either free to be assigned to a node or is busy working on a node.

- $r_t \in \mathbb{N}_0$: the time a crew is busy for. This value is computed based on travel distance and node installation times, which we explain later.

- $r_v \in \mathbb{N}^+$: the speed at which a repair crew is traveling at.

- r_q: repair crews have a list for their queued node assignments and their corresponding time to complete.

Despite repair crews not performing actual repairs to damaged nodes, we will still refer to them as repair crews in the context of network repair. Also, we assume crews have an unlimited supply of both fuel and COW-like node replacement units.

Since these models undergo a change during network recovery, we design a simulator to represent that process. Our simulator is designed as a discrete-event simulator that constantly loops over the same instructions, which facilitate environment state updates, until full network recovery is attained. Furthermore, this is done through fixed-increment time progression, in which an iteration in the loop reflects a certain time duration in the real-world.

There are tradeoffs to consider when choosing the real-life time duration that an iteration should represent. Letting the simulator progress over small time increments will yield a better time management of crew deployments at the cost of more iterations. For instance, let us say that at time $t = 0$ a crew is busy for 25 minutes, meaning that at $t = 25$ minutes it becomes free and can be deployed to a new node. If the simulator progresses over 30 minute intervals, then the crew would get deployed at $t = 30$ minutes, after idling for 5 minutes. But, if the simulator progresses at 1 minute intervals instead, then the crew would be deployed immediately after becoming free at $t = 25$ minutes. However, the former case, with 30 minute time progression per iteration, costs substantially fewer iterations compared to the latter case.

In order to determine how much real-life time an iteration in our simulator should represent, we must consider the actual time scale which the repair crews operate at. Because repair crews use trucks for transporting COWs, we can safely assume that traveling between nodes will take more than just a few minutes. Moreover, as mentioned in chapter 2, the installation time of COWs ranges from 10 to 40 minutes. We can therefore deduce that a reasonable time progression per iteration is in the order of several minutes. For our experiments, we let the simulator progress on 15 minute increments.

4.1.1 Simulator Control Flow

To clarify how the simulator operates, we delve into each of its stages and the flow between them. But before simulating a network recovery process, we first generate an initial state of the environment. That is, a disaster causes the destruction of nodes, and the initial states of our repair crews get chosen. Our simulator then proceeds to continuously iterate over the instructions facilitating the network recovery until all nodes have been recovered.

Figure 4.1 illustrates the stages and the flow within an iteration of the simulator, which are further explained as follows:

1. Every iteration begins by retrieving the set of free repair crews.

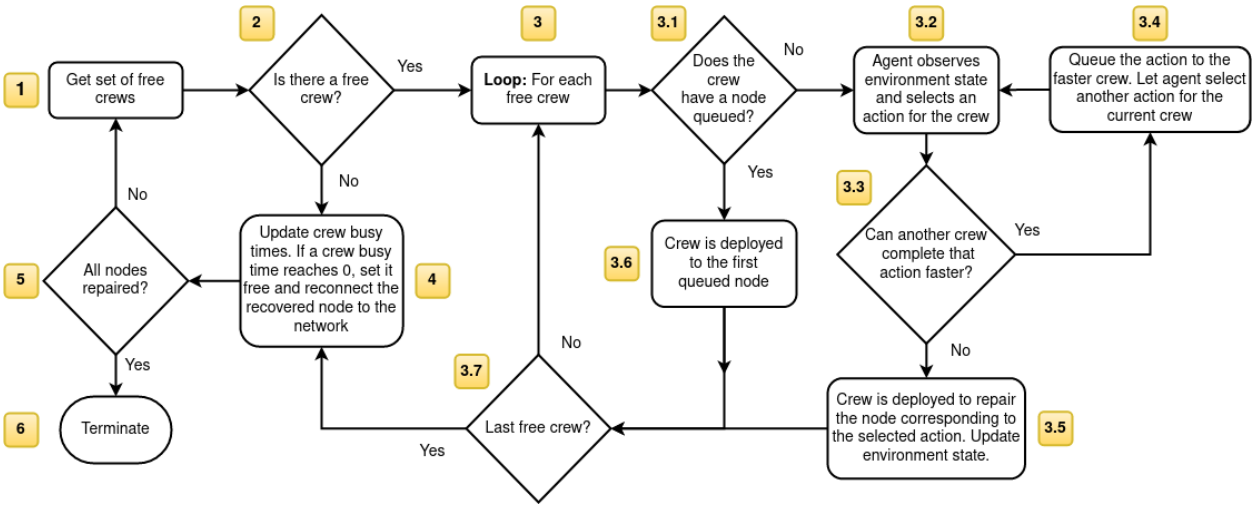


Figure 4.1: **Stages within an iteration**

2. Then, the simulator checks if any crews are free. If there is a free crew, the program enters stage 3. Otherwise, it transitions to stage 4 to prepare for the next iteration by updating each crew's work status.

3. The simulator iterates over each free crew.

   3.1. If the free crew already has a node in its queue, the simulator transitions to stage 3.6. If not, the simulator proceeds to stage 3.2.

   3.2. The environment state is given to the agent. With that information, the agent selects a node to assign the free crew to.

   3.3. For each repair crew, the time it would take for it to recover the selected node is computed. We calculate the time for a crew r to fix a node v as follows:

$$t_{fix} = t_{travel} + t_{install} + r_t + r_{tq} \tag{4.1}$$

   where $t_{travel} = haversine(r_l, v_l)/r_v$ is the time it takes for a crew to travel to the node. In the case where a repair crew has nodes in its queue, its travel time is computed with the location of its last queued node. Note, we use the haversine formula [43] to compute distances between two points on Earth. To roughly reflect actual setup times of COWs we let $t_{install} = 45\ minutes$ (3 iterations). We denote r_{tq} as the total time it takes a crew to recover all the nodes in its queue. Then, if the lowest t_{fix} belongs to the iteration's free crew, the simulator proceeds to stage 3.5, otherwise it enters stage 3.4.

   3.4. The queue of the fastest crew gets appended the node and the time it will take to recover that node ($t_{travel} + t_{install}$). Because the node has been queued for repair, its state is set to $v_s = 2$. The simulator transitions back to stage 3.2, where the agent selects a different node for the free crew. Further explanation on the queuing process is given in section 4.1.2.

3.5. The free crew is deployed to the selected node. The simulator updates the crew's r_t to equal t_{fix}, in addition to setting $r_l = v_l$, and $r_s = Busy$. Lastly, the state of node is set to $v_s = 2$, as it is now under repair.

3.6. The free crew is deployed to the first queued node, changing its r_l to that location and its r_t to the corresponding r_q time. That node is then removed from the crew's queue.

3.7. The simulator checks whether all of the free crews have been iterated over. If so, it proceeds to stage 4.

4. Crews are updated by decrementing their r_t. Crews whose r_t reaches zero have their r_s declared "Free" and their repaired nodes get reconnected to the network. Hence, the state of those nodes is set to $v_s = 0$.

5. The simulator checks whether all nodes are reachable. If all nodes are connected, then network recovery is complete and the program proceeds to stage 6. Otherwise, the whole iterative process is repeated from stage 1 onwards.

6. Simulation terminates.

4.1.2 Queuing Process

Without a crew coordination mechanism, the agent can make inefficient crew-to-node assignments. For example, consider a scenario of one free crew, one busy crew, and only one damaged node. The agent must then assign the free crew to the damaged node, regardless of whether the busy crew could finish recovering its current node and the damaged node faster. For this reason, we design a queuing process that does two things: (1) prevent inefficient crew-to-node assignments, and (2) queue node assignments for crews.

After the agent decides which node a free crew gets assigned to, its decision gets evaluated by our queuing process. As described in stage in Equation 4.1, we compute the times for each crew to finish their current and queued assignments plus the time to recover the chosen node. If the free crew is the quickest to recover the node it will get assigned to it. Otherwise, the node is queued to the fastest crew.

Each crew's queue, r_q, is formatted as a list of tuples that contains the queued node and the time it will take the crew to recover it. For example, a crew's queue could be as follows:

$$r_q = [(17,\ 9),\ (2,\ 7)]$$

Here, after the crew finishes its current job, it will go to node 17 and recover it in 9 iterations. At the time it gets deployed to node 17, this tuple will be removed from the queue. Completing the recovery of node 17 will lead the crew to node 2, for which it will take 7 iterations to recover.

Overall, our simulator enables for the simulation and evaluation of network recovery operations. However, to solve our problem of finding a good recovery order, we must implement an algorithm that trains our agent to make optimal crew-to-node assignments. Let us now discuss the approach we make for implementing the agent.

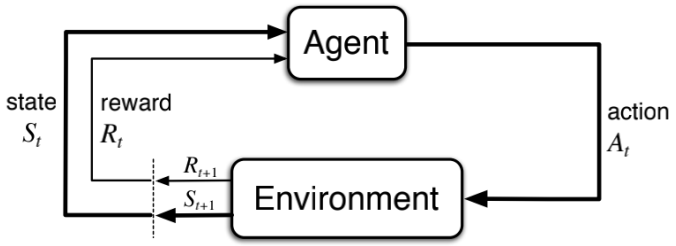


Figure 4.2: **Interaction between the agent and its environment. The environment receives the agent's action, and in turn gives the agent a reward and the updated environment state.**

4.2 Modeling a Markov Decision Process

A Markov Decision Process (MDP) is a discrete-time stochastic control process that models the decision-making of an agent within an environment. The agent's objective is to interact with the environment such that it accumulates maximal reward. It does so by making actions based on the environment state, for which it receives a reward and moves to a new state. Formally, an MDP is defined as the 4-tuple $(S,\, A_s,\, P_a,\, R_{t+1})$, where:

- S is the set of states s.
- A_s is a set of actions available for the agent to take from state s.
- $P_a(s, s') = \mathbb{P}\left[s_{t+1} = s' \mid s_t = s\right]$, is the probability of transitioning to state s' after taking action a from state s at time t.
- R_{t+1} denotes the reward that the agent receives after transitioning from state s to state s' at time $t+1$.

The goal in an MDP is to find an optimal policy, π^*, that maximizes the reward accumulation. Where a policy, π, is a mapping from each state $s \in S$ and action $a \in A_s$ to the probability $\pi(a|s)$ of taking action a in state s [33]. In essence, a policy specifies what action is to be taken from a given state. Furthermore, a state's value under a policy π is denoted as $v_\pi(s)$. This value equals the expected reward accumulation, discounted by $0 \leq \gamma \leq 1$, going from a state s, at time t, and following π:

$$v_\pi(s) = E_\pi \left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \,\middle|\, s_t = s \right] \tag{4.2}$$

Besides the state-value function described in Equation 4.2, an action-value function $q_\pi(s, a)$ is the expected reward return for starting in state s, taking a, and then following policy π:

$$q_\pi(s, a) = E_\pi \left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \,\middle|\, s_t = s, a_t = a \right] \tag{4.3}$$

Equation 4.3 is important, as later we introduce the Bellman equation that allows for the learning of a good policy.

In our approach, we implement the elements making up an MDP in the following ways:

4.2.1 State space

We formulate a state in the environment as a list containing each node's status and each repair crew's location and busy time. An environment state could for example be:

$$s_i = [v_s(0) = 0,\ v_s(1) = 1,\ v_s(2) = 0.9,\ ...,\ v_s(n-1) = 0,\ r_l(0) = 5,\ r_t(0) = 8,\ ...]$$

In this example, nodes 0 and $n-1$ are known to be working, while node 1 is known to be damaged, and node 2 is not known to work and is assumed to have a probability of 0.9 to be damaged. Also, repair crew 0 is at node 5, with 8 iterations until that node recovers.

