Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Huda Mohamad Muataz Albizreh Sakka Amini

Reinforcement Learning for Mobile Network Optimization

Master's thesis in Computer Science
Supervisor: Keith L. Downing
June 2023

NTNU
Norwegian University of
Science and Technology

Huda Mohamad Muataz Albizreh Sakka Amini

Reinforcement Learning for Mobile Network Optimization

Master's thesis in Computer Science
Supervisor: Keith L. Downing
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Reinforcement Learning for Mobile Network Optimization

Author: Huda Mohamad Muataz Albizreh Sakka Amini

Computer Science, Master's Thesis, June 2023

Supervisor: Keith L. Downing

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Artificial Intelligence - Department of Computer Science

Preface

This thesis represents the TDT4900 - Computer Science Master's Thesis. First, I want to thank my family for all their support. Furthermore, I would like to thank my supervisor, Professor Keith L. Downing, for his guidance during the project. I want also to thank Vegard Edvardsen, Gard Spreemann, and Jeriek Van den Abeele, the supervisors from Telenor Research. They have provided me with many helpful and valuable resources, ideas, and guidance. They helped me to understand the general problem and the existing system. In addition, they were open for discussion and available in all project phases.

Huda Mohamad Muataz Albizreh Sakka Amini
Trondheim, June 29, 2023

Abstract

This thesis uses reinforcement learning (RL) to optimize mobile network performance. The main focus is to optimize the performance of the handover process and its algorithms. The handover is where a base station (BS) decides if a connected user should be handed over to another BS while the user is moving to receive the best signal power and quality. The handover algorithms used are A2A4 RSRQ and A3 RSRP. The Soft actor-critic (SAC) RL algorithm has been used in many experiments and analyzed to optimize the handover performance, maximize the total data throughput, signal quality, and signal power received by all users, and minimize the number of handovers as possible. The performance of SAC is compared to the performance of the Twin Delayed Deep Deterministic (TD3) RL algorithm to evaluate which provides the best optimization. Both RL algorithms are used for continuous state and action spaces' environments. The mobile network, which includes some base stations and users, is simulated using the ns-3 simulator since it is very complex to apply the research and evaluate the performance on real-world networks. The structure of the communication channels between the RL agent and the simulator is also explained in this thesis. Several related research articles are summarised and discussed. A background explanation of the basic ideas and technologies used is added too. After applying different experiments using different setups, many interesting results were collected. Each experiment focused on a different perspective and had its unique goal. The results showed that both RL algorithms could improve the total data throughput, signal quality, and signal power, but TD3 showed a faster behavior than SAC. Both TD3 and SAC algorithms showed an adaptive behavior depending on the simulated network scenario, but SAC couldn't adapt to the network changes in all experiments. In addition, the A2A4 RSRQ handover algorithm showed different and more reasonable results than the A3 algorithm in different experiments. The improvements in the A2A4 algorithm were obvious in almost all experiments, while A3 was not improved in all experiments.

Sammendrag

Denne oppgaven bruker forsterkningslæring (RL) for å optimalisere ytelsen til mobilnettverket. Hovedfokuset er å optimalisere ytelsen til overleveringsprosessen (Handover) og algoritmene sine. Overleveringen er når en basestasjon (BS) bestemmer om en tilkoblet bruker skal overleveres til en annen BS mens brukeren beveger seg for å motta den beste signalstyrken og -kvaliteten. Overleveringsalgoritmene som brukes er A2A4 RSRQ og A3 RSRP. Soft actor-critic (SAC) RL-algoritmen har blitt brukt i mange eksperimenter og analysert for å optimere overleveringsytelsen, maksimere den totale datagjennomstrømningen, signalkvaliteten og signaleffekten som mottas av alle brukere, og minimere overleveringer som mulig. Ytelsen til SAC sammenlignes med ytelsen til Twin Delayed Deep Deterministic (TD3) RL-algoritmen for å evaluere hvilken som gir den beste optimaliseringen. Begge RL-algoritmene brukes for kontinuerlig observasjoner og handlingsroms miljøer. Mobilnettverket, som inkluderer noen basestasjoner og brukere, simuleres ved hjelp av ns-3-simulatoren siden det er svært komplisert å bruke forskningen og evaluere ytelsen på nettverk i den virkelige verden. Strukturen i kommunikasjonskanalene mellom RL-agenten og simulatoren er også forklart i denne oppgaven. Flere relaterte forskningsartikler er oppsummert og diskutert. En bakgrunnsforklaring av de grunnleggende ideene og teknologiene som brukes er også lagt til. Etter å ha brukt forskjellige eksperimenter med forskjellige oppsett, ble mange interessante resultater samlet. Hvert eksperiment fokuserte på et unikt perspektiv og hadde sitt unike mål. Resultatene viste at begge RL-algoritmene kunne forbedre den totale datagjennomstrømningen, signalkvaliteten og signalstyrken, men TD3 viste en raskere oppførsel enn SAC. Både TD3- og SAC-algoritmer viste en adaptiv oppførsel avhengig av det simulerte nettverksscenarioet, men SAC kunne ikke tilpasse seg nettverksendringene i alle eksperimenter. I tillegg viste A2A4 RSRQ-overleveringsalgoritmen andre og mer fornuftige resultater enn A3-algoritmen i forskjellige eksperimenter. Forbedringene i A2A4-algoritmen var tydelige i nesten alle eksperimenter, mens A3 ikke ble forbedret i alle eksperimenter.

Acronyms

AI Artificial intelligence. 22, 23

ANOH Average Number Of Handovers. 35

BS Base Station. xiii, 1, 10–12, 17–23, 25, 31, 65

BSs Base Stations. 10, 14, 18, 19, 21, 25, 29, 30, 41, 43–46

CACLA Continuous Actor-Critic Learning Automation. 9

DDPG Deep Deterministic Policy Gradient. viii, 8, 15, 18, 24, 25, 36, 73

DQL Deep Q-Learning. 15, 73

DQN Deep Q-Network. 8, 20

DRL Deep Reinforcement Learning. vii

ENB Evolved Node B which is the base station equipment. xiii, 33

ENBs Evolved Node B which is the base station equipment. 28, 42, 44–47

eNodeB Evolved Node B which is the base station equipment. 11

eNodeBs Evolved Node B which is the base station equipment. 30

FORLORN A Framework for Comparing Offline Methods and Reinforcement Learning for Optimization of RAN Parameters. vii, 17, 27, 28

HO Handover. 14, 19–22

NTNU Norwegian University of Science and Technology. 39

RAN Radio access network. vii, 1, 9, 10, 17, 18, 27

RL Reinforcement Learning. vii–ix, xi, 1–3, 5–8, 14, 15, 17–25, 27–30, 34–47, 50–54, 56, 57, 60, 68–73

RNG Random Number Generator. 28, 30

RRC Radio Resource Control. 11

RSRP Reference Signal Received Power. xi–xiii, 11, 12, 14, 15, 19–22, 27, 28, 30, 32–34, 38, 39, 42, 46, 67, 71–73

RSRQ Reference Signal Received Quality. xi–xiii, 11–13, 15, 18, 19, 27, 28, 30, 32–35, 38, 39, 42, 45, 46, 50, 54–56, 60–62, 68, 70–73, 81, 82

RSSI Received Signal Strength Indicator. 11

SAC Soft Actor Critic. viii, xi–xiii, 2, 8, 9, 14, 15, 17, 24, 25, 27, 34, 36–39, 42–45, 50–56, 69, 70, 72, 73, 82, 84

SAP Service Access Point. 11

SD Standard Deviation. 52, 58, 59

SLA Service Level Agreement. 18, 25

TD Temporal Difference. 7

TD3 Twin Delayed Deep Deterministic Policy Gradient. viii, xi–xiii, 8, 14, 15, 25, 27, 36, 39, 42, 44, 50–56, 69, 70, 72, 73, 81, 83

TTT TimeToTrigger. 12, 14, 20

UE User Equipment. xi, 9–12, 14, 15, 18, 20, 21, 23, 25, 29, 30, 32, 35

UEs User Equipment. xii, 2, 10, 15, 18, 20, 21, 25, 27–35, 41, 43–47, 50, 55, 60, 64, 65, 73

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The project initially explores the benefits of using Reinforcement Learning (RL) to improve the performance of continuously controlling telecommunications networks. A team from Telenor research center was interested in that and had been working on finding out how smarter RL algorithms will improve 4G/5G cable or wifi mobile networks using ns-3 open source network simulator. The project's direction was quite open and many options and starting points were mentioned in the beginning (e.g. Scheduling/resource allocation, handovers, load balancing, beam management, and power saving). After some discussions, the main direction of this project was chosen to be *handovers*. The handover is where a Base Station (BS) decides if the user should move to another BS while the user is moving to receive the best signal power and quality.

The network scenario is implemented in the ns-3 simulator, then the interface of the RL environment is built in Python. Finally, the RL agent is developed to optimize the handover performance.

1.1 Background and Motivation

The demands on the Radio Access Networks (RAN) in mobile networks are increasing, and mobile data traffic is expected to continue growing exponentially, currently and in the following years [1]. Therefore, there is a huge need to optimize and improve these networks' quality to meet the customer's needs. Reinforcement Learning has shown a lot of improvements in controlling continuous real-world systems .e.g healthcare [15], natural language processing [16].

A team of three researchers in Telenor Research has already started comparing offline methods and Reinforcement Learning for the optimization of RAN parameters in a 4G network. [1]. This project takes their work as a foundation and continues working further. In the start phase of the project, many starting points have been listed (e.g. Scheduling/resource allocation, handovers, load balancing, beam management, and power saving). The chosen direction is handovers, which is explained in Section 2.4. The ability to improve its performance using Reinforcement Learning is the main point to discuss and approve during this project. Since Reinforcement Learning depends on trial-and-error data, the ns-3 simulator is needed to generate the simulated situations and to let the RL agent discover and configure them since it will be difficult to use real-world data.

1.2 Goals and Research Questions

The project's main goal is to explore how reinforcement learning can optimize mobile network performance. It is applied to the handover algorithm to optimize it specifically because of its frequent needs nowadays. The base stations are responsible for taking the handover decision for a user device based on the measurements received from it. The handover algorithm takes these values in, decides, and triggers the handover from one base station to another if it is needed. This algorithm takes the decision based on some attributes, and the RL agent will be responsible for optimizing these attributes to optimize and improve the performance of the handover algorithm. This is the main goal of the project.

Goal *Investigate the potential of RL to optimize mobile network performance*

However, there are several handover algorithms. Each of them takes its decision based on different attributes. The decision is made by comparing measurements received from the user devices with the handover algorithms' attributes as explained in section 2.4.1. The values of the handover attributes are the changeable part of the handover algorithm and their values will be chosen by the RL agent to get the best handover performance.

Research question 1 *Which handover algorithm and attribute values will be used for the best optimization?*

The design of the RL agent is an essential part of the project. The performance of the *soft actor-critic* (SAC) RL algorithm will be evaluated and compared with *the Twin-Delayed Deep Deterministic* (TD3) algorithm. Both RL algorithms are used for environments with continuous state/action spaces. Additionally, for the RL agent to learn the best actions, it must receive a reasonable reward (positive/negative or good/bad). The reward gives feedback to the agent about the performed action. Therefore, choosing a suitable reward calculation is important to guide the agent during the learning process. So choosing how parameters, received from UEs in the measurement reports, affect the reward finding is an essential part too.

Research question 2 *Does the soft actor-critic model provide advantages over the Twin-Delayed Deep Deterministic model for tackling the handover problem?*

1.3 Research Method

Since the simulated network built in the ns-3 simulator was implemented first, understanding the simulator and how it is structured and used was the first step of the research. Some initial experiments were done using ns-3 to understand the simulator structure and to find the network setup utilized by the agent. The simulated network built by the Telenor research team was used as a foundation and updated to build the dynamic simulated network used in this project. The simulated network used by the RL agent is fixed to be regenerated in every RL episode.