Although the environment state is relayed to the agent, the simulator has complete knowledge of the damage extent. That is, the simulator knows which nodes are damaged with full certainty. Doing so allows for the functional but disconnected nodes to be reconnected to the graph immediately when their neighbor is recovered. For example, when a core node is destroyed, the functionalities of its connected base stations will be uncertain to the agent. Then, when a crew recovers the core node, all of the functional base stations connected to it will also be reconnected to the network and with their status changed to $v_s = 0$.

4.2.2 Action space

The action space represents the set of actions available to the agent in a given state. This is simply the set of nodes which the agent can assign repair crews to. In our implementation, the action space for a state s could for example be:

$$A_s = \{7,\ 9,\ 13\}$$

where nodes 7, 9, and 13 are damaged or unreachable and thus available to the agent for making crew deployments to.

4.2.3 State transition probability

After the agent selects an action, probabilities of to which state it leads to are computed. However, as discussed later, we employ a model-free RL algorithm for training our agent, hence the use of P_a is not needed.

4.2.4 Reward function

A fundamental factor for training our agent is how to reward its actions. As defined in chapter 3, our problem is to minimize the area under the communication network's 1-WATTR curve during the recovery operation. For this reason, we let the environment reward the agent with a value equal to the estimated 1-WATTR area between the time of a crew-to-node deployment and the recovery of the action's corresponding node. This means that the network's 1-WATTR

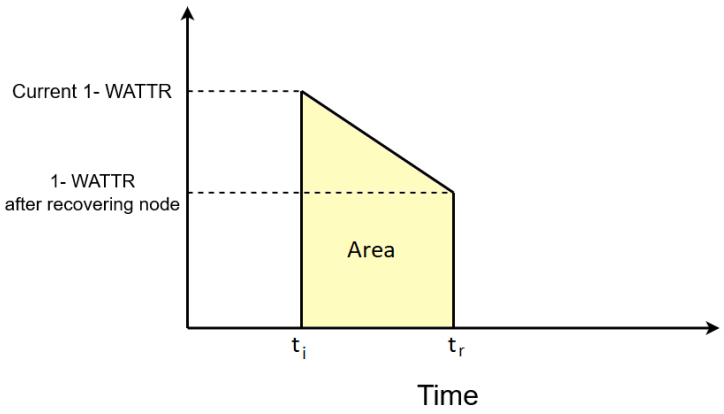


Figure 4.3: **Example of 1-WATTR area produced by an action leading to the recovery of a node.**

must be computed twice: when a crew gets assigned a node and after that node is recovered.

We first compute the network's 1-WATTR at the time a crew is deployed to a node, t_i. Then we estimate the network's 1-WATTR for the time the node is recovered, t_r. For computing the 1-WATTR after recovering a node, all other nodes that recover between t_i and t_r are included in the network for the computation. However, if a core node will be recovered within that time frame, then its base stations will also be recovered but based on their v_s (failure probability). We generate a random number on the uniform distribution [0,1] for each of those base station nodes. If the randomly generated number is greater than the node's failure probability, then we include that node for the 1-WATTR estimation. This approximation is necessary because the agent does not always know to what environment state its actions lead to, and thus neither the network's exact future 1-WATTR value.

With t_i, t_r, and the network's 1-WATTR at those time points, we draw a line between those points as illustrated in Figure 4.3. The area underneath this line is then computed with basic geometry equations and used as the agent's reward for its action.

We choose this method for rewarding the agent because the agent's total reward accumulation will relate to the 1-WATTR area of the whole recovery process. Therefore, by tuning the agent to accumulate as little reward as possible, it will take actions that lead to the minimization of the network's 1-WATTR.

It might seem more intuitive for the objective to be to maximize the area under the WATTR connectivity curve. However, doing so will lead to issues where the agent selects actions that drag out the recovery process. Due to the agent being rewarded the area that its actions produce, it will accumulate maximal reward by taking actions that recover the network slowly. Conversely, if the objective were to minimize the area under the WATTR connectivity curve, then the agent could prefer making crew deployments that do not provide substantial connectivity gains soon after a disaster. Consider, for instance, a disaster destroying all nodes of a network that consists of numerous base stations and a single core node. Recovering all base stations first and the core node last would

produce an WATTR connectivity curve with zero area.

4.3 Reinforcement Learning

Reinforcement Learning (RL), an area of machine learning, involves an agent interacting with an environment, modeled by an MDP, to learn which action to take in a given situation [33]. In RL systems, the environment is made up of states in which there are a set of actions available for the agent to take. To learn which action to take, the agent receives a reward based on the quality of the action it takes. Therefore, when the agent returns to a state it has visited before, it will remember which action yielded the best reward.

In each state, the agent can either take the best action it knows of, called exploiting, or explore alternative paths by taking a random action. Exploring is crucial for learning, since it can lead to discoveries of better and worse states. The exploration rate of the agent is denoted with the parameter $0 \leq \epsilon \leq 1$.

Training the agent to learn which actions are good is done through running numerous episodes of the simulation scenario. In finite state-action problems, this can be done until a convergence to the optimal action-value function, q_π^*, is achieved.

Numerous RL algorithms exist for training an RL agent, but for our work we only consider Q-learning and Deep Q-learning.

4.3.1 Q-learning

Q-learning is a simple RL algorithm where the quality of state-action pairs, known as Q-values, are stored in a table, called the Q-table. Each Q-value in the table is initialized with an arbitrary fixed value prior to the training process; we choose zero for the initial Q-values in our work.

During training, at time t, the agent takes an action a_t in the state s_t for which it subsequently receives a reward R_{a_t}. The value of this state-action combination is then computed, using a Bellman equation, and is then updated in the Q-table:

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha \cdot \Big(R_{a_t} + \gamma \cdot \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \Big) \tag{4.4}$$

where α is the learning rate, a coefficient on how important new information is, and γ is the discount factor indicating the significance of future rewards. To ensure that old information does not get overridden, we let the learning rate be $\alpha = 0.1$. Since the agent seeks to maximize the overall accumulated reward, we let the discount rate be $\gamma = 0.99$. The exploration rate in our implementation begins at $\epsilon = 1$, and after each simulation episode it is reduced by 0.0002 until $\epsilon = 0.001$.

We format the environment states and actions for the Q-table the same way as described in 4.2, as $s_i = [v_s(0), v_s(1), ..., r_l(0), r_t(0), ...]$. While classic Q-learning examples construct and initialize the Q-table for all of the possible states in the environment, we build our Q-table on the go. This is due to the fact that, in most cases, our environment state space will be too large to fit in a table on the average PC desktop. Additionally, to prevent incredibly large Q-tables, we discretize v_s to be $v_s \in \{0, 1, 2, 3, 4, 5\}$, where:

- $v_s = 0$: node is known to be functional.
- $v_s = 1$: the node's failure probability is on the interval (0, 0.25).
- $v_s = 2$: the node's failure probability is on the interval [0.25, 0.5).
- $v_s = 3$: the node's failure probability is on the interval [0.5, 0.75).
- $v_s = 4$: the node's failure probability is on the interval [0.75, 1].
- $v_s = 5$: means that the node is under repair.

Not only does this prevent the Q-table from becoming enormous, but having fewer states will also reduce the training time. This is slightly different from how we modelled the node status property in chapter 3, which is for our Deep Q-learning implementation.

Properly rewarding actions made by the agent is critical for it to correctly learn the optimal actions in given states. Since our objective is to find the solution with the lowest 1-WATTR area during a recovery operation, we reward the agent according to the scheme presented in section 4.2. However, this makes the agent's goal be to accumulate the smallest possible reward. Therefore, we modify the Bellman equation to consider the minimum Q-value of the next state when updating the Q-value of the current state-action combination:

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha \cdot \Big(R_{a_t} + \gamma \cdot \min_a Q(s_{t+1}, a) - Q(s_t, a_t) \Big) \tag{4.5}$$

This means that smaller Q-values are better. Thus, when our agent decides to exploit in a state, it selects the action corresponding to the lowest Q-value. A reward for that action is then produced, followed by having Equation 4.5 update the Q-value for the state-action combination.

Unfortunately, the Q-learning algorithm comes with some major flaws. Its main issue is that the agent is slow to converge to a good solution, especially when γ is near one [1]. Furthermore, with large-scale networks and a high amount of repair crews, the state-space will be tremendous. This causes the amount of training episodes to be prohibitively many. Thus it is clear that the scalability of Q-learning systems is inefficient.

4.3.2 Deep Q-learning

Because of the training challenges in Q-learning, we shift our approach to utilize Deep Q-learning instead. This RL algorithm, introduced by Mnih et al. [18], uses a deep neural network to approximate the Q-function, instead of a Q-table. The authors coined the term Deep Q-Network (DQN) for this neural network, as it returns Q-values for a given environment state.

Before diving into DQNs, we must first understand the basic building block of neural networks, the artificial neuron. An artificial neuron is designed to mimic the functioning of a biological neuron. We illustrate its structure in Figure 4.4. A neuron's role entails receiving inputs over weighted connections to process into an output signal. That is, the neuron sums up the received weighted input values and passes the result to an activation function, ϕ, which in turn produces the neuron's scalar output, y. The activation function is used to introduce non-linearities to the neuron's output, allowing for better learning

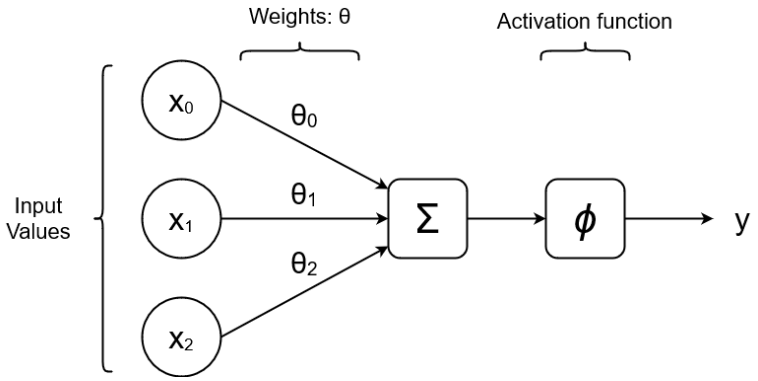


Figure 4.4: **Artificial Neuron Model**

in non-linear problems. Various activation functions exist, but for our work we incorporate the ReLU function which is defined as:

$$\phi(z) = max(0, z) \tag{4.6}$$

Now, by organizing neurons into layers, with neurons getting connected to other neurons in adjacent layers, we can construct a DQN.

As shown in Figure 4.5, a DQN consists of an input layer, hidden layers, and an output layer. First, the input layer gets fed the current environment state. The input is subsequently propagated to the neurons of the first hidden layer. After receiving their inputs, the neurons of the first hidden layer perform the computations as described before. Results from each neuron are then propagated to the next hidden layer where the computational and propagation processes repeat. Finally, the last hidden layer feeds the output layer its values, which in turn produces the Q-values of every action for the inputted state. An example diagram of a DQN for an environment made up of three nodes and one repair crew is shown in Figure 4.5.