In addition, many related research papers about mobile networks and reinforcement learning were read for more understanding and research guidance. The main focus was on designing the RL agent, so some reading about the different RL algorithms for different uses was also very important, especially reading various scientific papers about different RL systems designs applied to similar problems. This wasn't the only method to find the best RL design. Evaluating the performance of the chosen algorithms was necessary since the outcome might differ between different solutions. The optimal handover attribute values decided are not constant in all situations. Therefore, validating the performance of different solution and network situations, which includes the positions, power level, user count, and some other parameters, was also important.

Many experiments are applied with different purposes to answer the research questions from different perspectives and find the best RL model, reward function, and handover optimization. The two handover algorithms were used and the RL agents' performances were validated in each experiment. The validation is done based on the signal power and quality levels and the data throughput.

1.4 Thesis Structure

This thesis starts with the introduction chapter 1 which includes a brief introduction about the problem in general, the background for it, and the research method. Some research questions are raised to discuss the problem from its different sides. These questions are going to be answered during the rest of the report by showing and evaluating the methods used and their results.

The second chapter 2 is the background theory and motivation, where the main components of this project will be explained briefly. These components are:

- Reinforcement Learning and some of its algorithms and elements, especially some of the RL algorithms used for continuous state/action spaces
- An introduction to the mobile network and its components
- The ns-3 simulator
- A brief explanation of the handover process and its main algorithms and attributes
- The Background Theory and Motivation

The third chapter 3 includes some related work for this project. They show how other scientists have used RL for behavior optimization of problems related to mobile networks or other problems. 13 different papers are summarized in this chapter.

The next chapter 4 is the methodology chapter. It explains the structure of the RL environment, the simulated networks used, the state and action spaces, and the reward functions. It includes also the RL system setup and the RL algorithms used in this project.

The next chapter 5 is the experiments and results chapter. The description of the experimental plan and all experiment setups are explained in this chapter. This chapter provides an insight into the results achieved from these experiments. The final results are extracted and evaluated in different directions after applying the different experiments.

Finally, the conclusion and future work chapter 6 provides a conclusion about the research and results. It addresses the research questions and summarizes how they were solved during this project. Moreover, this chapter includes a list of the possible future steps to be applied further.

Chapter 2

Background Theory and Motivation

This chapter will go through the main and basic topics and technologies for this project. This project aims to optimize the usage and performance of the mobile network, specifically the handover algorithm. Therefore, mobile networks and the handover process are explained briefly. For optimization, Reinforcement Learning (RL) is the main field to study and apply, so it's important to describe it too. The ns-3 simulator is used to create the simulated networks, so it is briefly explained.

2.1 Reinforcement Learning (RL)

Machine learning includes three main types of learning: supervised, Unsupervised, and Reinforcement. Supervised learning works with labeled data where the target to be achieved is clear and obvious. It's trained using neural networks with some labeled training data to predict the target label for unlabeled data. Unsupervised learning works on unlabeled data and uses neural networks too. Its goal is to cluster or collect the data into groups with the same properties. Then, after the training, it predicts which group to add the new data to.

However, the main idea back Reinforcement Learning is to learn by interacting with an environment that can be a game, a road for a self-driving car, networking, recommendation systems, and other examples of real-world systems [17]. In the environment, the agent can be described as a robot or a brain. An RL agent, who has no previous knowledge about the environment, learns the best way to interact with it by mapping the environment states to its performed actions and maximizing a reward r [18]. This interaction between the RL agent and the environment is generally shown in Figure 2.1. The current state s of the environment changes to a new one (successor state $s^{'}$) after each action a. The action a is decided by the agent. The RL agent doesn't know what the best action to take in a specific state is, so it has to learn that by trying different actions during the training period and increasing the reward to be the highest possible.

The elements of RL are [18]:

- **The policy** π is a probability distribution over all actions in each state. This distribution guides the agent to choose the actions with the highest probabilities and create an action path to achieve the best performance.
- **The reward signal r** is a feedback value from the environment to the agent to validate the action applied in a particular state.

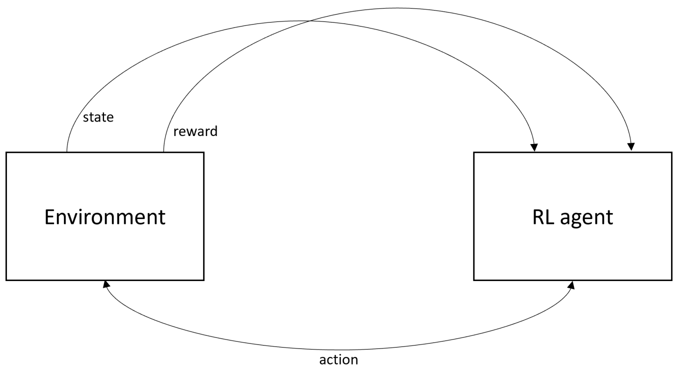


Figure 2.1: Agent and Environment in reinforcement learning

- **A value function v** is the total and cumulative reward received by taking a sequence of actions from a state s until the final state. The calculation of the value function can be done based on the state $V(s)$ or on both state and action pair $V(s,a)$ or $Q(s,a)$ as in Q-learning, which is explained in section 2.1.1.
- **A model of the environment** is used to plan which states in the environment will occur if a sequence of actions is taken. The actions are just tested but not performed.

The mathematical formulation of RL can be done using a Markov Decision Process (MDP) [19] and the Bellman equations [20]
The Bellman equation of the value function: [21]

$$V(s) = \max_a(r(s,a) + \gamma V(s^{'}))(2.1)$$

Where γ is a discount factor that is a value between [0, 1]. This value is multiplied by the reward to show the importance of the future reward in comparison to the current reward, if the discount factor is small in some environments, it reduces the delayed reward to guide the agent that immediate rewards are necessary.

RL algorithms can be implemented in different types:

- **On-policy**: The agent uses just one policy to generate the training actions and learn from its experience by optimizing this policy.

- **Off-policy**: It has two different policies: behavior and target policies. The agent learns and optimizes the best target policy by following the behavior policy which interacts with the environment.

- **Model-based**: The agent explores the environment and creates a model of it. That allows the agent to plan its future decisions and optimize its behavior.

- **Model-free**: The agent learns from interactions with the environment by following its policy without building any model. It is used in situations with complex or unpredictable environments.

2.1.1 Temporal difference learning (TD)

Temporal difference learning is model-free learning. Its methods work by bootstrapping from the outcome of the value function directly in each step. There is no need to wait for the whole episode to be finished or to get the final results [18]. Some RL algorithms based on TD learning:

Actor-critic Algorithm

It is an on-policy TD method. The agent learns the best order of actions, which is the optimal policy, by using two separate parts. They are the actor and the critic, illustrated in Figure 2.2 with their interaction with the environment. The actor represents the policy. Its work is to select the actions based on the learned policy. When a new action, that is not discovered yet with a specific state or an action that gives a better outcome for the current state, is met, the actor will update and optimize the policy. The estimated value function is the critic. The values produced by the critic are sent to the actor to validate the action taken in the current state. Thus, the policy will be updated by the actor, as mentioned. [18]

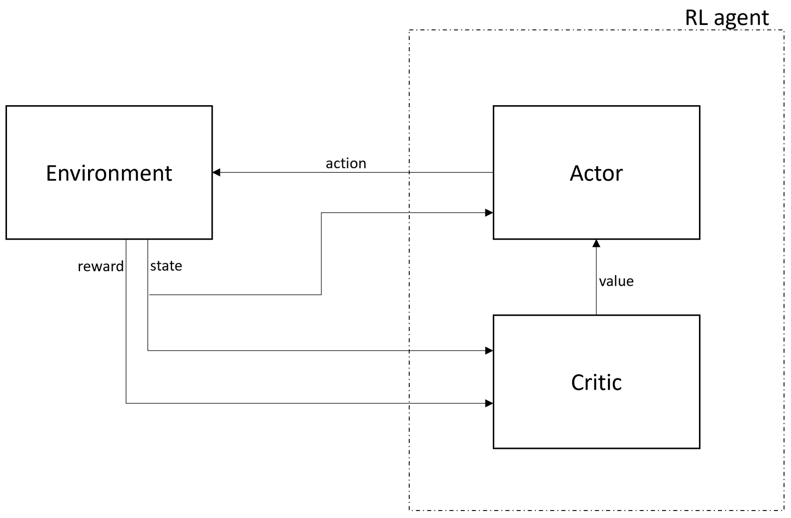


Figure 2.2: The Actor-Critic Architecture. **This figure is inspired by figure 2 in [22]**

Q-Learning Algorithm

It is an off-policy TD control algorithm. The value function learned by the agent depends on both the state and the action, and it is called the Q value $Q(s, a)$. It stores the Q values for each state-action pair in a Q-Table. It aims to find action and state pairs that maximize the Q-value. [18]. The Bellman equation of the Q-value is shown in 2.2:

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_a Q(s^{'}, a) - Q(s, a)](2.2)$$

α is the learning rate or the step-size parameter to control the difference between the next Q value and the current one [18]. γ is the discount factor.

2.1.2 RL algorithms for continuous environments

Below is a list of some good RL algorithms used to deal with continuous environments with high-complexity action and state spaces.

Deep Q Learning (DQN)

It is the Q-learning algorithm that uses a neural network instead of Q-tables. The neural network takes the state as an input and returns the Q value for each possible action as an output. Then the action with the highest Q value is chosen. It learns by updating the neural network weights after applying the action and receiving the reward. This algorithm is used in more complex environments involving big state/action spaces. [23]

Deep Deterministic Policy Gradient (DDPG)

A model-free and off-policy learning algorithm implemented based on both deterministic policy gradients and Deep Q Learning (DQN) (using a replay buffer to increase the efficiency of learning by saving and reusing past results). It uses the actor-critic algorithm, where both actor and critic are neural networks based. The actor finds the behavior by deterministically mapping states to a specific action. It doesn't use probability distribution for actions. At the same time, the learning process for the critic is based on Q-Learning. It's easily used for difficult problems with big networks and continuous action spaces where action optimization in each step is needed to find the greedy policy. [24]. This algorithm is used in the related work 3.2. One other application showed its success in another field. [25]. However, DDPG has function approximation error, leading to value overestimation. Thus a sub-optimal policy is shown in [26].

Twin Delayed Deep Deterministic Policy Gradient (TD3)

A model-free and off-policy learning algorithm. It's an extension of the DDPG algorithm to solve the problem of overestimating value functions that can happen in DDPG. Therefore it learns two Q values, using two critics networks, and chooses the minimum one to be the target Q value during policy update. That is the reason for the Twin word in the algorithm name. TD3 works well for continuous action/state spaces. It updates the behavior policy in the actor with less frequency than the Q value in the critics for more network stability. TD3 trains a deterministic policy and adds some noise to target Q values to reduce differences and smooth the target policy. That happens to prevent overfitting. This makes the policy more stable in choosing the best actions. [27], [26].

Soft Actor-Critic (SAC)

Like TD3, SAC is an off-policy and the most efficient model-free RL algorithm [28]. It also uses two critic networks that produce two Q values. It then chooses the minimum one. It differs from other RL algorithms because it works based on the maximum entropy RL framework [29] besides maximizing the reward to optimize the behavior policy. That means that the optimal policy is the policy that maximizes the long-term reward and entropy term.

The entropy represents the randomization value. In a probability distribution, the entropy value will increase in parallel with the uncertainty level. When the probability is near 0.5, the uncertainty is the highest; thus, the entropy is the highest. In contrast, it decreases when the probability approaches the bounds, which are 0 and 1. Here, the probability is going toward being deterministic. The entropy value in RL is high when the agent acts as randomly as possible and low when the agent repeats the same actions. That means that a higher uncertainty level refers to a higher entropy value.