It is known that single DQN systems are vulnerable to unstable learning. Therefore, our implementation incorporates the two key ingredients proposed in [19] that help overcome this issue: a target DQN and experience replay. A target DQN is identical to the online DQN, except its weight parameters, θ^{target}, update to be $\theta^{target} = \theta^{online}$ every τ steps. Essentially, the target network is used to measure the loss or error in relation to the online network. This helps prevent the online network from drifting away into a worse policy.

We train the DQN by adjusting the weights of the network, θ^{online}, such that the loss is minimized. The loss function in our implementation calculates the mean squared error of the target Q-values and the predicted Q-values:

$$L = [(r + \gamma \cdot \min_{a} Q(s_{t+1},\, a;\, \theta^{target}) - Q(s_t,\, a;\, \theta^{online})]^2 \tag{4.7}$$

To reach minimal loss, we compute and backpropagate the partial derivatives of the loss function with respect to each of the weights. We can think of the loss function as an irregular surface, where the partial derivative (the slope) aids us in the search for the lowest point on it. For updating the weights, θ^{online}, based on the computed gradient, we use the Adam optimizer [6]. Adam extends

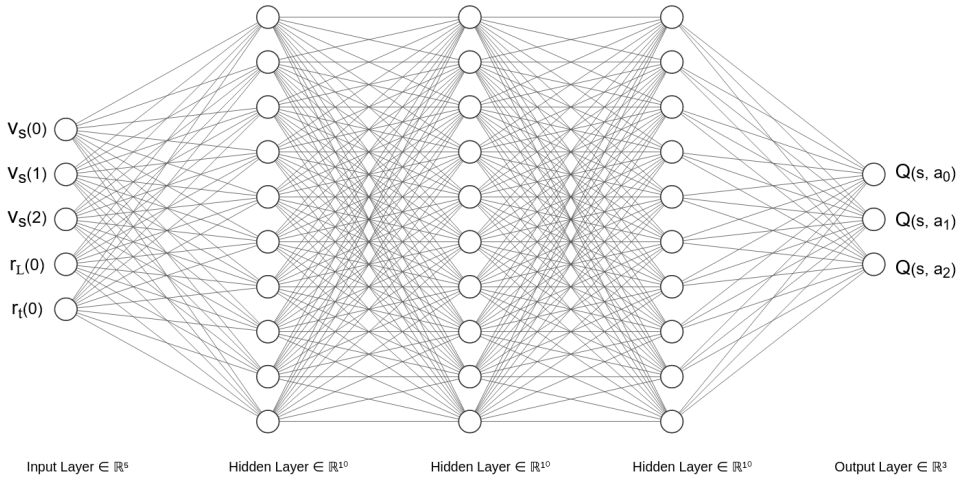


Figure 4.5: **Deep Q-Network [15]**

and improves on the classic stochastic gradient descent method by incorporating two additional elements for tuning the weights towards the minimum loss: momentum and adaptive learning rate.

Momentum simply allows for larger steps on the loss surface to be made when the direction of movement stays the same, and smaller steps are made when the direction changes. This is take into account when determining the learning rate α. More details about the Adam algorithm can be found in [6].

Since the Adam optimizer requires the training data to be independent and identically distributed, we include a memory of previously made transitions to avoid sampling correlated experiences. Thus, during training we randomly sample a batch of transitions from the memory to run through the neural network. This technique is referred to as experience replay. Note that after the memory is filled, we let new transitions replace the oldest ones.

Figure 4.6 shows how we connect all of these pieces together to create our DQN system. Furthermore, we made this implementation using the open source machine learning framework PyTorch [14]. There are several hyperparameters in our system that need manual tuning.

- γ: the discount factor, which we set to 0.8 based on empirical results.
- ϵ: the agent's exploration rate is initialized to 1, and is decremented by ϵ_{dec} during training.
- ϵ_{dec}: how much ϵ decays by after every transition. We set this value to 4e-5.
- ϵ_{min}: the minimum exploration rate of the agent is set to 0.001.
- α: the agent's learning rate, set to 0.0001.
- Memory size: the amount of transitions that can be stored in the memory. We found that a memory size of 10,000 works well.

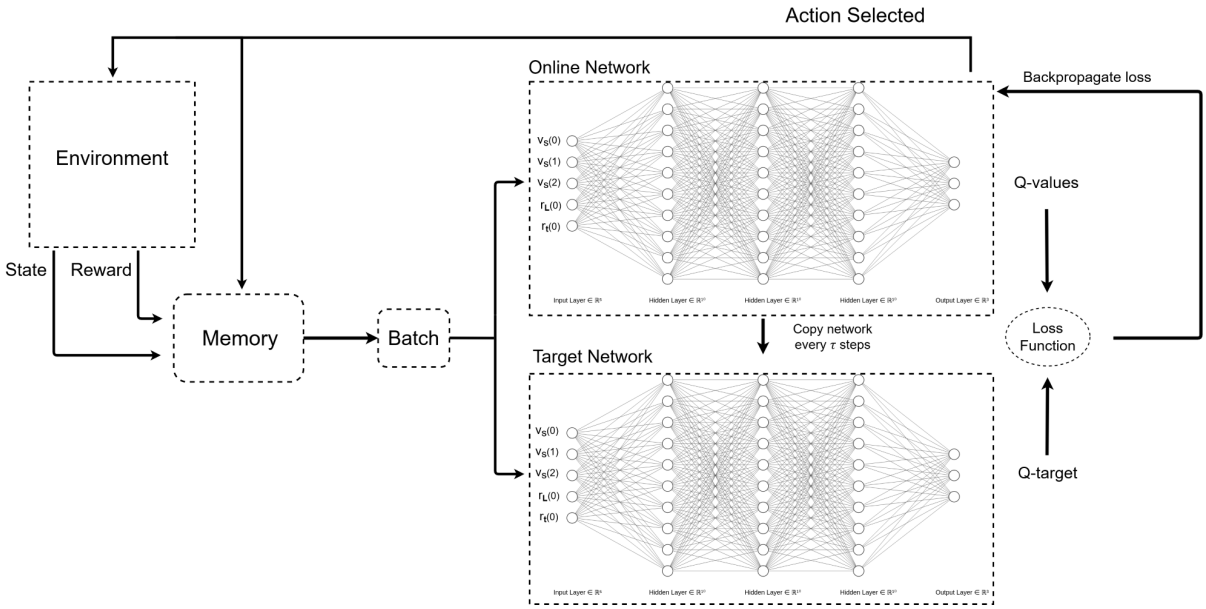


Figure 4.6: **Deep Q-learning Architecture**

- Batch size: the amount of transitions sampled. We set the batch size to be 64, which is fairly typical for DQNs.
- τ: the rate at which the target network updates to match the online network. We set this rate to such that for every 1,000 transitions, the target network gets updated.

For our experiments, we will use a DQN with three hidden layers. Each of the hidden layers is composed of a number of neurons twice the number of nodes plus repair crews. We found this configuration to work the best out of several other ones.

Chapter 5

Disaster Model and Graph Generator

In this chapter, we first present how we simulate the destructive effects that an earthquake has on our communication network. We then introduce our algorithm for generating the graphs used in our experiments.

5.1 Earthquakes

Earthquakes occur frequently around the world and are known to be able to cause great damage to communication networks [36]. While an earthquake's power is characterized by its magnitude, the earthquake's intensity is not uniformly spread over the area around it. The release of energy caused by an earthquake creates seismic waves that propagate radially from the hypocenter [11]. The attenuation of these waves are dependent on several variables, including sedimentary composition through which the waves travel. Because of this, the intensity or shaking is not exactly the same at equal distances from the epicenter; making it challenging to generate a realistic earthquake for simulating.

In our experiments, we make use of data from historic earthquakes provided by the United States Geological Survey (USGS) [40]. Such earthquake data is stored in an `XML` file that formats a high resolution grid of locations over the impacted region with their corresponding intensity values. The intensity values provided in these files correlate with the Modified Mercalli Intensity scale (MMI) [39]. The MMI scale translates intensity values into physical damages, described in table 5.1, and with it we can make estimates on the failure probability of every node in a network following the earthquake.

Using the rough indications on the damage probability to structures, we compose and implement a function to estimate the node failure probability for a given MMI intensity level to use in our experiments. As shown in table 5.1, damage does not scale linearly over the intensity levels, and only at intensity `VI` does the damage start to become increasingly likely. Therefore, we assume that the damage probability scales exponentially with intensity. For these reasons

Intensity	Damage Description
I	Minimal shaking, not felt by most people.
II	Noticeable to a few people, more so to those on upper floors.
III	Felt by some people. Comparable to the shaking caused by a passing truck.
IV	Felt by many, light objects disturbed. Low chance of any damage.
V	Shaking is felt by most. Still, low chance of any damage.
VI	Felt by all. Heavy furniture moved. Some probability of damage.
VII	Minimal damage to well-designed structures. Considerable damage to poorly built structures.
VIII	Slight damage to well-designed structures. Great damage to poorly built structures.
IX	Major damage to well-designed structures. Poorly built buildings might collapse, and many become shifted off foundations.
X	Almost all structures destroyed, save for some well-built wooden structures.

Table 5.1: **Modified Mercalli Scale [39]**

we estimate the node the failure probability using the following function:

$$Node\ Failure\ Probability = exp\left(\frac{Intensity - 10}{2.75}\right) \tag{5.1}$$

In other words, the node failure probability computed in Equation 5.1 is based on the intensity at the location of the node. A visualization of this is provided in Figure 5.1. Using this equation, along with an earthquake intensity file, we can simulate the effects of the earthquake by computing each node's failure probability and destroying them accordingly. Furthermore, we let each node's status, v_s, equal its computed failure probability if it is unreachable. Note, the simulated earthquake is for demonstration purposes and has not scientific basis.

5.2 Graph Generator

Since topologies of real-world communication networks are not publicly available, we developed a simple method to build similarly structured networks. To create such networks, we supply the generator with a `CSV` file of locations to build a network around. More specifically, the rows contain the following information for each city:

$$id,\ City\ Name,\ Population,\ Latitude,\ Longitude$$

where the *id* is the row number in the file, city can be any string, population can be any positive number, latitude and longitude must be the decimal degrees of the city's coordinates.

We choose specific cities to host our network's core nodes. Then, each core node is connected to the two core nodes closest to it. Furthermore, to increase robustness, we select a radius around each core node such that they connect to all other core nodes within that distance.

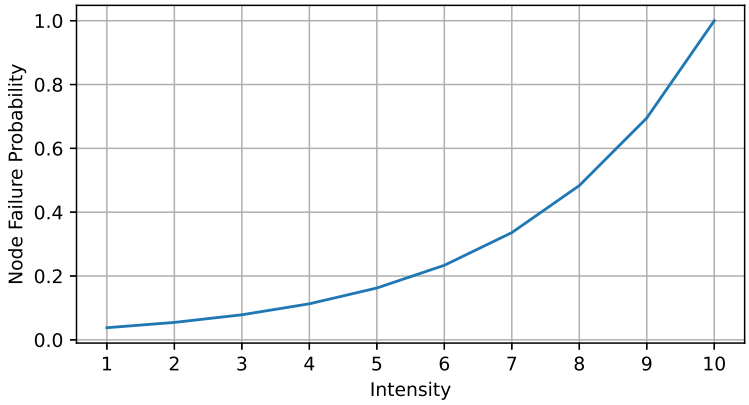


Figure 5.1: **Node Failure Probability Function**

For every city there is at least one base station at the location corresponding to the coordinates given in the `CSV` file. The amount of base stations per city can be determined in two ways: in relation to the city's population or randomly.