Maximum entropy improves exploration and robustness [29]. That is because SAC optimizes a stochastic policy and prevents the agent from converging toward a local optima. The stochastic policy is used in real-world or simulator applications since it provides higher robustness and more randomization than the deterministic policy that uses a smaller space of actions. This algorithm uses the actor-critic algorithm based on neural networks. The actor's mission is to find the optimized policy. The critic is responsible for the Q value. SAC have shown state-of-the-art performance on many tasks with continuous and high-dimension spaces. [30] It generalizes the final result since it discovers multiple situations more widely. [31] [32]. It is used in the related works sections 3.12 and 3.13 and showed good results.

Continuous Actor-Critic Learning Automation (CACLA)

The CACLA algorithm is a model-free algorithm used for continuous environments based on using parameterized function approximators (FAs) to replace the tables used in the actor-critic algorithm by FAs [33]. It has shown promising results in many tasks [34] and has been used in the LTE schedule as explained in [35].

2.2 Introduction to Mobile Networks

The mobile network (cellular network) is a wireless network that connects and provides signals to user equipment (UE) like smartphones, tablets, home machines, or other devices which need to be connected. It consists of two parts which are the Radio Access Network (RAN) and the Mobile Core. [36]

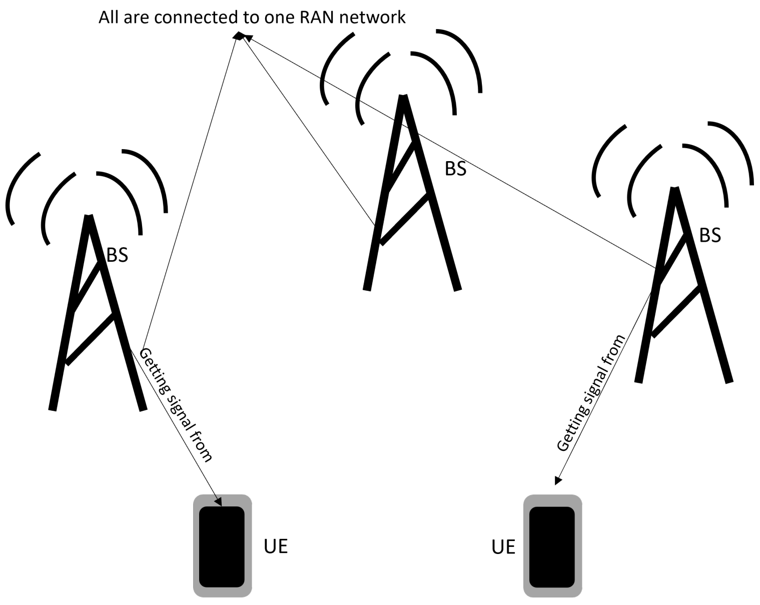


Figure 2.3: Radio Access Network illustration

2.2.1 RAN

RAN is a radio link between UEs and the controlling stations, which provide the signals to control the information, calls, locations, and other data transferred through the network. These stations are known as towers, cells, or base stations (BSs) in mobile networks. All elements connected to RAN are called nodes. RAN covers a specific area; therefore, there are many RANs for many areas [37]. Figure 2.3 shows an illustration of how BSs are connected with each other and with the UEs in RAN.

2.2.2 Mobile core

The mobile core connects UEs to external networks and uses different network services by providing external packet data network connectivity and ensuring their authentication and service qualities [36].

2.3 NS-3 Simulator

It's an open-source platform that allows users to simulate mobile or internet network structures and perform some applications on them instead of applying them on real-world networks. That makes it easier to conduct many experiments on such networks and develop their performance in both research and education. Any network structure created in ns-3 is written in C++ since it is C++ based. It runs on the command line with specific building, running, and updating commands. [38].

There is also a Python wrapper (packages) for the C++-based ns-3 to facilitate Python programming using ns-3. [39], but it does not cover all ns-3 functionalities. While using C++-based ns-3, other software can be needed for visualization or animation. Therefore it's possible to use such software in combination with ns-3 libraries.

Since this project cannot be performed with real systems, the network is highly controlled, and not only a single representation of the network will be tested, ns-3 simulator needs to be used to study the behaviors of the network. Even though there is a Python wrapper for ns-3, not all the functions are included in Python. Therefore the C++ part will be used to create the simulated network to take advantage of the functions needed in the development. It runs on Linux or macOS operating systems. Therefore Ubuntu 22.04.1 on Windows is used to run the simulator [39].

2.4 The Handover Process

A UE needs to be transferred between signal providers, which are BSs while it is moving and its call is in progress. It will be connected to the nearest BS that delivers the best signal quality to keep the call on and not affect it in any way. [40] Nowadays, a massive number of users use mobile networks. Because of the development happening in transportation and the speeds people move in, there is a considerable demand for handover applications. The positions of the BSs are constant, and the UEs move between them a lot. BSs serve the UEs and need to provide them with a high-level quality power of signals, especially with the fourth and fifth generations of mobile networks (4G/5G).

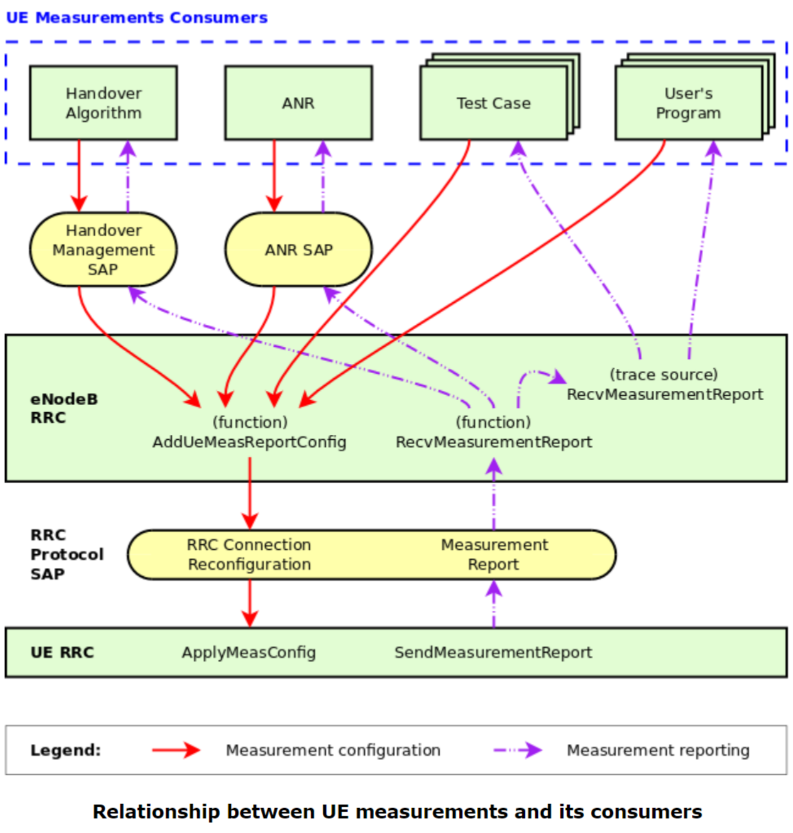

UE Measurements Consumers
Handover Algorithm
ANR
Test Case
User's Program
Handover Management SAP
ANR SAP
eNodeB RRC
(trace source) RecvMeasurementReport
(function) AddUeMeasReportConfig
(function) RecvMeasurementReport
RRC Protocol SAP
RRC Connection Reconfiguration
Measurement Report
UE RRC
ApplyMeasConfig
SendMeasurementReport
Legend:
Measurement configuration
Measurement reporting
Relationship between UE measurements and its consumers

Figure 2.4: Relationship between UE measurements and its consumers. **(This figure is taken from NS3-Manual [14])**. This Figure shows the flow of the measurement report configured and sent from the UE Radio Resource Control (RRC) level with a measurements function. The report is then sent to the RRC level in the eNodeB, the base station, for decision-making. The decisions include the handover algorithm and Automatic Neighbour Relation (ANR) to manage the relations of the BS with its neighbors, user's programs, and test cases. SAP is the Service Access Point. After applying some changes, the handover algorithm and other consumers can also request configurations on the measurement report. This request will be sent to the UE by the eNodeB (BS).

The first step in handover starts from the UE since it sends a measurement report to the base stations. Figure 2.4 shows the flow of measurement reports from a UE to the BS. These reports include some values to measure the signals which are mentioned in [41]:

- **Reference Signal Received Power (RSRP)** is the average power of Reference signal (RS) within a specific frequency bandwidth.
- **Reference Signal Received Quality (RSRQ)** provides validation of the signal quality of the connection. It is computed based on RSRP, the number of used resource blocks (N), and the Received Signal Strength Indicator (RSSI). This is the formula for it: RSRQ = (N * RSRP) / RSSI.

More information about these values, their calculation formulas, and usage examples are explained in [42].

The network, which contains the source BS (the serving cell) and the target BS, decides when to trigger the handover and control its process. The different types of events that can trigger a measurement report are five events A1, A2, A3, A4, and A5, which are explained in Table I in [43]. However, the three events that will be used in this project are:

1. Event A2: (serving cell's RSRQ < serving cell's threshold) which means that the UE has poor signal quality.
2. Event A3 (neighbor cell's RSRP > serving cell's RSRP) The handover is triggered for the UE to the neighbor cell which has better signal power.
3. Event A4 (neighbor cell's RSRQ > serving cell's threshold) is used to store the RSRQ of the neighboring cell.

2.4.1 Handover algorithms in NS-3

In ns-3 there are three implemented handover algorithms [44] which are:

1. **A2-A4-RSRQ**: As illustrated in Figure 2.5, the cell, which is the base station, triggers the handover based on RSRQ values received. After getting the measurements reports, and based on the A2 and A4 events, the algorithm checks if the serving cell's RSRQ < serving cell's threshold, then it finds the best RSRQ between neighbor cells, checks that the UE will get a better signal quality from the current serving cell, then the current cell applies the handover to this neighbor [45].

   The attributes used in this algorithm are:

   - **ServingCellThreshold:** It is a determined value that can be changed or optimized. The RSRQ of the serving cell needs to be lower than it to trigger the handover, corresponding to the A2 event.
   - **NeighbourCellOffset:** It is also a determined and adjustable value. The difference between neighbor RSRQ and the serving cell RSRQ needs to be bigger or equal to this value to trigger the handover.

2. **Strongest cell handover algorithm (A3-RSRP)**: triggers the handover based on the RSRP value for the serving and neighbor cells, which means the A3 event. It finds the best neighbor cell that delivers the best RSRP value [46]. The adjustable attributes used in this algorithm are:

   - **Hysteresis**: This value controls how much the neighbor cell RSRP value needs to be higher than the serving cell RSRP value.
   - **TimeToTrigger (TTT)**: The neighbor cell RSRP needs to be higher than serving cell RSRP with Hysteresis value or more during a determined period which is TTT.

   Figure 2.6 shows the usage of these two attributes in this handover between two base stations.

3. **No-op** It's straightforward and can be used to stop the triggering of handover automatically.

   New handover algorithm from scratch: Implementing a new handover algorithm in ns-3 with a chosen sequence of functions and attributes is possible.