Our population-based scheme involves generating a base station per 100,000 inhabitants of a city. This design choice is made to prevent the generation of networks too large to experiment with. In the other method, each city gets an amount of base stations randomly sampled from a chosen uniform distribution. For example, let us say that a city can have at least two base stations and at most five base stations. Then the generator would generate a random number of base stations on that interval for each city.

After determining the number of base stations in each city, they are then assigned locations around the city's coordinates in a hexagonal manner with a radius of 5 km. Note, we set a limit of 19 base stations per city to keep things simple.

Assigning weights to base stations can also be done in two ways: in proportion to the city's population or randomly.

- Base stations of each city are assigned a weight equal to the city's population divided by the number of base stations located in it.
- The user chooses two positive numbers. Each base station weight is then randomly sampled from a uniform distribution on that range.

Lastly, each base station is linked to the nearest core node, and the generated graph gets stored in a `GML` file.

Chapter 6

Experiments

6.1 Experimental Setup

For our experiments, we use Southern California as the location of our simulated disasters and communication networks. We choose this location as it is regularly struck by devastating earthquakes. Specifically, we use USGS' MMI intensity data on the 1994 Northridge earthquake for our simulation experiments.

The basic characteristics of the communication networks used in our experiments are described in table 6.1. While the weights of the nodes in networks 1 and 2 were made to reflect the actual population of their city, the weights in network 3 were randomly chosen. Figure 6.1 shows the networks overlaid on a map of Southern California.

Network	Amount of Core Nodes	Amount of Base Stations
1	6	57
2	8	115
3	26	61

Table 6.1: **Communication Networks' Characteristics**

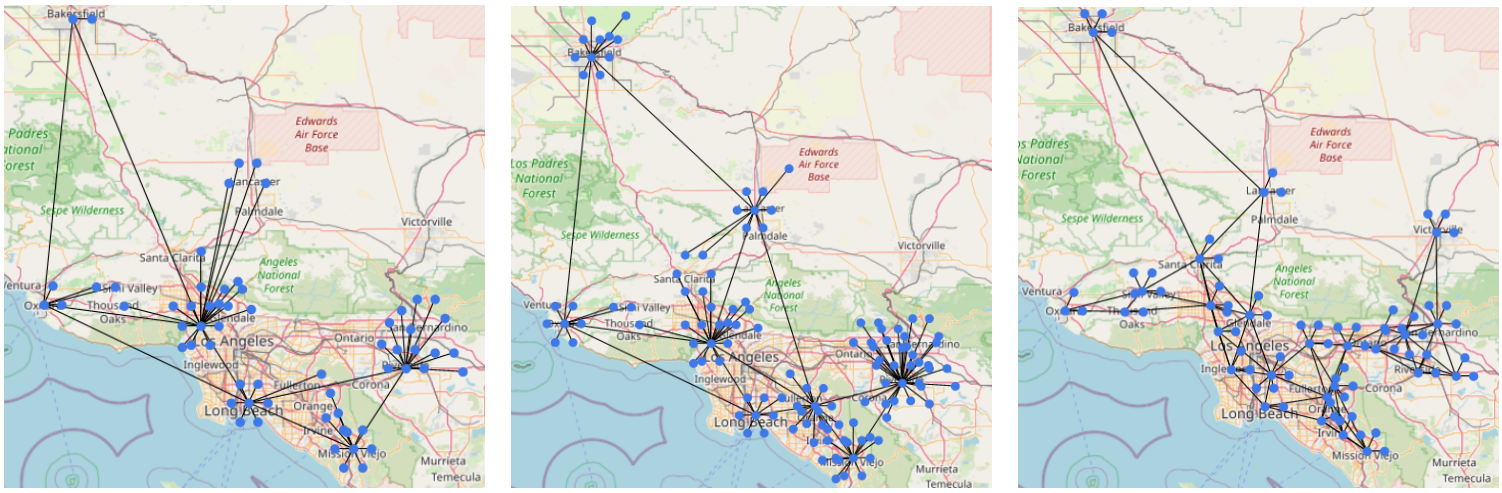
The system which we use to carry out our experiments on has the following specifications described in table 6.2. This system runs on the Xubuntu 20.04 Linux operating system.

6.1.1 Approaches Compared

We compare our RL algorithms against the following alternative approaches:

- **Weighted approach:** repair priority is on nodes with the greatest weight. In this approach, the weights of the core nodes equal the combined weight of their connected base stations.

- **Greedy approach:** repair priority is on nodes that produce the smallest immediate reward. Before a crew is deployed, the rewards of all potential node assignments are computed. The agent then makes the crew-to-node assignment with the smallest reward.



(a) Network 1. (b) Network 2. (c) Network 3.

Figure 6.1: **Networks used in our experiments.**

Component	Model or Description
CPU	AMD Ryzen 5 5600X
GPU	Nvidia GTX 960
RAM	16GB

Table 6.2: **System specifications.**

6.2 Experiments

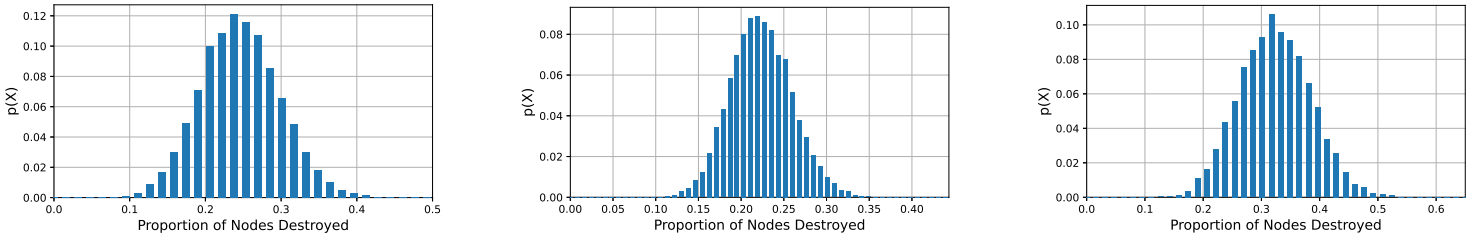
6.2.1 Training and Testing Procedures

To evaluate our RL algorithms, we conduct a training phase followed by a testing phase. Since we want our agent to learn to recover a network from the Northridge earthquake, we train it on the earthquake's potential destructive outcomes. That is, the node destruction for each training episode is generated probabilistically with Equation 5.1 and the USGS' Northridge intensity data. Shown in Figure 6.2, the destructiveness of the scenarios used in training appear to follow a Gaussian distribution.

In order to show that our approach is flexible and can make good repair decisions for any situation, we must test it on disaster scenarios that were not trained on. To properly generate test disasters, the node failure probability function is modified such that disasters are either weaker or stronger. We found that the following functions consistently generate destructiveness that are around two standard deviations from the mean of the training disasters' destructiveness:

$$Node\ Failure\ Probability = exp\left(\frac{Intensity - 10}{1.75}\right) \tag{6.1}$$

$$Node\ Failure\ Probability = exp\left(\frac{Intensity - 10}{4}\right) \tag{6.2}$$



(a) Network 1. (b) Network 2. (c) Network 3.

Figure 6.2: **PMF's of the destructiveness of the generated Northridge earthquake disaster outcomes used for training.**

6.2.2 Experimental Objectives

Our main experimental objective is to learn about the performance of our system in terms of training time and the quality of the solution it produces. In addition, to get the most out of our system, we need to determine the optimal time slice for our simulator to progress on. Thus, we perform experiments that address the following four questions:

1. What is an optimal time slice duration for an iteration in our simulator for training the agent on?

2. How does Deep Q-learning fare compared to Q-learning, the Greedy algorithm, and the Weighted algorithm in terms of the area of their produced recovery schedule?

3. Is there a difference in the training times of Q-learning and Deep Q-learning?

4. Is it possible for the DQN to train on scenarios and parameters different from the ones used in the testing phase and still obtain good results?

6.2.3 Choosing an Optimal Time Slice

Before looking into the performance differences between the approaches, we must first understand what time slice duration for an iteration in our simulator is optimal. For this we consider two factors: the training time and the area produced on different time slices. We do this because we want a system with a minimal training time that also converges to good solutions.

It is our expectation that smaller time slices produce better recovery solutions because it reduces the idling of repair crews, which we discussed previously. However, as this increases the amount of iterations in the simulation, we expect the training time to increase. For bigger time slices, we expect shorter training times but network recovery processes with greater areas.

First, let us examine the Deep Q-learning training times, for 10,000 episodes on network 1 and three repair crews, with whole number time slices on the interval $[1, 20]$ minutes. The outcome we receive, shown in Figure 6.3, confirms

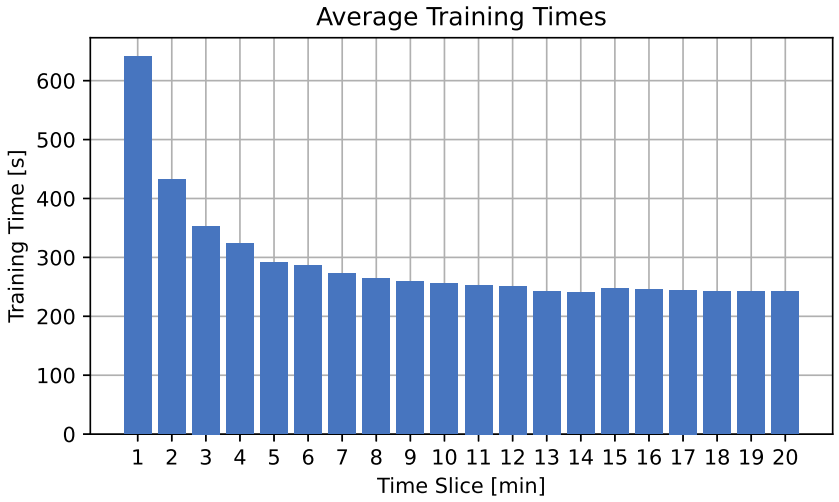


Figure 6.3: **The training times for each time slice.**

that the training time is lower for larger time slices. However, we observe diminishing returns for time slices over 10 minutes in length.

Along with the training times for each time slice, we tested the Deep Q-learning approach on a randomly generated disaster. Our results, illustrated with the red bars in Figure 6.4, show a linear growth in 1-WATTR area with increasing time slices. However, we discovered that the node recovery order was rather similar regardless of the time slice used. Therefore, it appears that training on big time slices yields a similar recovery order as when training on small time slices. To confirm this, we train for each time slice as before but make the test phase operate on a 1-minute time slice. The results we obtain are plotted, with blue bars, in Figure 6.4.

It is clear, from Figure 6.4, that training on any time slice leads to nearly the same recovery order. That is, training on any time slice and conducting the test phase on a 1-minute time slice produces a solution with an area close to as having trained on a 1-minute time slice. Therefore, the choice of what time slice to use should mostly be based on its training time. For our remaining experiments, we configure our system to train on 15-minute time slices and test one 1-minute time slices.

6.2.4 Approach Performance Comparisons

Recovering Network 1

Let us begin by analyzing the performance of the different approaches under ideal conditions, meaning that the only uncertainty lies in the damage extent. Hence, we train and test our RL approaches with the repair crews traveling at a speed of 100 km/h and node replacements taking 45 minutes. Throughout the training and testing phases, the different initial locations of the repair crews do not change. Furthermore, we repeat this experiment to gain insight on whether the RL approaches lead to the same results.

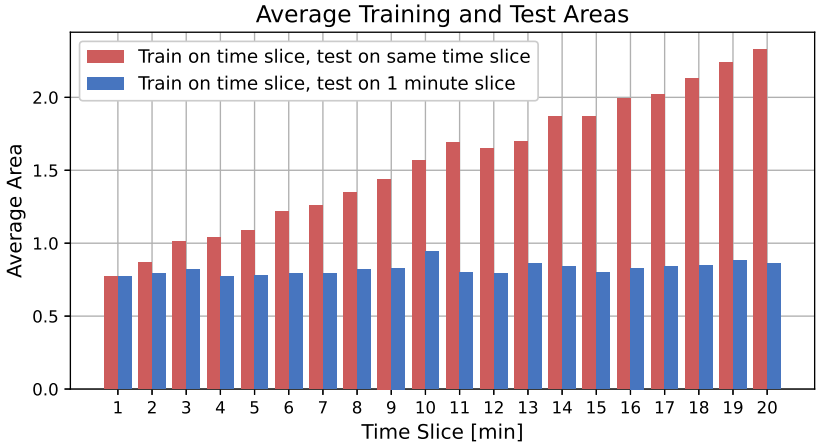


Figure 6.4: **Areas produced by each time slice.**

We train with probabilistically generated disasters for each of the episodes, according to Equation 5.1. To be clear, we train the RL approaches on 5,000 episodes in this experiment. Afterwards, testing is done on a set of 10 disasters that were generated through Equations 6.1 and 6.2. Training and testing for both one and three repair crew setups twice, we obtain the results shown in figure 6.5.

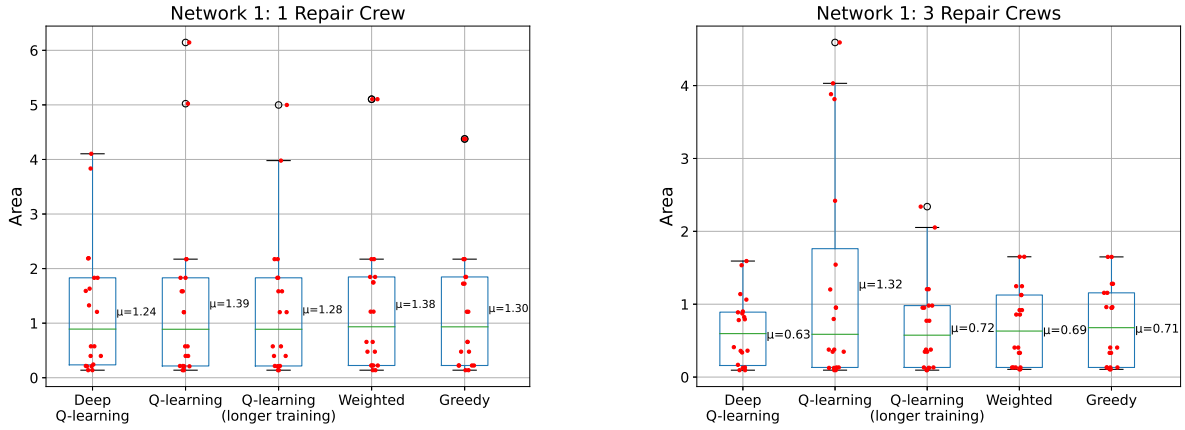
In Figure 6.5(a), it appears that the RL approaches converge toward solutions for recovering network 1 with a single repair crew. This is due to the fact that there are multiple pairs of red dots of equal area in the plot. Furthermore, both RL approaches perform slightly better on average compared to the greedy algorithm.

Adding two more repair crews gives the results shown in Figure 6.5(b). The key takeaway from this plot is that the Q-learning approach performs, on average, only 5.3% better compared to with just a single crew. It can also be seen that the areas of the disaster recovery tests on the Q-learning approach are dispersed. This performance degradation is due to insufficient amount of training on the larger state-space of this setup. Training the Q-learning approach for longer (50,000 episodes) confirms this, as its results exhibit signs of convergence in addition to substantial overall improvements. But, even after training for longer, the Q-learning approach performs 9.5% worse than the Deep Q-learning approach and 4.3% worse than the baseline Weighted algorithm on average.

Regarding the training times, we found that the Q-learning system, tuned to train on 50,000 episodes, was approximately 2x longer than the Deep Q-learning system, which trained for roughly 310 seconds on average. Since it is clear that the Deep Q-learning system beats Q-learning by all measures, we will not continue experimenting on the Q-learning system.

Scaling Up to Network 2

Let us now look into the performances of the approaches on a larger network. Using network 2, we repeat the training and testing procedures, except for



(a) Recovery areas of each approach with 1 repair crew.

(b) Recovery areas of each approach with 3 repair crews.

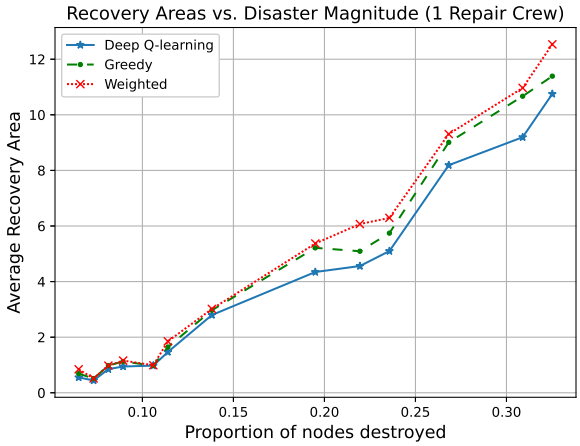
Figure 6.5: **Performance of each approach on recovering network 1. Each red dot represents the 1-WATTR area from recovering a test disaster.**

our test set we now use 16 disasters that vary considerably in destructiveness. Furthermore, we let our approach train on 20,000 episodes, which for our system took about 1,300 seconds to complete. After training and testing four separate times, the results of applying the approaches to recover the larger network are shown in Figure 6.6.

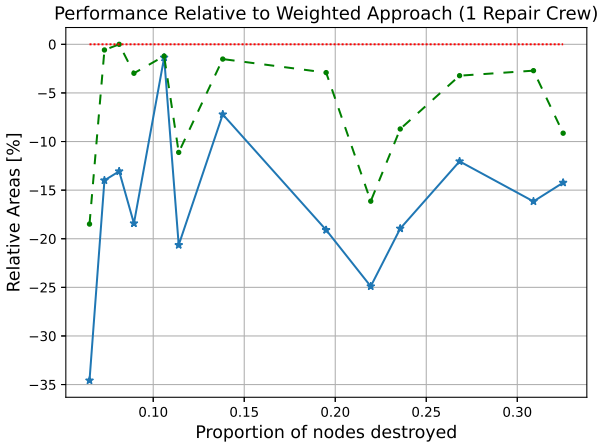
Under both the one and three repair crew setups, it is apparent that the recovery areas grow linearly with the proportion of nodes destroyed for all approaches. This finding is shown in Figures 6.6(a) and 6.6(c). In regards to the relative performance difference between the approaches, we can see that it is less predictable for smaller disasters. Whereas for larger disasters, the Deep Q-learning performs around 10-25% and 7-17.5% better than the Weighted algorithm for one and three repair crew setups, respectively. Compared to the Greedy algorithm, Deep Q-learning performs on around 5-15% and 3-9% better for one and three repair crew setups respectively.

Interestingly, there is a noticeable relationship between the performances of the Greedy and Deep Q-learning approaches, as can be seen in Figures 6.6(b) and 6.6(d). This relationship stems from their similar crew to node assignment strategies. However, the Greedy algorithm does not take the future into account, as it only considers the reward from the time of making crew deployments.

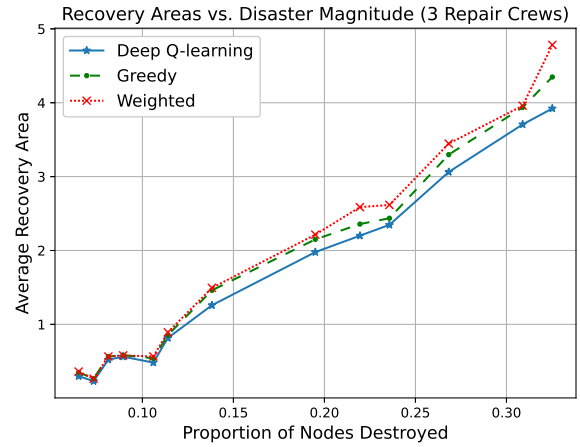
Further investigation revealed that the Greedy approach takes the same actions as the Deep Q-learning approach during the early stages. This is due to the way the node weights are distributed in the network. While the node weights are somewhat even in our network, there is a set of nodes with weights over 6x greater than the average node. Moreover, this set of nodes makes up for over 80% of the network's WATTR and is clustered at the location of the disaster (Los Angeles). Hence, the aspect of optimizing travel is not highlighted until when the remaining damaged nodes share similar weights. This is clearly



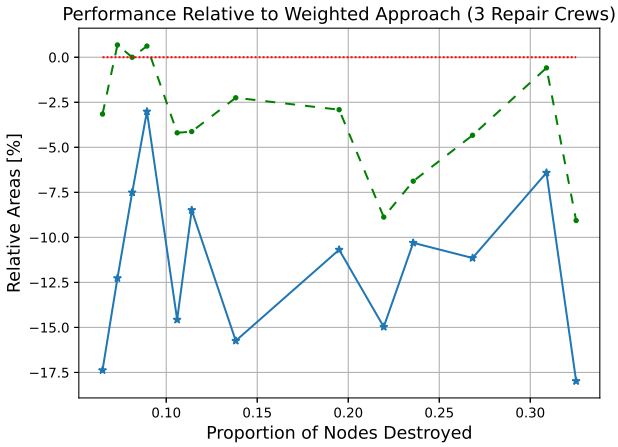
(a) Average recovery areas of each approach with 1 repair crew.



(b) Performances relative to Weighted approach with 1 repair crew.



(c) Average recovery areas of each approach with 3 repair crews.



(d) Performances relative to Weighted approach with 3 repair crews.