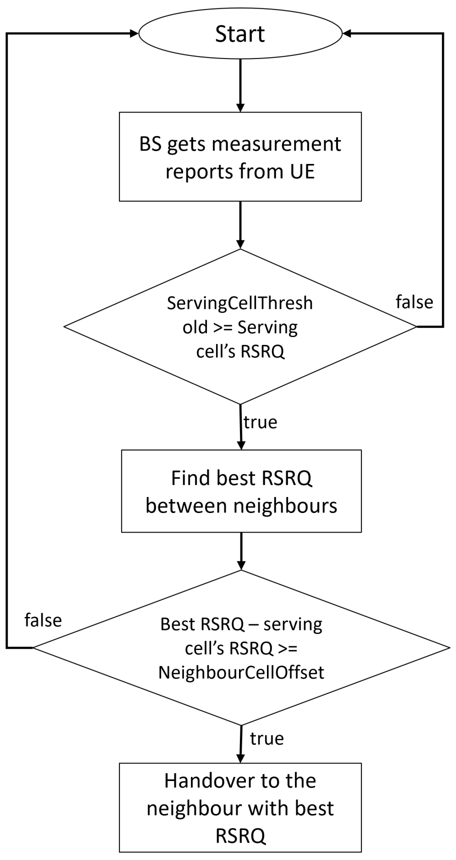

Start
BS gets measurement reports from UE
ServingCellThreshold >= Serving cell's RSRQ
false
true
Find best RSRQ between neighbours
false
Best RSRQ – serving cell's RSRQ >= NeighbourCellOffset
true
Handover to the neighbour with best RSRQ

Figure 2.5: A2-A4-RSRQ algorithm. **Inspired by the figure in [45]**

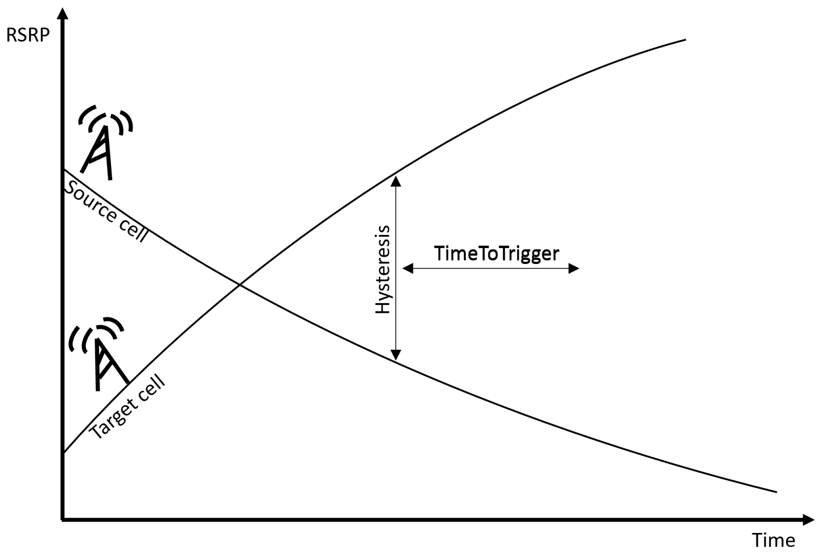


Figure 2.6: The handover process for a UE between source and target cells based on the A3-RSRP algorithm and its attributes (Hysteresis and TTT). The two curves represent the RSRP values delivered to that user from the source and target base stations over time. The Hysteresis is the value to be compared with the RSRP difference between the source and target BSs while TimeToTrigger is the period used to check that the difference is still equal or higher than the Hysteresis value. **The figure is inspired by the figure in [47].**

2.5 Background Theory

There is a massive demand for the handover since the number of mobile network users is growing. The users are in continuous movement at different speeds. For example, they can have a low speed while walking or a high speed if they move with cars or trains. The aim is to maximize the power and quality of the signal, and the data throughput delivered to the users and reduce the HO frequency if possible. The users should not experience any signal problems while moving even though the handover action is triggered. An RL agent will be trained to optimize the handover algorithm attributes to achieve the wanted results.

The RL environment has continuous action space in this case since the handover attribute values are real. The RL agent's goal is to pick the optimal attribute values for different network scenarios. The state space is also continuous since the data throughput and signal quality, and power delivered by each user are real values, and the number of their combinations is infinite. More details about the state and action spaces are explained in the methodology chapter 4. Two reinforcement learning algorithms will be used to optimize the handover performance. They are Soft Actor Critic (SAC) and Twin Delayed Deep Deterministic Policy Gradient (TD3). Both algorithms are used for continuous environment situations.

2.6 Motivation

The topic of optimizing mobile networks is quite extensive. It contains many possibilities like **Handover** between base stations to ensure that all UEs receive the best signal quality and power. **Resource allocation** between users, where the base stations need to control the signal quality and frequency to be delivered to each user. **Power Saving** by finding an optimal usage for the electrical power used by the base stations during their work. Other ideas have also been mentioned at the beginning of the project period.

The chosen direction for this thesis is **handover optimization** because its demand is widely increasing after the development happening in the world in many aspects. Almost everyone has a mobile phone and always needs to travel or move between different areas. Sometimes they can suffer from lousy signal quality or data delivery delays while they move, which was an important reason to work on this project. To make the optimization easier and reachable for researchers, a good network simulator needs to be used, and this simulator is ns-3, which is based on C++. Network setup can easily be programmed using it.

Reinforcement Learning has been used in many systems and has shown good performance and optimizations. It has also been used in mobile networks and has shown many positive results. Some of the work is presented in the related work chapter 3. It has also been used for handover optimization especially and improved its performance in an obvious way. What was interesting is that many RL algorithms have been tested, like Deep Q Learning (DQL) in the related paper 3.4, Deep Deterministic Policy Gradient (DDPG) in the related paper 3.2, and other algorithms. Additionally, multiple RL designs have been used like one general RL agent in the related paper 3.4 or multi-agents for each UE in the related paper 3.6.

The highest motivation is finding an RL algorithm for a continuous environment that has shown promising results in other applications and is not used on many mobile network applications. This algorithm is Soft-Actor-Critic (SAC) which is explained in section 2.1.2. To validate the performance of the SAC algorithm, a comparison with another RL algorithm was needed, and the chosen algorithm is Twin Delayed Deep Deterministic Policy Gradient (TD3) which was used in the related paper 3.11 for the continuous action space part and has shown good performance.

The main two Handover algorithms in ns-3 (A2A4-RSRQ, and A3-RSRP) are the algorithms used in this thesis. The performance of both two has been combined and optimized without using RL in [41]. Many experiments use the two algorithms to optimize their attribute values and overall handover process performance. That includes improving the total data throughput, signal power, and quality all users receive.

2.7 Summary

This chapter overviews this work's central parts, including a brief introduction to reinforcement learning and its algorithms, especially the RL algorithms used for continuous environment spaces situations. These algorithms are used in many applications, including the related work papers summarized in chapter 3. SAC and TD3 are the RL algorithms used and compared in this project. The mobile network is also described briefly. A general overview of the ns-3 simulator, which is mainly used in this work instead of the real-world networks, is explained. Since the handover process is the main focus, it has been explained with its algorithms, parameters, and attributes in detail.

Chapter 3

Related Work

Some related scientific papers which have used different optimization methods, RL algorithms, and handover algorithms are introduced in this chapter. Many papers use the ns-3 simulator to create the simulated networks. The first paper explains the system implemented by the Telenor research team, which is the foundation work for this project. The second paper is a new system implemented using RL for mobile network optimization. The third, fourth, fifth, sixth, eighth, and eleventh papers focus on optimizing the handover algorithm using different RL algorithms or other machine learning methods. The optimization method itself is also different between these papers.

The seventh paper is about finding another algorithm to optimize handover. The ninth and tenth papers explain two frameworks implemented as a bridge between the ns-3 simulator and RL in Python and facilitate their interactions. The eleventh paper compares the RL algorithms used for discrete and continuous action spaces in cellular network optimization. The last two papers show the usage of the SAC algorithm in a cellular network application and another environment with continuous state and action spaces.

3.1 FORLORN: A Framework for Comparing Offline Methods and Reinforcement Learning for Optimization of RAN Parameters [1]

Telenor Research team had worked on improving network resource usage using RL and the ns-3 simulator. They have focused on using RL to adjust network parameters in eNBS (base stations) to improve the quality of the signals and avoid any delay in data sending. Their system is called "Framework for Comparing Offline Methods and Reinforcement Learning for Optimization of RAN Parameters" (FORLORN). This system aims to continuously improve the parameters and update the network to ensure good quality. Their system is written in both C++ (especially for ns-3) and Python (The RL agent, parsing functions, and visualization). Where the interaction between these two is happening through the text-based protocol. The data produced in ns-3 is saved in a text file, and the Python code parses this data. Optuna, which is an optimization framework, has been used to choose the RL algorithm and its hyperparameters and to guide the RL agent while optimizing the RAN parameters. The results of a network scenario for each training step are stored in an SQLite database to compare the performance each time. During the network setup, three eNBS have been established in ns-3, and having a good load balance between them was essential. The download throughput is considered to be an evaluation value for the performance.

RL agent follows Actor-Critic with neural network algorithm. The action space is considered to be discrete to represent transmission power increasing and decreasing. The states are current transmission power, number of UEs, and RSRQs for each UE. A "user experience score" value is produced from ns-3 and represents the sum of all UE experiences of throughput. This value is involved in RL reward calculation. After 100 evaluation trials, the RL agent converges to distribute the UEs among eNBS evenly and chooses the power level based on that. Then with more offline trials, the agent showed that it's adapting the results during the changes happening in the network. Even though this design has improved, more opportunities for further improvements in the system, environment, and RL design are discussed.

3.2 ICRAN: Intelligent Control for Self-Driving RAN Based on Deep Reinforcement Learning [2]

This paper focuses on implanting Intelligent Control for Self-driving radio access networks (ICRAN) using deep reinforcement learning (DRL) to maximize resource utilization and minimize service level agreement (SLA) violations under different network conditions. This work has been done using the ns-3 simulator and OpenAI Gym framework explained in section 3.9. The performance and test were on 4G mobile networks, but the aim is to get it to work on 5G with some adjustments because the 5G network is more complex to control. The environment includes a number BSs and their connected users. Users are collected into three types based on their service requirements. These types are called slices. The evaluation occurred in four stages with two scenarios (Network Congestion and Network Failure) and different network conditions. 1) To check convergence performance during the training. 2) Validate the decisions taken by the agent. 3) Compare ICRAN to other state-of-the-art methods. 4) Study the throughput and delay generally under different network load levels. ICRAN utilizes the available radio resources 7% higher than the next best system.

ICRAN involves two designs for DRL, which are centralized and decentralized implementations. Both are implemented based on deep deterministic policy gradient (DDPG) mentioned in section 2.1.2 to handle continuous action space and find the optimal policy by increasing the antenna coverage, distributing the traffic loads, or adjusting the data rate for a specific slice. The state space includes the antenna down-tilt and average throughput for the eNB. At the same time, the action space includes antenna tilt optimization, traffic load balancing, and traffic shaping. The reward is calculated depending on the quality of service (QoS) and the network efficiency and includes end-to-end delay and throughput for UE.

Centralized design (ICRAN-C) has a single RL agent taking the responsibility for all eNBS and UEs connected to them. This agent can observe and change all the network information. While the **Decentralized** design (ICRAN-D) includes multiple agents for each eNB and its following UEs. The decision is taken individually in each eNB, but each agent has some information about its neighbor eNBS to avoid conflicting strategies between the agents. Some challenges occur with a multi-agent design, like the exponentially increasing complexity of the state and action spaces and the general instability caused by the effect on other agents' policies issued by the change of the policy in one agent during the training. Therefore, Multi-agent DDPG (MADDPG) is used where the agents share a centralized critic. A centralized solution can show better performance. Still, it uses more time to find the optimal solution because the total search space increases as the size of the network increases.

3.3 Deep Learning based Adaptive Handover Optimization for Ultra-Dense 5G Mobile Networks [3]

This paper shows an experiment using supervised learning with GRU recurrent neural network in handover optimization and changing the environment as quickly as possible. The purpose is to reduce the amount of service traffic transmitted through the communication channels and increase the amount of helpful information transmitted. Recurrent neural networks connect the output information from the previous steps to the current and future ones. In this paper, the performance of LSTM has been combined with GRU. Both can filter the information in the cell based on some conditions using three filters. However, GRU training time is lower since it uses less training data and can solve the problem of gradient disappearance by determining how much data from the past passed to the future.