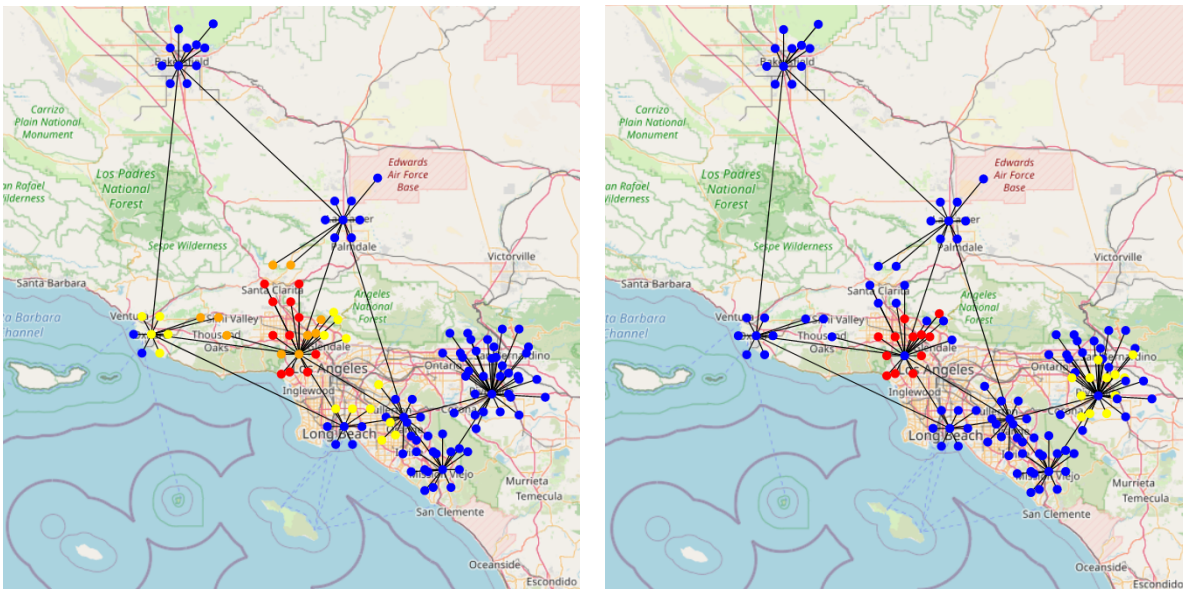
Figure 6.6: **Performances of each approach on recovering network 2 for both one and three repair crew setups.**

illustrated in Figures 6.7(a) and 6.7(b), where the nodes with by far the greatest weights are all located in close proximity to each other and with a high probability of failure.

Disaster Impact to Network Recovery Performance Relationship

Knowing about the striking weight differences between nodes, it is clear that disasters mainly affecting the Los Angeles area allow for major WATTR restoration without significant travel costs.

Looking at our test disasters, one of them destroys the core node in Los Angeles and three of its 12 major base stations, along with 13 other nodes within



(a) Node failure probabilities: red 40-65%, orange 30-39%, yellow 20-29%, blue 0-19%

(b) Node weights: red >600, orange 200-600, yellow 100-199, blue 0-99

Figure 6.7: **Node failure probabilities and weights in network 2.**

a 75 km radius around it. For this test case with a single repair crew, our Deep Q-learning approach performed only 5% better than the other two approaches. With three repair crews, our approach performed around 9% better. Plotting the network's 1-WATTR during the recovery process made by the different approaches, for both 1 and 3 repair crew setups, is shown in Figure 6.8.

Recovering from this disaster with a single repair crew allows for uncertainty to be solved through restoring the damaged core nodes first. It turns out, all approaches begin by recovering the Los Angeles core node since its recovery produces by far the best reward. After it is recovered, the agent now knows with full certainty which of the major base stations are damaged, and proceeds to recover them.

More significantly, with three repair crews, the Deep Q-learning agent has learned about the risk of deploying a crew to an unreachable base station while its corresponding core node is being restored. Furthermore, this also means that the agent learns to coordinate the repair crews in the whole recovery process. This is unlike the Greedy approach, which prefers deploying a crew to an uncertain node in an attempt to gain as much connectivity as quickly as possible.

Let us now examine whether the Deep Q-learning approach performs better, relative to the other algorithms, when recovering from disasters mainly affecting light-weight nodes. We expect Deep Q-learning to perform far better in those disasters since the repair crew travel aspect plays a bigger role.

In our set of test disasters, there are a few that mostly affect light-weight nodes. Let us examine test disaster 15, which causes the destruction of two core nodes and 27 base stations. Of the heavy-weight nodes in the Los Angeles

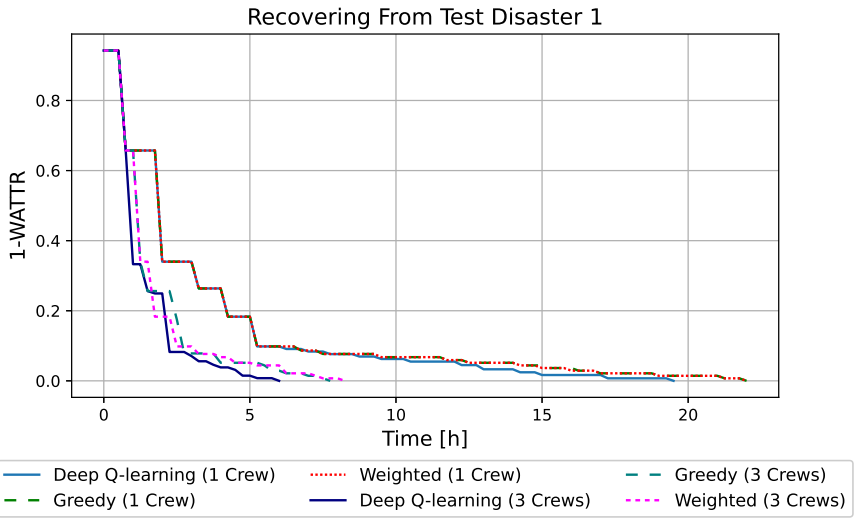


Figure 6.8: **The network's 1-WATTR curves under different approaches for recovering network 2 from test disaster 1.**

area, only four base stations are destroyed, while their corresponding core node survives. Using a single repair crew to recover the network from this disaster, our Deep Q-learning approach performed 12% better than the Greedy approach and 19% better than the Weighted approach. However, with three repair crews the Deep Q-learning approach performed only 4% better than the Greedy approach and 10% better than the Weighted approach. Again, the approaches begin by recovering the core nodes first to gain full knowledge on the damage extent. Figure 6.9 shows the network's 1-WATTR during the recovery under the different approaches and repair crew setups.

These results show that Deep Q-learning performs better, relative to the other approaches, when node damages are more spread out and across nodes with similar weights.

In the test disasters that do not destroy any core nodes, where the damage extent is fully known, the approaches performed similarly well. Of course, the crew coordination capability of the Deep Q-learning approach gave it an edge.

Varying the Travel Speed and Node Replacement Times

Although we have shown that the Deep Q-learning agent is able to train and test on fixed travel speeds and node replacement times, these two parameters can vary in the real-world. Therefore, let us investigate the possibility of training on varied parameter values.

For our training phase, each episode gets these parameters values randomly chosen from uniform distributions of travel speeds and node replacement times. We let the speed at which the repair crews can travel range between 60 and 120 km/h. Furthermore, we let the time it takes for a repair crew to replace a damaged node range from 30 to 120 minutes. We define these distributions as:

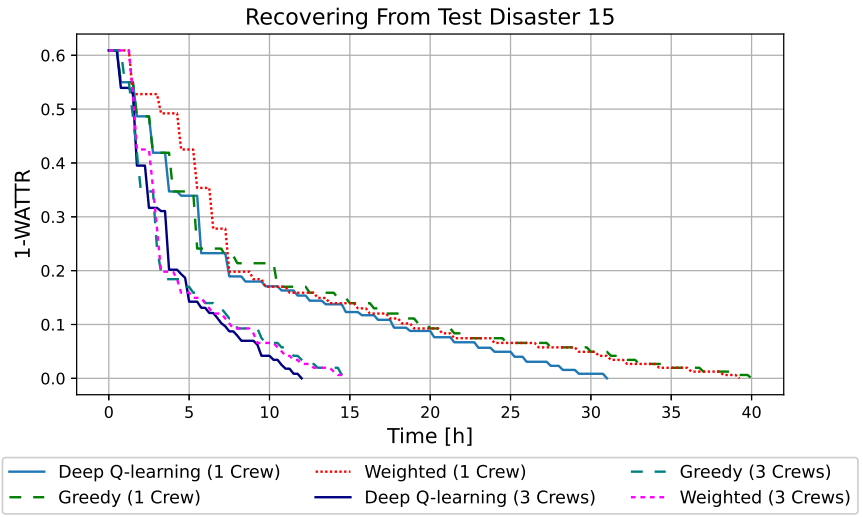


Figure 6.9: **The network's 1-WATTR curves under different approaches for recovering network 2 from test disaster 15.**

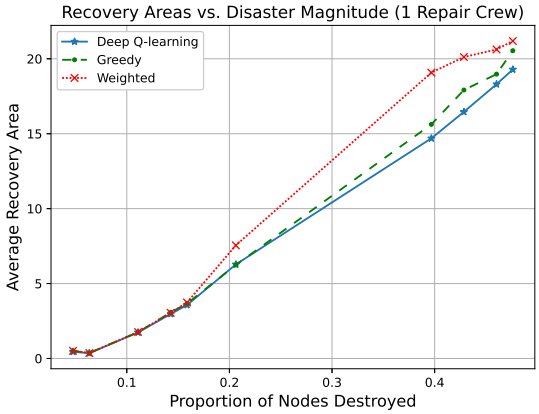
$$\begin{aligned} Travel\ Speed &\sim U_{[60,120]\,km/h} \\ Node\ Replacement\ Time\ &\sim U_{[30,120]\,minutes} \end{aligned} \tag{6.3}$$

With the parameters sampled from these distributions during the training process, we let the repair crews travel at 50 km/h and let node replacements take 75 minutes in the test phase. Because this travel speed is not used in the training episodes, the tests will reveal whether the approach works under unexpected scenarios. After training for 20,000 episodes on network 1 with varying parameters (repeated four times), the results we obtain from a set of 28 test disasters are shown in Figure 6.10.

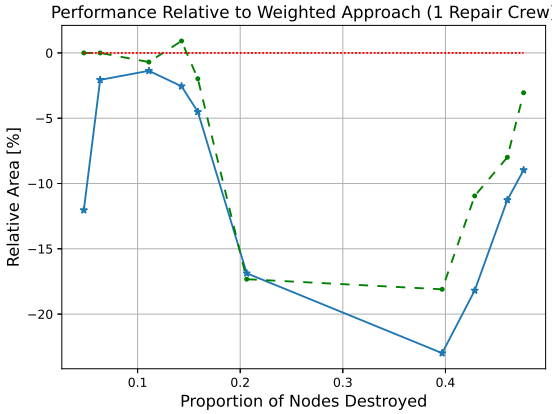
As shown in Figure 6.10, the Deep Q-learning's performance is consistently better than the Weighted approach. But as discussed previously, the characteristics of the test disasters influences the performance of the approaches to some degree. We can see that for the three repair crew setup, our Deep Q-learning approach is far better than the other two approaches. In fact, it performs about 10-20% better than the Greedy approach on disasters that damage more than 15% of the network's nodes. That is, on disasters that damage core nodes, which thus introduce uncertainty on the damage extent.

We can now safely say that it is possible to train the Deep Q-learning algorithm on varying disasters and parameter values, and perform tests with parameter values different from the ones used in training and still get good results.