The experiment aims to predict the number of users in a particular cell at each time of a day or several days. The data collected for the experiment is from the user's mobility over two months at intervals of 5 minutes. The test data is about the subscribers for five days at intervals of 5 minutes. The results showed that GRU sometimes produces better results with a minimum delay, but LSTM improves when the number of learning epochs increases. Users of the networks are in different places, and their location is always changeable. It will be impossible to use constant offsets for the handover. This paper focuses on Mobility Load Balancing (MLB) offsets and Mobility Robustness Optimization (MRO). When the cell is overloaded, the neural network will use MLB by considering a coefficient K close to 80-90%. Thus, the power shifts to the side of this cell, reducing its size and making the handover from it difficult. So it will be more accessible from the second cell. MRO is used instead in the not-overloaded cells.

3.4 Intelligent Handover Algorithm for Vehicle-to-Network Communications With Double-Deep Q-Learning [4]

This paper discusses finding optimal handover (HO) decision-making for high-quality infotainment services by using DRL with configurable parameters as inputs since more frequent handover between BSs is needed in high-mobility vehicles. The handover algorithm is A3 with Reference signal received power (RSRP) implemented in the ns-3 simulator. The algorithm measures throughput, packet loss, errors, and latency at a network level because the technological factors of quality of services (QoS), such as reliability, scalability, and network congestion, are based on these values. It uses RSRQ and RSRP values since they are the most important values that affect the handover decision. The algorithm is developed based on the Double deep Q-Learning network (DDQN) because the environment is highly complex with huge state space or high-dimensional state inputs. The algorithm has two phases, exploration (training) and exploitation (execution), to get more stability potential during the training since the neural network approximation will be only updated with each training iteration. So, the value of the future states will not be affected. A fully connected feed-forward ANN with three hidden layers is used.

The state is a combination of a vector of RSRP values measured by all BSs and the serving BS ID. They are in the same order as the RSRP vector. The action space is all listed BSs in the local area, including the serving BS. The reward is calculated based on normalizing the value of the RSRP of the target BS with the highest reported value to emphasize the RSRP difference between the current BS choice and the local maximum RSRP. If a handover decision is made, a punishment factor is subtracted from the reward. A replay buffer that stores tuples of (state, action, reward, next state) is used to avoid discarding their values after using them once during the experience. To evaluate the performance of the HO algorithm, the time needed by DDQN HO algorithm to make the HO decision is combined towards the optimal HO instant. As a result, the DDQN-based HO improves the triggering instant compared to the A3 RSRP baseline. It learned the RSRP features along the whole trajectory and performed an additional HO to improve optimal signal strength.

3.5 A Parameter Optimization Method for LTE-R Handover Based on Reinforcement Learning [5]

This purpose is to find an adaptive optimization method based on the Q-Learning with the Monte Carlo method to achieve a real-time estimation of the handover attributes of the Long Term Evolution-Railway (LTE-R) communication system since it has low-delay network services. The best handover performance is the goal. Different performance situations for handover attributes with different speeds have been used. Mobile users were provided a basis for accessing opportunistic channels during the handover process. RL has been chosen to solve the highly dynamic feature of handover attribute selection in the LTE-R system. The characteristics of high-speed train movement make the effect of handover attributes on handover performance even more significant.

The state set of the environment is the combination of handover attributes (Threshold and offset in A2A4 HO algorithm, TTT, and Hysteresis for A3) of the UE at a certain speed. At the same time, the action set is the set of parameters chosen by the UE for handover at this speed. The handover success rate in [0, 1] is the reward. The speed of the user passing a particular base station is constant. The experiment has been performed by building up a high-speed railway mobile wireless communication network architecture in ns-3 to compare the UE handover success rate and average throughput. The impact of the changes in UE speed on the handover situation is observed. The simulation experiment is divided into two parts 1) Verifying the rationality of the algorithm by comparing the situation maps formed by the algorithms in this paper. 2) Comparing the optimal scheme given in the continuous handover situation. The simulation results show that the optimized handover attributes can significantly improve the handover performance.

3.6 Joint Optimization of Handover Control and Power Allocation Based on Multi-Agent Deep Reinforcement Learning [6]

This paper focuses on the handover (HO) and power allocation problem in a two-tier heterogeneous network (HetNet), which consists of a macro base station and some millimeter-wave small base stations, to maximize the throughput and reduce the HO frequency. The proximal policy optimization method solves the problem using a multi-agent reinforcement learning design. Each UE trains its policy with centralized training, then obtains a decentralized policy after finishing the training based on its observations which include the BS connection, signal measurement, and the number of UEs served by each BS.

The number of UEs served by each BS is public information, and each UE decides its action for BS selection and power requirement. In this situation, the action space grows exponentially with the number of UEs, making the learning process and finding an optimal global policy difficult. Each UE learns with its actor-critic to achieve independent proximal policy optimization. The centralized critic estimates the joint value function Q based on global information, and the decentralized actor only makes decisions based on local observations. Due to the lack of global state and joint action, the advantage function for each UE can only be estimated based on the shared reward and each UE's local information by comparing the Q-value estimated by the critic for the executed action. The total reward is a submission of all individual rewards for each agent. Each reward is found based on throughput, interpretation time, and HO penalty if the handover is triggered (this value can be adjusted to relieve the HO frequently). All agents share the same reward function, and it is found based on individual observations.

3.7 Frequent-Handover Mitigation in Ultra-Dense Heterogeneous Networks [7]

This paper focuses on implementing an algorithm to solve the frequent-handover mitigation (FHM) problem for the ultra-dense heterogeneous network (HetNets) and avoid packet losses or delays during handovers. This problem is based on a high-speed frequent handover experience (FHE) which involves fast-moving and slow-moving users. Both user types can experience unwanted handovers where the handovers for users moving fast happen between small cells in small coverage areas, and slow-moving users experience unwanted handovers for the same cell (this handover frequently is called ping-pongs). Therefore, the main goal of this algorithm is to reduce the handover frequency.

FHM algorithm splits users into two categories: fast-moving with low estimated dwelling times are then transferred to the macro layer, and ping-pongs with a pattern in the serving cell history with low dwelling times are managed by adjusting the user-specific handover parameters. If that is not enough and has already happened for the user, it will be handed over to the macro layer. The algorithm uses handover history information and measurement reports, including RSRP delivered from the users to identify their category and find the estimated dwelling time. The handover algorithm used for this algorithm and its experiments is A3. The environment used for the experiments is simulated using the ns-3 simulator. After applying the experiments, the results show that the overall handovers are reduced by 79.56%, and the network throughput is increased by 10.82%.

3.8 5G Handover using Reinforcement Learning [8]

This work aims to optimize handover in 5G cellular networks using a centralized RL agent to improve mobility and throughput performance. The RL agent is responsible for analyzing the measurement reports received from the UEs to control HO between BSs and maximize the utility. In dense 5G deployments, the connected state UE can receive access beams (the beams used for cell acquisition) from multiple BSs with sufficient power to perform the initial entry procedure. The focus is to produce a technique for connected-state intra-frequency HOs in the 5G context. When a UE is in a connected mode, the BS allocates resources to it, and there will be active signaling on the data and control channels.

The RL agent is designed based on the contextual multi-arm bandit (CMAB) problem where the actions taken by the agent do not modify the environment. The CMAB agent applies handover for a UE to maximize its average link-beam gain and throughput. In this system, the serving BS is not responsible for the handover decision. Its mission is to forward the measurement reports to the CMAB agent. The agent decides the action, and the optimal policy is found using Q-Learning. Each BS can be considered as an arm in the CMAB problem. The best arm is the current BS or the appropriate BS chosen to make the HO based on maximizing the reward, which is the RSRP of the link beam after HO. Some challenges can face this implementation, like a state-space explosion. Therefore, a Q-Table is built with representative states during the training and with a suitable choice of similarity function to find the closest state during the active phase for deciding HO. The performance results are analyzed in three distinct environment setups. They are based on the synthetic data generated from a system emulator with different configurations of access and link beams. The measurement in each step is sent to the CMAB agent, which exploits the Q-Table to evaluate the possibility of the HO decision. In all these environments, this system performs better than the existing state-of-the-art algorithms.

3.9 ns3-gym: Extending OpenAI Gym for Networking Research [9]

NS3-Gym is a toolkit that simplifies the integration between RL and ns-3 simulator to solve and optimize network and communication problems ranging from scheduling, resource management, congestion control, routing, and adaptive video streaming. It's based on OpenAI Gym [48], which is written in Python and can be used in multiple applications to control the RL agent learning process and react with the environment. The environment in ns3-gym is created in the ns-3 simulator. So ns3-gym is an interface between OpenAI Gym and ns-3. This interface manages the ns-3 simulation process life cycle and transfers state and action information between the Gym agent and the simulation environment. It has some design principles: scalability, low entry overhead, fast prototyping, and easy maintenance. State and action spaces supported types in Python are discrete value, vector, Tuple, or Dict.

While running every step, the ns3-gym observes the current state by calling the following functions: GetObservation, GetReward, GetGameOver, and GetExtraInfo. For actions, the user needs to implement the execution function called ExecuteActions, where the action is represented numerically. The complexity of ns3-gym implementation and working steps is hidden behind, and the user can easily use the finished functions. It's also possible for the user to create a predefined observation state, action, and reward function in some custom environments if needed. The communication between RL and ns-3 are realized over ZMQ sockets using the Protocol Buffers library to serialize messages. This framework aims to reduce the time needed to prototype RL-based networking solutions.

3.10 ns3-ai: Fostering Artificial Intelligence Algorithms for Networking Research [10]

NS3-AI (like ns-3 gym) is designed to connect ns-3, written in C++, and multiple AI frameworks implemented using Python. Since applying AI algorithms on real-world networks produces a massive amount of data to be collected and controlled and makes it challenging to test and validate the results produced by these algorithms on real-world systems, the ns-3 simulator is a crucial part to be included when using AI algorithms to improve network performance. This module is designed to facilitate exchanging data with high speed between ns-3 and AI algorithms because network complexity increases in the new generations of the network (like 5G), and more data must be fed into the AI algorithms. NS3-AI can be easily adapted to different projects because it provides a faster interface in Python and C++.

NS3-AI differs from ns3-gym because it can be used with both Deep Learning and RL, while ns3-gym applies only to RL since it depends on OpenAIGym. In addition, its architecture is based on a shared memory structure between the ns-3 environment and a Python-based AI algorithm. The memory is implemented in ns-3 using C++. Therefore, it's controlled in ns-3 mainly, but it's accessible for the AI model. It has three main parts: the memory blocks, the control blocks, and the main control block. Memory structure uses a pooling mechanism to avoid time synchronization and conflict between several processes performed. This solution supports transferring vast amounts of data from the environment to an AI algorithm, modifies different data types quickly, and reduces the transmission time. However, this is more complex in ns3-gym because it uses ZMQ sockets for data transferring. The data transferred to the AI module is generated by the ns-3 simulator with information about changes happening in the environment. The shared memory delivers This data to the AI model and is then used for training and testing. Channel quality indicator (CQI) using Long Short-Term Memory (LSTM) is an application of ns3-ai to show how good this toolkit is. The results of testing ns3-ai showed that it's 50 to 100 times faster than ns3-gym while ns3-gym performance is stable.