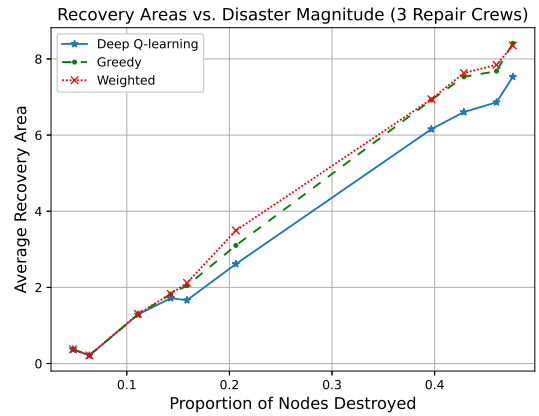
With the same training procedure made on network 2, we obtain the results shown in Figure 6.11. These results reveal similarities to our previous experiment on network 2, shown in Figure 6.6. Even though we use the same test disasters as before, here our Deep Q-learning approach with a single repair crew



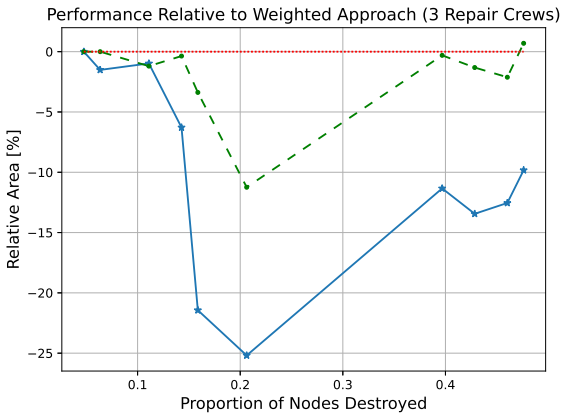
(a) Average recovery areas of each approach with 1 repair crew.



(b) Performances relative to Weighted approach with 1 repair crew.



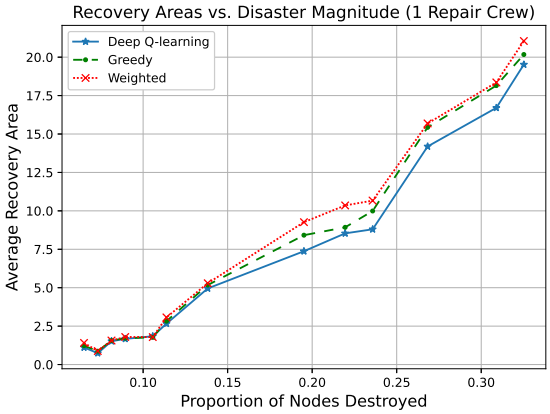
(c) Average recovery areas of each approach with 3 repair crews.



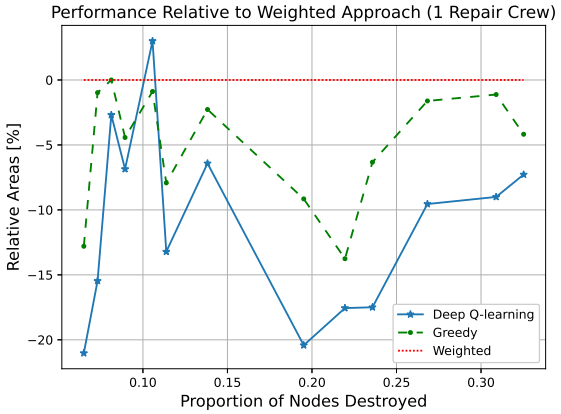
(d) Performances relative to Weighted approach with 3 repair crews.

Figure 6.10: **Performances of each approach on recovering network 1 for both one and three repair crew setups with varied travel speeds and node replacement times during training.**

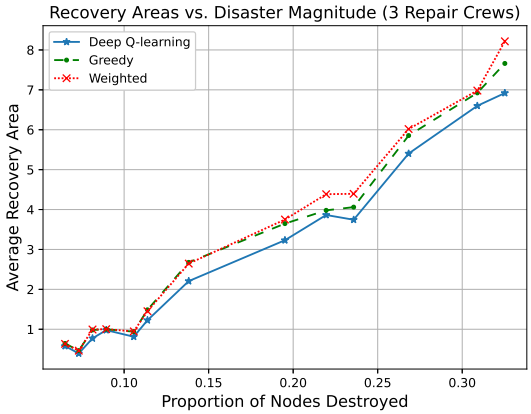
performs not as well as before, relative to the Weighted approach. Still, the difference is only about 5% and is most likely due to insufficient training. With three repair crews, however, our Deep Q-learning approach performs better on average relative to the Weighted approach compared to our previous experiment, shown in Figure 6.6(d). This is because the Weighted approach will make more travel inefficient crew-to-node assignments when the crews operate slower.



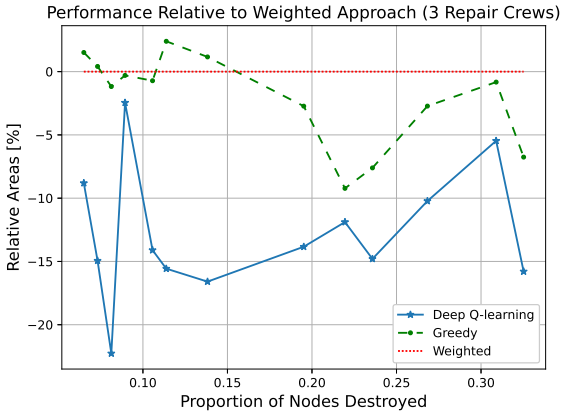
(a) Average recovery areas of each approach with 1 repair crew.



(b) Performances relative to Weighted approach with 1 repair crew.



(c) Average recovery areas of each approach with 3 repair crews.



(d) Performances relative to Weighted approach with 3 repair crews.

Figure 6.11: **Performances of each approach on recovering network 2 for both one and three repair crew setups with varied travel speeds and node replacement times during training.**

Increasing the Uncertainty and Balancing the Node Weights

Our third network was created following the discovery of the issues with networks 1 and 2. That is, the weight differences and low core node counts of these two networks are unrealistic. Network 3 incorporates more core nodes and balances the base station weights better. As noted in table 6.1, network 3 consists of 26 core nodes and 61 base stations.

Adding more core nodes will lead to greater uncertainty of a network's state after a disaster. This will allow for more realistic experiments where the damage extent can not be known after recovering just a few nodes. Furthermore, the base station weights are randomly chosen from a uniform distribution the

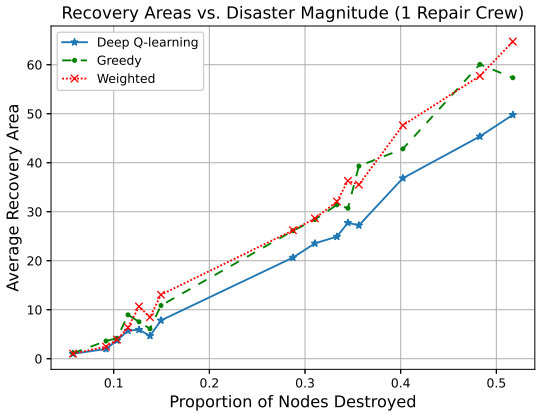
interval of 80 to 120. This prevents unrealistic scenarios where a small set of nodes contribute to majority of the network's WATTR. In addition, distributing weights this way makes the aspect of travel and node replacement times more significant.

For this network, we train our Deep Q-learning agent on 20,000 episodes of disasters generated from Equation 5.1 under the Northridge earthquake. Additionally, we vary the travel speed and node replacement time parameters during training on the same distributions as before. As for the testing phase, we let the parameters equal the same values as before (travel speed of 50 km/h and node replacement time of 75 minutes). We generate 16 test disasters that span the PMF of the training disasters destructiveness, shown in Figure 6.2(c).

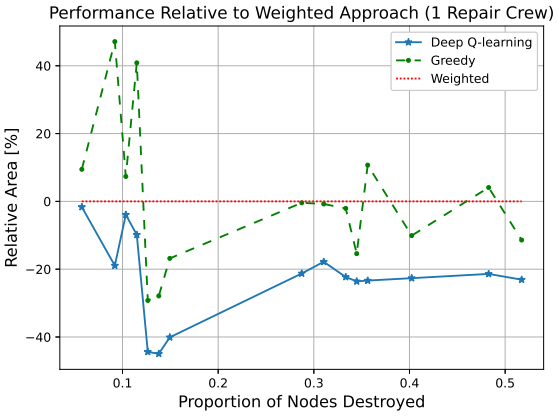
Repeating the training and testing four times, we obtain the results shown in Figure 6.12. The results show that our Deep Q-learning approach consistently outperforms the other two algorithms. Even though our test disasters vary in destructiveness and differ in the spread of damages, our Deep Q-learning approach maintains a 20% better performance over the Weighted algorithms. The results also shows that our approach can make good recovery decisions on scenarios it has not seen or trained on before. The same can be said for the three repair crew setup, except the performance varies slightly more. Lastly, the full training time on this network takes around 1,500 seconds.

6.2.5 Remarks

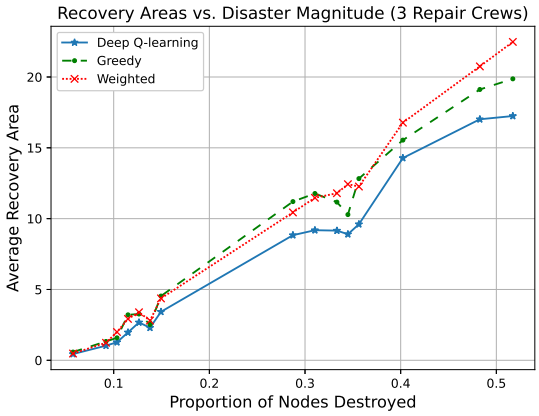
For our experiments, we initially trained our system on a specific test disaster scenario. That is, the disasters used in training were generated such that they matched the state of the test disaster. Because this process was time consuming to do for many test disasters, we explored the possibility of training on potential disaster scenarios of an earthquake. It turns out that the network recovery performance difference between these two training methods are minuscule. Because this was discovered rather late, we did not have much time to incorporate more test disasters in our experiments.



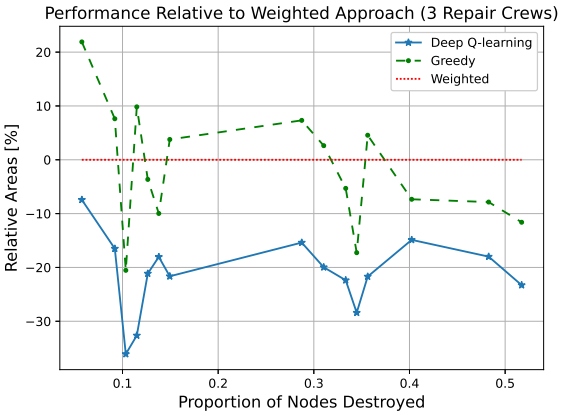
(a) Average recovery areas of each approach with 1 repair crew.



(b) Performances relative to Weighted approach with 1 repair crew.



(c) Average recovery areas of each approach with 3 repair crews.



(d) Performances relative to Weighted approach with 3 repair crews.

Figure 6.12: **Performances of each approach on recovering network 3 for both one and three repair crew setups with varied travel speeds and node replacement times during training.**

Chapter 7

Conclusion

Communication network components are vulnerable to destruction caused by natural disasters, which can subsequently leave countless people unable to contact first responders. Post-disaster network recovery efforts are known to have been frightfully slow in many real-world events. However, recent technological advancements in mobile cell sites have made rapid communication network expansion for feasible in post-disaster scenarios. Still, the problem of how to optimally deploy these cell sites remains open.

In our work, we consider an optimal network recovery strategy to be the one which minimizes the area underneath the curve of the network's 1-WATTR over the recovery process. To address this, we designed a deep reinforcement learning algorithm that is trained to compute optimal repair crew to node deployments, given the observed state of the network and details on the repair crews. Our system takes into account the uncertainty that lies in disconnected nodes when making predictions of the future network state. This is because an unreachable node might survive a disaster and still be functional.

A simulator was made to model the environment state, that our deep reinforcement learning agent interacts with. Its design was kept simple, as it works by repeatedly iterating over instructions that maintain the network state and crew deployments, with each iteration representing a real-life time duration.

We created two algorithms to compare our approach against: a greedy algorithm that prioritizes the recovery of the node resulting the highest connectivity increase over the time it takes to recover it, and a weight based approach in which priority is given to recover the node with the greatest weight. Experiments revealed that our deep reinforcement learning algorithm produces better strategies than the other algorithms in all disaster scenarios. More precisely, on networks 1 and 2, our deep reinforcement learning algorithm performed on average 9% better than greedy algorithm and 15% better than the weighted algorithm. On the third network, our approach performed about 20% better than the other approaches. Our work shows that RL offers a flexible and efficient approach to optimize post-disaster network recovery.

Chapter 8

Future Work

For this work we chose to model an MDP rather than a Partially Observable MDP (POMDP), even though the true underlying state is unknown. This was done because having access to the node failure probabilities would enable for good predictions of the network's WATTR following actions. Furthermore, feeding the DQN with the current and previous environment observations has been reported to aid the agent in learning its environment [18] [13].

Nonetheless, this practice has become obsolete with the rise of the Long Short-Term Memory (LSTM) which is capable of handling partial observability. In essence, an LSTM is a type of a Recurrent Neural Network (RNN) that is designed to capture long term time dependencies [42] [26]. Typically, an LSTM is incorporated between the last hidden layer of the DQN and the output layer to create a Deep Recurrent Q-Network (DRQN) [10].

Other things worth investigating include varying the repair crew travel speeds and the node replacement times within training episodes. This better reflects the uncertainty present in the real-world. Additionally, a reasonable addition would be to introduce a limit to the capacity of mobile cell sites that repair crews can carry.

As technology keeps evolving, new solutions such as UAV base stations emerge. UAV base stations (UABS) have received considerable attention in recent years and are said to be suitable in various post-disaster situations [44] [16] [9]. However, UABS still face challenges with regards to power consumption and optimizing their placements among others. If this technology turns out to be practical, modeling it for our system would be simple.

Another up-and-coming technology is SpaceX's Starlink project, which aims to bring internet access to every corner of the Earth via a satellite internet constellation. In June 2021, after launching close to 1,800 satellites, almost 100,000 people were connected to Starlink through personal user terminals [3]. Additional signs of success come from the fact that ground stations are being made for connecting cloud services to Starlink [34]. Although it is not yet clear how Starlink could be utilized to provide people with network connectivity following a disaster, the technology offers interesting solutions that deserve deeper investigation.

Bibliography

[1] M. G. Azar, R. Munos, M. Ghavamzadeh, and H. Kappen. Speedy q-learning. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2011.

[2] N. Bartolini, S. Ciavarella, T. F. La Porta, and S. Silvestri. Network recovery after massive failures. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2016.

[3] Aimee Chanthadavong. Spacex president says starlink global satellite broadband service to be live by september, 2021. [Online; accessed 24-August-2021].

[4] S. Ciavarella, N. Bartolini, H. Khamfroush, and T. La Porta. Progressive damage assessment and network recovery after massive failures. *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017.

[5] Melissa C Daniels and Barry M Popkin. Impact of water intake on energy intake and weight status: a systematic review. *Nutrition Reviews*, 68(9), September 2010.

[6] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. `https://arxiv.org/abs/1412.6980`, 2017.

[7] Joseph B. Evans, Gary J. Minden, K. Sam Shanmugan, Glenn Prescott, Victor S. Frost, Benjamin J. Ewy, R. Sanchez, C. Sparks, K. Malinimohan, James Roberts, R. G. Plumb, and David Petr. The rapidly deployable radio network. *IEEE Selected Areas in Communications*, 17(4), April 1999.

[8] Sifat Fardousi, Massimo Tornatore, Ferhat Dikbiyik, Charles U. Martel, Sugang Xu, Yusuke Hirota, Yoshinari Awaji, and Biswanath Mukherjee. Joint progressive network and datacenter recovery after large-scale disasters. *IEEE Transactions on Network and Service Management*, 17(3), September 2020.

[9] Azade Fotouhi, Haoran Qiang, Ming Ding, Mahbub Hassan, Lorenzo Galati Giordano, Adrian Garcia-Rodriguez, and Jinhong Yuan. Survey on uav cellular communications: Practical aspects, standardization advancements, regulation, and security challenges. *IEEE Communications Surveys Tutorials*, 21(4):3417–3442, 2019.

[10] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[11] D.G. Honegger and D. Wijewickreme. Handbook of seismic risk analysis and management of civil infrastructure systems.

[12] Genya Ishigaki, Siddartha Devic, Riti Gour, and Jason P. Jue. Deeppr: Progressive recovery for interdependent vnfs with deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 38(10), October 2020.

[13] Jonás Kulhánek, Erik Derner, Tim de Bruin, and Robert Babuska. Vision-based navigation using deep reinforcement learning. *CoRR*, abs/1908.03627, 2019.

[14] Facebook AI Research Labs. Pytorch. `https://pytorch.org/`.

[15] Alexander Lenail. Nn-svg. `https://github.com/alexlenail/NN-SVG`, 2019. Tool for drawing neural network architectures.

[16] Arvind Merwaday and Ismail Guvenc. Uav assisted heterogeneous networks for public safety communications. In *2015 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 329–334, 2015.

[17] Karen Miranda, Antonella Molinaro, and Tahiry Razafindralambo. A survey on rapidly deployable solutions for post-disaster networks. *IEEE Communications Magazine*, 54(4), April 2016.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[20] K. T. Morrison. Rapidly recovering from the catastrophic loss of a major telecommunications office. *IEEE Communications Magazine*, 49(1), January 2011.

[21] NetworkX. Networkx: Network analysis in python. `https://networkx.org/`, 2021.

[22] Jorik Oostenbrink, Fernando A. Kuipers, Poul E. Heegaard, and Bjarne E. Helvik. Evaluating local disaster recovery strategies. *ACM SIGMETRICS Performance Evaluation Review*, 46(2), January 2019.

[23] M. Pourvali, C. Cavdar, K. Shaban, J. Crichigno, and N. Ghani. Post-failure repair for cloud-based infrastructure services after disasters. *Computer Communications*, 111, October 2017.

[24] Mahsa Pourvali, Kaile Liang, Feng Gu, Hao Bai, Khaled Shaban, Samee Khan, and Nasir Ghani. Progressive recovery for network virtualization after large-scale disasters. *2016 International Conference on Computing, Networking and Communications (ICNC)*, February 2016.

[25] Gianluca Rizzo, Sasko Ristov, Thomas Fahringer, Marjan Gusev, Matija Dzanko, Ivana Bilic, Christian Esposito, and Torsten Braun. Emergency networks for post-disaster scenarios. *Guide to Disaster-Resilient Communication Networks. Computer Communications and Networks*, July 2020.

[26] Clément Romac and Vincent Béraud. Deep recurrent q-learning vs deep q-learning on a simple partially observable markov decision process with minecraft. *CoRR*, abs/1903.04311, 2019.

[27] Toshikazu Sakano, Zubair Md. Fadlullah, Hiroki Nishiyama, Masataka Nakazawa, Fumiyuki Adachi, Nei Kato, Atsushi Takahara, Tomoaki Kumagai, Hiromichi Kasahara, and Shigeki Kurihara. Disaster-resilient networking: A new vision based on movable and deployable resource units. *IEEE Network*, 27(4), 2013.

[28] Toshikazu Sakano, Satoshi Kotabe, and Tetsuro Komukai. Overview of movable and deployable ict resource unit architecture. 13, May 2015.

[29] Toshikazu Sakano, Satoshi Kotabe, Tetsuro Komukai, Tomoaki Kumagai, Yoshitaka Shimizu, Atsushi Takahara, Thuan Ngo, Zubair Md. Fadlullah, Hiroki Nishiyama, and Nei Kato. Bringing movable and deployable networks to disaster areas: Development and field test of mdru. *IEEE Network*, 30(1), January 2016.

[30] Toshikazu Sakano, Satoshi Kotabe, Tetsuro Komukai, and Atsushi Takahara. Movable and deployable ict resource unit for instant delivery of local ict services. *2014 Asia-Pacific Microwave Conference*, November 2014.

[31] Solaris Technologies Services. Solaris technologies services. `https://solaristechservices.com/`, 2021.

[32] Jingran Sun and Zhanmin Zhang. A post-disaster resource allocation framework for improving resilience of interdependent infrastructure networks. *Transportation Research Part D Transport and Environment*, 85, August 2020.

[33] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[34] Dan Swinhoe. Spacex's starlink to house satellite ground stations at google data centers, partner on edge, 2021. [Online; accessed 24-August-2021].

[35] Diman Zad Tootaghaj, Hana Khamfroush, Novella Bartolini, Stefano Ciavarella, Seamus Hayes, and Thomas La Porta. Network recovery from massive failures under uncertain knowledge of damages. *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, June 2017.

[36] Anthony M. Townsend and Mitchell L. Moss. Telecommunications infrastructures in disasters: Preparing cities for crisis communications. April 2005.

[37] International Telecommunication Union. Itu handbook on telecommunication outside plants in areas frequently exposed to natural disasters. https://www.itu.int/en/ITU-D/Technology/Documents/OutsidePlants/Appendix3_Q22-1_2.pdf, 2013. [Online; accessed 30-August-2021].

[38] United States Department of Homeland Security. Portable cellular systems application note. https://www.dhs.gov/sites/default/files/publications/Port-Cell-Sys_AppN_0714-508.pdf, 2014.

[39] United States Geological Survey. The modified mercalli intensity scale. https://www.usgs.gov/natural-hazards/earthquake-hazards/science/modified-mercalli-intensity-scale. Last accessed: Jun. 5, 2021.

[40] United States Geological Survey. Search earthquake catalog. https://earthquake.usgs.gov/earthquakes/search/. Last accessed: Apr. 25, 2021.

[41] Jianping Wang, Chunming Qiao, and Hongfang Yu. On progressive network recovery after a major disruption. *2011 Proceedings IEEE INFOCOM*, April 2011.

[42] Daan Wierstra, Alexander Foerster, Jan Peters, and Jürgen Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In Joaquim Marques de Sá, Luís A. Alexandre, Włodzisław Duch, and Danilo Mandic, editors, *Artificial Neural Networks – ICANN 2007*, pages 697–706, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[43] Wikipedia contributors. Haversine formula — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Haversine_formula&oldid=1022071808, 2021. [Online; accessed 22-May-2021].

[44] Yong Zeng, Rui Zhang, and Teng Joon Lim. Wireless communications with unmanned aerial vehicles: opportunities and challenges. *IEEE Communications Magazine*, 54(5):36–42, 2016.

[45] Yangming Zhao, Mohammed Pithapur, and Chunming Qiao. On progressive recovery in interdependent cyber physical systems. *2016 IEEE Global Communications Conference (GLOBECOM)*, December 2016.