3.11 Self-Optimization of Cellular Networks Using Deep Reinforcement Learning with Hybrid Action Space [11]

This system aims to increase the self-managed cellular network stability and adaptability to rapid or sudden changes in the environment. The system is RL-based, and its environment is implemented using the ns-3 simulator and ns3-gym as the interface between Python and C++. It works toward achieving five main targets: maximizing the sum throughput of the network, achieving load balancing among cells, minimizing the number of blocked users out of coverage, satisfying the Quality of Experience (QoE) for users, and minimizing the energy consumption. These are achieved by optimizing the individual cell offset (CIO) values between eNBS and the eNB transmission power and enabling the MIMO capability (multiple inputs and multiple outputs) of the eNBs. This solution is the first done on a wide area of optimizing multiple features simultaneously. Each eNB sends its transmission in the Down-Link (DL) with a power level, and each UE measures the Signal to- Interference-plus-Noise-Ratio (SINR) of near eNBS and sends the value to find the highest cell.

The RL system focuses on dealing with hybrid action space, a combination of continuous and discrete actions. The discrete actions turn MIMO in each eNB on and off, and the continuous ones are transmission power for each eNB and CIO controlling between every two neighboring cells. Therefore, two main RL algorithms have been used to design the system. **Double Deep Q-Network (DDQN)** for discrete action spaces and **Twin Delayed Deep Deterministic Policy Gradient (TD3)**, which is explained in section 2.1.2, for continuous action spaces. The decision is taken in two stages. First, the agent decides on the discrete action, then the continuous action, and finally, the action is applied. The state space includes subsets of the network KPIs: Resource Block Utilization (RBU), Total DL throughput of each cell, Number of active users in each cell, and Modulation and Coding Scheme (MCS) Matrix. The reward is computed based on a linear combination of the total network throughput, the throughput sum of blocked scaled by an adjustable hyperparameter, and the number of eNBS that have the MIMO feature turned on, which is also scaled by another adjustable hyperparameter. Two environment scenarios are evaluated, and the results show that this algorithm achieves higher sum throughput than the baseline system without RL control and other previously proposed algorithms.

3.12 Cooperative Resource Allocation based on Soft Actor-Critic with Data Augmentation in Cellular Network [12]

This paper explains the usage of the soft actor-critic (SAC) RL algorithm to improve the resource allocation problem of cellular networks with simultaneous wireless information and power transfer (SWIPT). Data augmentation with random and adaptive schemes has been used for the experience replay buffer of SAC to improve the training speed and produce more datasets with low complexity since filling the whole replay buffer with agent-environment interaction data requires more time. The random scheme includes randomly generating a mini-batch of the current state, action, reward, and following state tuples called experience tuples. The mini-batch collection will terminate when the replay buffer reaches its upper limit. In contrast, the adaptive scheme includes experience tuples generated based on permutation equivalence and data augmentation.

The purpose is to maximize the total data rate and system fairness and minimize the channel switching penalty in each time step. This task of resource allocation is continuous with continuous spaces. Users of cellular networks and their wireless needs are increasing, and the number of radio resources supporting them is limited. Therefore, the need for more optimized recourse allocation between users is also increasing.

The action space includes the power allocated by each user and adequate communication time. The states include the channel occupying state, channel gain, and the remaining energy of each user. The SAC algorithm consists of two main critic networks, two target critic networks, and one actor-network. The adaptive scheme performs better than basic SAC, especially regarding the data rate and system fairness, but the switching penalties are increased. The cumulative discount reward, the maximum submission of all rewards in all time steps for each action, shows that the SAC algorithm assisted by data augmentation performs better than the baseline in learning speed.

3.13 Soft Actor-Critic for Navigation of Mobile Robots [13]

This paper compares the performance of soft actor-critic (SAC) and Deep Deterministic Policy Gradient (DDPG) RL algorithms in mobile robot navigation through continuous control. The robot moves from a starting position toward a target position inside a virtual environment. Two different environments have been used for the agent to move in. The first environment has a U shape, representing the symmetric scenario, and the second one is asymmetric and more complicated to encourage the agent to build a wiser policy. The structure of both RL algorithms includes one actor network and one critic network. The reward is negative when the agent collides with any object and positive when it arrives at the target.

After the training, the SAC algorithm converges to an optimal reward faster and shows more consistency than the DDPG algorithm in both environments. The DDPG algorithm showed some instability in reward after some training episodes. The SAC performance seems to have a very clean trajectory and accuracy to achieve the target. That improves the performance of the SAC network, but both algorithms are suitable for these applications.

3.14 Summary

These papers include relevant research and techniques used and evaluated to optimize and solve relevant cases. The first paper shows the work done by the Telenor research team to optimize the mobile network performance. Their work is used as the initial step in this project, especially the first implementation of the handover simulated network in ns-3. They hadn't used RL to optimize the handover algorithm, but this work provided a general overview of using RL in mobile networks and implementing different simulated networks in ns-3. In addition, the second and twelfth papers focus on optimizing the mobile networks' performance in other directions. The problems introduced in these papers have continuous environments. They explain the usage of different RL techniques for optimization purposes.

The second paper uses RL to maximize resource utilization and minimize SLA violations. Two RL setups are implemented. The first one is a general setup that includes one RL agent for many BSs and their connected UEs, while the other one includes an agent for each BS and its connected UEs. On the other hand, the sixth paper, which discusses the optimization of handover and power allocation performance, shows a different RL setup where the agent has a UE perspective and every agent is responsible for one UE. These setups have been evaluated in this project, and the first general setup has been chosen to avoid the policy overlapping between different agents and find the optimal handover attribute values for all BSs by considering all actions happening in the network. The measurements are produced by the UE and received by the BS, but the handover decision is taken and applied by the BS. Therefore, the RL agent has a BS perspective.

The fourth, fifth, and eighth paper shows the usage of different RL algorithm to optimize the handover algorithms. The A3 handover algorithm is used in the fourth paper, while both handover algorithms are used in the fifth paper. These papers guided the building of the state and actions spaces used for both handover algorithms in this project. The reward functions used in these papers have been evaluated, too, since they include a factor to optimize the number of handovers and the signal quality and power received by the UE. The seventh paper provided a deeper understanding of the handover algorithms, especially the A3 algorithm, the ns-3 simulator, and the methods to reduce the number of handovers. The third paper showed the usage of RNN to optimize handover performance. The ninth and tenth papers are implemented tools to bridge the Python RL environment and the C++ ns-3 simulator. These toolkits are evaluated to be used in this project, but they are not used since OpenAI gym is not used, and the communication between the simulated network in ns-3 and the RL environment is implemented differently.

The Soft actor-critic algorithm has been used in the last two papers. The twelfth paper uses the SAC algorithm to optimize the resource allocation problem in mobile networks. It compares the performances of different SAC implementations used in the same problem. In contrast, the last paper compares the performance of SAC and DDPG in a problem with a continuous environment and shows that SAC could achieve better performance. The success achieved by SAC in different applications with continuous environments and the lack of the number of research that applies SAC in the handover problem precisely and in the mobile network generally were the main reasons to apply it in this project and explore and evaluate its performance. The TD3 algorithm has been used in the eleventh paper for the continuous actions part and showed promising results. It uses a deterministic policy while SAC uses a stochastic policy. Therefore it has been chosen to be compared with the SAC algorithm. DDPG has not been chosen because its performance has been compared with SAC in other works unrelated to mobile networks. However, the comparison between DDPG and SAC in mobile networks can be a future work.

Chapter 4

Methodology

The implemented solution for the two handover algorithms will be discussed in this chapter. First, the existing simulation for the A2A4 handover algorithm implemented by the Telenor research team will be explained. The new simulations, the reinforcement learning environment structure for each handover algorithm, and the RL agent design will be described using the SAC and the TD3 RL algorithms and their components. The grid search experiments, which are conducted to evaluate the performance of the RL agent, are introduced too.

4.1 Existing Work

The Telenor research team has worked on the FORLORN project explained in the first related work in 3.1. This project aims to optimize the RAN parameters using the actor-critic algorithm in reinforcement learning. Their code is included in this GitHub repository [49].

Their work doesn't include applying reinforcement learning on the handover algorithm to optimize its performance, but they have implemented a simulated scenario in ns-3 for the A2A4 RSRQ handover algorithm with nine base stations (BSs) and three user devices (UEs) moving randomly. This scenario is visualized in Figure 4.1 where the triangles from 1 to 9 are the base stations, and the three white points are users moving around. It is implemented using the A2A4 RSRQ handover algorithm from ns-3 without changing the scenario parameters like the number of users and their maximum and minimum speeds and initial positions. The handover attributes are also constant without any ability to change them. This scenario happens over 60 s, and RSRQ, RSRP, and throughput values for each user are measured by parsing the ns-3 values received in the simulator code written in C++. They are saved in a text file, and this file is parsed and visualized in the Python code. The Python parser and visualizer to generate the Figure 4.1 was implemented.

Each RSRQ plot presented in Figure 4.1 is the RSRQ value received from each base station and measured for each user during the whole simulation session. The A2A4 handover algorithm actively scans the signal quality the user receives from different base stations and picks the best base station to trigger the handover. Therefore the RSRQ is measured from different base stations even though the user is not handed over to these base stations. The red lines in the RSRQ plots are shown when the handover was triggered to a new base station. The throughput plots to the right in Figure 4.1 show the data throughput measured for each user. The data throughput is the number of bytes the user device receives each time step.

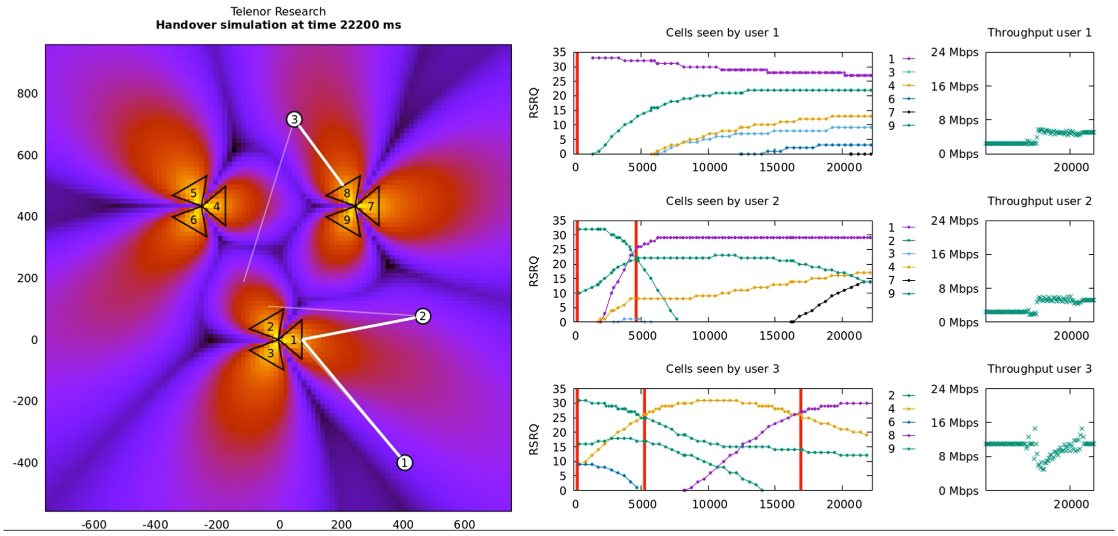


Figure 4.1: The simulated scenario implemented in C++ was visualized using Python with the returned values in each time step. The triangles in the radio map are the base stations, and the yellow to purple colors around them are a reference for the signal power. The white points are the users moving around. The plots to the right are RSRQ and throughput plots for each user over time

4.1.1 Applied changes in this work

The simulated network built by the Telenor research team has been used as a foundation for the simulated network in this project's RL environment. It is updated to be a dynamic network that takes the network parameters as input and builds the simulated network based on them. The network parameters that affect the network structure become adjustable, and they are decided and sent from the RL environment in each simulator call. The measurements produced by the simulator are collected and sorted to fit the needs for generating the RL environmental observations.

The adjustable network parameters are the simulation duration, the number of base stations (ENBs), the number of users (UEs), their maximum and minimum speeds, their initial positions, the random number generator (RNG), and the handover attribute values (ServingCellThreshold and NeighbourCellOffset for A2A4 RSRQ, and TimeToTrigger and Hysteresis for A3 RSRP). All these parameters are sent with the simulator cmd call, parsed in the simulator, and used to build the simulated network.

The RL system, including the RL environment and the RL models, is built independently from the RL system built by the Telenor research team. That is because the RL models used in this project differ from those used in the FORLORN work and use other RL algorithms.

4.2 Reinforcement Learning Environment Structure

The environment leads the interaction between the agent and the simulator by deciding the network setup parameters values, generating the observations (states), applying the actions received from the RL agent, and computing the rewards. The general view of the interaction between the environment and the simulator is shown in Figure 4.2. The environment passes the network parameters and the handover attribute values to the simulator to produce the desired simulated network in each step. After receiving the parameter values, the ns-3 simulator starts the simulation for the chosen duration. The simulator runs the whole simulation in each step (episode) and logs the produced outputs to the log file. The outputs include information about each UE while connected to a BS. Finally and after finishing the episode, the environment reads the whole log file produced by the simulator after finishing the simulation duration and generates its observations.

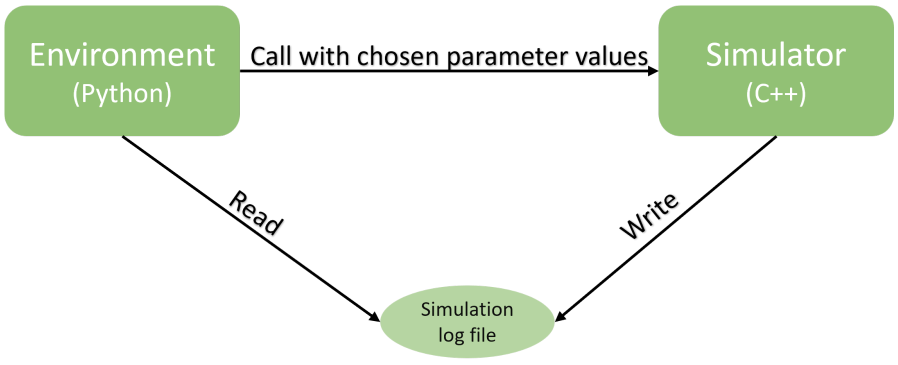


Figure 4.2: A general view of the interaction between the RL environment implemented in Python and the ns-3 simulator which builds the simulated networks

4.2.1 Simulated networks

A simulated network scenario includes all about network setups like the number of BSs and UEs, their locations, UEs movement speeds, the values of the simulation result values reported between UEs and BSs, the handover algorithm type, and the handover attribute values used in the handover algorithm.

These are the scenario parameters that are modified and decided in the RL environment and used by the simulator to build the simulated network:

- **Simulation duration:** The length of the whole simulation in seconds
- **Number of users:** The number of moving users in the simulated area during the simulation.
- **Number of base stations:** The number of base stations in the simulation area.
- **Users maximum speed:** The upper limit of users' moving speed.
- **Users minimum speed:** The lower limit of users' moving speed.

- **Users start positions:** x and y points represent the users' positions when the simulation starts.
- **Attribute values:** The values of handover attributes which are Hysteresis and Time-ToTrigger for The A3 RSRP algorithm, and ServingCellThreshold and NeighbourCellOffset for the A2A4 RSRQ algorithm.

The mobile network is simulated using the open-source ns-3 simulator. As explained in 2.3 ns-3 is built using C++. The 'Lte helper' object, which is responsible for managing LTE factors like base stations (eNodeBs), users (UEs nodes), the handover algorithm, and other network properties, is used to build the network. [50]

During the project, multiple network scenarios with different parameter values were tested to analyze their effect on the results. Some parameters continuously change during the RL training phase to evaluate the performance in different network situations. These parameters are the simulation duration, the number of users, and user maximum and minimum speeds and start positions. At the same time, other parameters are constant during the training period, like the number of BSs and their positions.

To randomize the user movements and speeds during the simulation, a random value of the Random number generator (RNG) is also passed to the simulator from the environment. Without feeding this value to the simulator and if the other parameter values are the same, the same simulated network with the same moving positions of all users will be produced since ns-3 uses a deterministic seed with a fixed number each time [51].

The RNG value is changed in every step to generate random and unique moving directions and speeds for users. This will make the agent see the same parameter values (like the number of users, speed, and simulation duration) from different perspectives and evaluate the action to be generally and correctly chosen to fit a specific network setup.

Each step will involve a unique and independent simulated network because of the combination of all parameter values besides the randomized RNG. This means the number of users can vary, and their movement will be randomly directed between all base stations. Thus, they will go through different signal quality situations during the simulation. That allows the agent to experience different network situations and suggest the best action values.

4.2.2 The observation space

The observation space includes status information about the simulated world and produced results after applying the actions. It shows how the users' devices reacted during the simulation and how good the signal quality, signal power, and throughput they received were.

Feature Engineering

Under the simulation, a detailed log about the user positioning, signal power, signal quality, and data throughput for each UE in each time step is generated. The information is saved as row data and needs to be re-structured to create valuable observations for the RL agent. The dimensions of the observations need to be static in all training steps. Thus, the observation space is created to be a flattened list of smaller lists that represent information about each base station besides other values that generally represent the simulation status. All observation values are normalized.

The observations are divided into two main parts. The first is a flattened list of base station lists, each including salient values about the base station and its connected UEs. The other part is some general values that give general information about the changeable parameters in the environment setup in each step. A general overview of the structure of the observations is shown in Figure 4.3

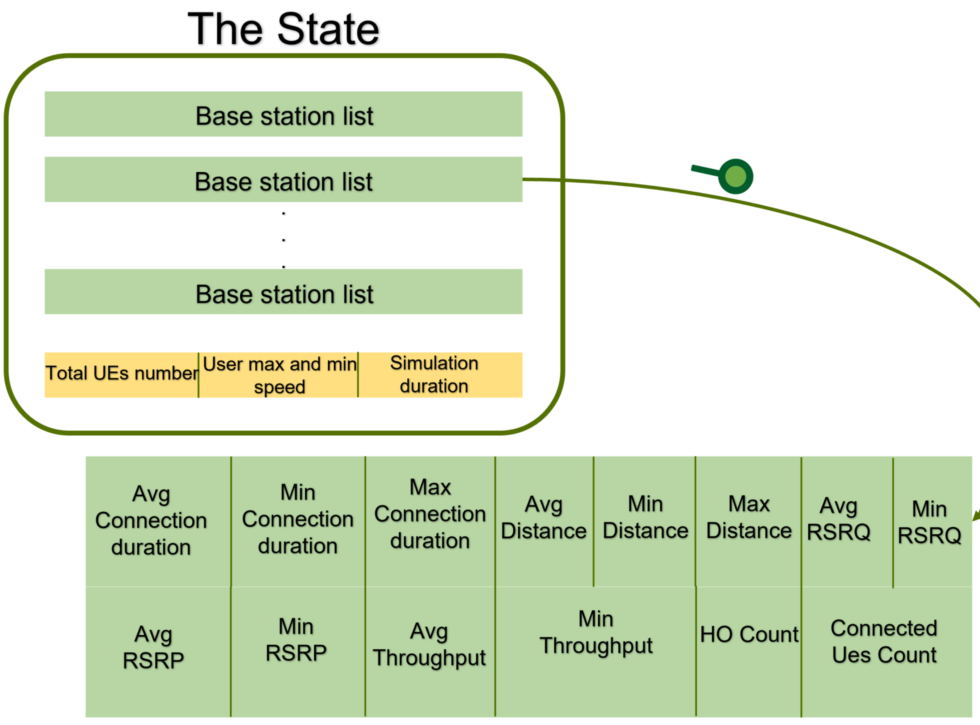


Figure 4.3: An overview of the state produced after each simulation. The observations will then be a flattened list of these presented lists

Base station lists: A smaller list includes some informative values for each base station which represent the relations between this base station and its connected UEs during the simulation. The main focus is the base stations because their count is constant over all training steps, but the number of users is changeable. As mentioned, The size of the observations needs to be the same for all training steps. Therefore, each base station has a list of these informative values. These values are:

- **Connection duration:** This value is based on the connection period between each BS and UE. It starts when a user is handed over to a specific base station and ends, after a while, when a user has been handed over to another base station. If this user is connected to the same base station multiple times, all connection duration values will be added together. All duration values for all users connected to a specific base station are collected, and the values added to the state space are:
  - **The Average of Connection duration** is the average of all duration values for each base station.

  - **The minimum Connection duration** is the minimum value.
  - **The maximum Connection duration** is the maximum value.
- **Distances between base stations and their connected users** The distances between each base station and its connected devices are found every 100 milliseconds. Since the users move around during the simulation period and they don't have a constant position while connected to a particular base station, how far they are to the base station can affect the signal power, signal quality, and throughput they receive. For example, if the distance is very big, the throughput in the user device can be low. The distances are found based on the equation 4.1.

  $$distance = \sqrt{(x_b - x_u)^2 + (y_b - y_u)^2} \tag{4.1}$$

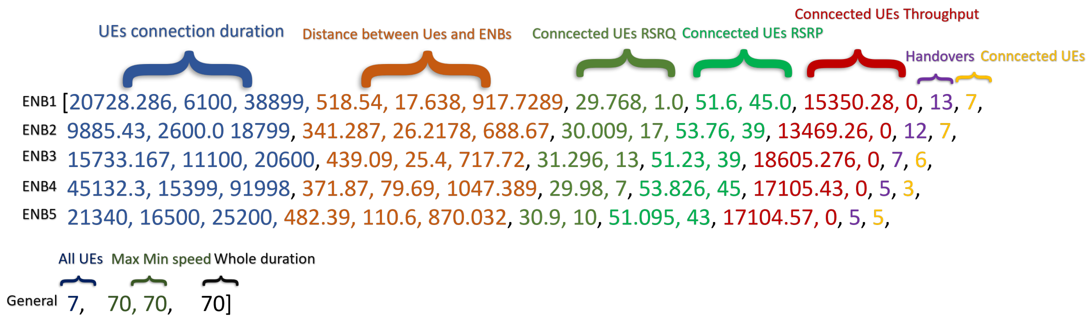
  Where x_b and y_b are the x and y coordinates of the base station's position. x_u and y_u are the x and y coordinates for the user's position.

  The values added to the state space are:
  - **The Average of user distances** is the average of all distance values for each base station.
  - **The minimum distances** is the nearest position a UE had while connected to this base station.
  - **The maximum distances** is the farthest position a UE had while it was connected to this base station.
- **Throughput value** is the number of bytes received per 100 milliseconds by each user device. Throughput values for all connected users are collected, and the values chosen to be added to the state space for each cell are:
  - **The Average throughput** is the average of all throughput values for each base station.
  - **The minimum throughput**
- **RSRQ value** is Reference Signal Received Quality explained in section 2.4. It is also saved for all connected user devices for every time step. This value shows the agent how good the signal quality was while a particular UE was connected to this base station.
  - **The Average RSRQ** is the average of all RSRQ values for each base station.
  - **The minimum RSRQ** is the minimum value saved to show the agent the lowest signal quality received for this base station.
- **RSRP value** is Reference Signal Received Power explained in section 2.4. Like the RSRQ, it is saved for every time step for all connected UEs.
  - **The Average RSRP**
  - **The minimum RSRP**
- **Number of handovers** This value represents the total number of handovers in every base station during the simulation, which means how many times a UE has been handed over to this base station. If the same UE was connected to the same base station multiple times, each time would be counted as an independent handover.
- **Number of connected UEs** This value represents the number of UEs which were connected to the base station during the simulation. A UE is counted once if it was connected once or multiple times to the same base station during the simulation episode.

General informative values: They are some other values added to the state space about the simulation environment, in general, to make the agent learn about the best solution for each network setup. These values are:

- **The total number of UEs** which are users moving around during the whole simulation.
- **The user's maximum and minimum speeds** This means the upper and lower limits of the moving speed of the users.
- **The simulation duration** in milliseconds.

An example of the state before normalization from a training step in an experiment is shown in Figure 4.4. These values are produced after calling the simulator and finishing the simulation duration in this particular step. The Base station lists are displayed in every line as ENB list. The last line in the figure shows the general informative values. The normalized version of the state in Figure 4.4, which is used as a real environment observation, is shown in Figure 4.5


UEs connection duration
Distance between Ues and ENBs
Conncected UEs RSRQ
Conncected UEs RSRP
Conncected UEs Throughput
Handovers
Conncected UEs
All UEs
Max Min speed
Whole duration

```
ENB1 [20728.286, 6100, 38899, 518.54, 17.638, 917.7289, 29.768, 1.0, 51.6, 45.0, 15350.28, 0, 13, 7,
ENB2  9885.43, 2600.0 18799, 341.287, 26.2178, 688.67, 30.009, 17, 53.76, 39, 13469.26, 0, 12, 7,
ENB3  15733.167, 11100, 20600, 439.09, 25.4, 717.72, 31.296, 13, 51.23, 39, 18605.276, 0, 7, 6,
ENB4  45132.3, 15399, 91998, 371.87, 79.69, 1047.389, 29.98, 7, 53.826, 45, 17105.43, 0, 5, 3,
ENB5  21340, 16500, 25200, 482.39, 110.6, 870.032, 30.9, 10, 51.095, 43, 17104.57, 0, 5, 5,

General 7,   70, 70,    70]
```

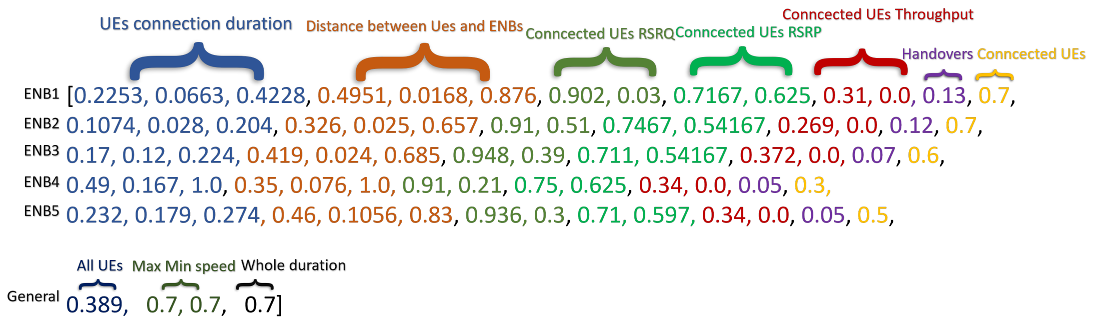
Figure 4.4: An example of the observations before normalization from one step in experiment A2A4_7

The normalization process is done for each ENB list by dividing by the maximum achieved values for distance, duration, RSRQ, RSRP, and throughput. For example, the normalized average connection duration between the base station and users is $= \frac{AverageConnectionDuration}{MaximumConnectionDuration}$, where the maximum duration is the longest connection duration happened between a base station and a user device in the step. The maximum values used to divide by and normalize the observations in Figure 4.4 and get the normalized observations in Figure 4.5 are shown in Table 4.1

Connection Duration	Distance	RSRQ	RSRP	Throughput
92006.033	1047.344	33	72	49517

Table 4.1: The maximum values used to divide by to normalize the observations in the ENB lists in Figure 4.4

The remaining observations are **normalized** in all experiments by dividing by some constant values. The number of handovers in each ENB list is divided by 100, and the number of connected



```
ENB1 [0.2253, 0.0663, 0.4228, 0.4951, 0.0168, 0.876, 0.902, 0.03, 0.7167, 0.625, 0.31, 0.0, 0.13, 0.7,
ENB2 0.1074, 0.028, 0.204, 0.326, 0.025, 0.657, 0.91, 0.51, 0.7467, 0.54167, 0.269, 0.0, 0.12, 0.7,
ENB3 0.17, 0.12, 0.224, 0.419, 0.024, 0.685, 0.948, 0.39, 0.711, 0.54167, 0.372, 0.0, 0.07, 0.6,
ENB4 0.49, 0.167, 1.0, 0.35, 0.076, 1.0, 0.91, 0.21, 0.75, 0.625, 0.34, 0.0, 0.05, 0.3,
ENB5 0.232, 0.179, 0.274, 0.46, 0.1056, 0.83, 0.936, 0.3, 0.71, 0.597, 0.34, 0.0, 0.05, 0.5,

General 0.389, 0.7, 0.7, 0.7]
```

Figure 4.5: The normalized version of the observations in Figure 4.4

UEs is divided by 10. The number of all UEs in the general values list is divided by 18, which is the maximum number of users in all experiments. The maximum and minimum speeds and the simulation duration are divided by 100.

4.2.3 The action space

The action space includes the adjustments controlled by the agent on the handover algorithm attribute values. The attributes depend on the handover algorithm type.

For the **A2A4** algorithm, the attributes are NeighbourCellOffset and ServingCellThreshold. Both have limits between 0 and 34 dB.

And for the **A3** algorithm, the attributes are Hysteresis which has limits between 0 and 15 dB, and TimeToTrigger, with limits between 0 and 5120 ms.

All attribute values are float/real values; therefore, the action space is continuous for both handover algorithms. The handover attributes will be mentioned without units in future sections.

4.2.4 The reward functions

The goal is to choose the handover attribute values to optimize the performance of the handover algorithm by increasing the data throughput delivered to the user devices and reducing the number of handovers. To achieve that, the reward value must guide the agent during the training. Three different reward values have been used in different experiments. Each of them guides the agent to different goals: maximizing the throughput and minimizing the number of handovers or just maximizing the throughput. To maximize the throughput received by the users, it could be necessary to trigger several handovers to connect to the base stations that provide better RSRQ and RSRP. All methods are used for the two handover algorithms because they don't depend on specific factors in the handover algorithm. The rewards are:

The Optimize Ratio: The main goal was to maximize the value of throughput for the UEs and optimize the number of handovers happening during the experiment to produce the best result. Therefore, the Optimize Ratio function, which is used and explained in [41], has been used as a reward in some of the experiments to check the ability of the SAC RL agent to find the optimal attribute values. This value is used for both handover algorithms.

As mentioned in [41], the optimize ratio is calculated for a constant user speed, but it has been tested as a reward function in varied and constant user speed situations. The optimize ratio is found based on the equation 4.2.

$$optimizeRatio = \frac{TotalThroughput}{ANOH} \times \frac{1}{5 \times 10^6} \tag{4.2}$$

The TotalThroughput is the total number of bytes received by all UEs per 100 ms for all base stations. The ANOH is the average number of handovers per UE per 100 ms.

$$ANOH = \frac{HandoverCount}{UECount \times T} \tag{4.3}$$

The ANOH is found based on equation 4.3 where HandoverCount is the number of all handovers that happened during the simulation episode. UECount is the number of all users in the simulation episode, and T is the simulation duration.

$$TotalThroughput = \sum_{BS=1}^{BS=M} \sum_{UE=1}^{UE=N} \sum_{t=1}^{t=T} \frac{Bytes(t)}{T} \tag{4.4}$$

TotalThroughput is found based on equation 4.4 where M is the total number of base stations, N is the total number of users, T is the simulation duration, and Bytes(t) is the number of bytes received by the user in 100 ms.

The reason for dividing by 5×10^6 is to reduce the high optimize ratio values computed in different steps for better RL agent guidance toward the highest reward value possible.

Total Throughput: The other goal was to maximize the throughput received by the UEs regardless of the number of handovers. The reason for ignoring the number of handovers is that the throughput can be maximized by triggering several handovers. The throughput is not minimized when more handovers happen. The total throughput is the value computed by the equation 4.4. The total throughput is divided by 5×10^4 to produce the reward value used by the RL agent because the calculated total throughput value is a big number which can confuse the RL agent since the difference between the total throughput values in different steps can be inconsiderable in comparison with the total throughput value itself.

Total Throughput and RSRQ: This reward function is computed using equation 4.5:

$$Reward = \frac{TotalThroughput + TotalRSRQ}{5 \times 10^4} \tag{4.5}$$

Where TotalRSRQ is found by equation 4.6:

$$TotalRSRQ = \sum_{BS=1}^{BS=M} \sum_{UE=1}^{UE=N} \sum_{t=1}^{t=T} \frac{RSRQ(t)}{T} \tag{4.6}$$

M is the total number of base stations, N is the total number of users, T is the simulation duration, and RSRQ(t) is the RSRQ received by the user in 100 ms.

This reward function has been used in one A2A4 experiment to check the effect of adding the RSRQ value to the total throughput on the RL guidance to choose the best attribute values that maximize the throughput and signal quality.

4.3 RL Agent Design

The RL agent is responsible for finding the optimal handover attribute values for a specific area and the users moving in this area. A constant number of base stations is considered in this area. The RL algorithm used is the Soft Actor Critic (SAC) algorithm, and its performance is compared with the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm.

4.3.1 Soft actor-critic algorithm

The Soft Actor-Critic (SAC) algorithm is used for the RL implementation because it has not been used that much previously for mobile network improvements. It has shown promising results in other applications with continuous action/state spaces, which are compatible with our problem as discussed in [31]. Additionally, the users move randomly between the BSs, and there are many network situations to test by changing users' speed and positions. Because of that, an agent who tests and evaluates many different actions is needed to cover and explore as many situations as possible. That is available in SAC because it depends on the maximum entropy RL framework and trains a stochastic policy to ensure that the agent acts randomly. The SAC algorithm is built based on two critic networks and one actor network. Most of the experiments are done using this algorithm.

4.3.2 Twin delayed DDPG (TD3) algorithm

This algorithm has been used in some experiments to compare its performance with the SAC algorithm and to see if it will find the optimal attribute values for each network situation. The TD3 algorithm trains deterministic policy. It has shown some excellent results in mobile network applications with continuous state and action spaces, as mentioned in the related work 3.11. This algorithm uses two critic networks and one actor network.

4.3.3 RL agent setup

The reinforcement learning system used in this project is shown in Figure 4.6. It shows the interaction process between the RL agent, the training and evaluation environments, and the policy update. The tools used to build this system and the training and evaluation processes are explained below.

Tools and implementations

The foundation of building the RL system is the TensorFlow tutorials like [52] and [53].

The RL Environment:

The environment is built as a custom Python environment from TensorFlow tf agents. It is implemented by inheriting the **tf_agents. environments.PyEnvironment** [54] and adjusting the action and observation spaces and the reset and step functions.

The RL Agent:

The **tf_agents.agents.SacAgent** is used from TensorFlow tf_agents as the RL SAC agent with one Critic network as tf_agents.agents.ddpg.critic_network and one actor network as actor_distribution_network.ActorDistributionNetwork from tf_agents.networks. For TD3 agent **tf_agents.agents.Td3Agent** has been used. The same critic network as the SAC agent is used, but the actor network is tf_agents.agents.ddpg.actor_networ. The training and evaluation policies are implemented using tf_agents PyTFEagerPolicy. The Collector and Evaluators were shown in Figure 4.6 are 'tf_agents.train.actor'.

Training Process

In every training episode and as shown in Figure 4.6, the collector runs the environment using actions produced by following the agent's collection policy to collect data from the environment during the training. The environment runs the simulator with the fixed parameters and attribute values, computes the reward after applying the received action, and returns the observations and the computed reward. Then the collector pushes a tuple of (state, action, reward) to the replay buffer to be then used by the Learner and sent to the agent to be trained. The Learner is a tf_agents Learner that runs the RL agent using the data stored in the replay buffer. The RL agent updates its target policy with the action distribution. It also updates the collection policy to select training actions.

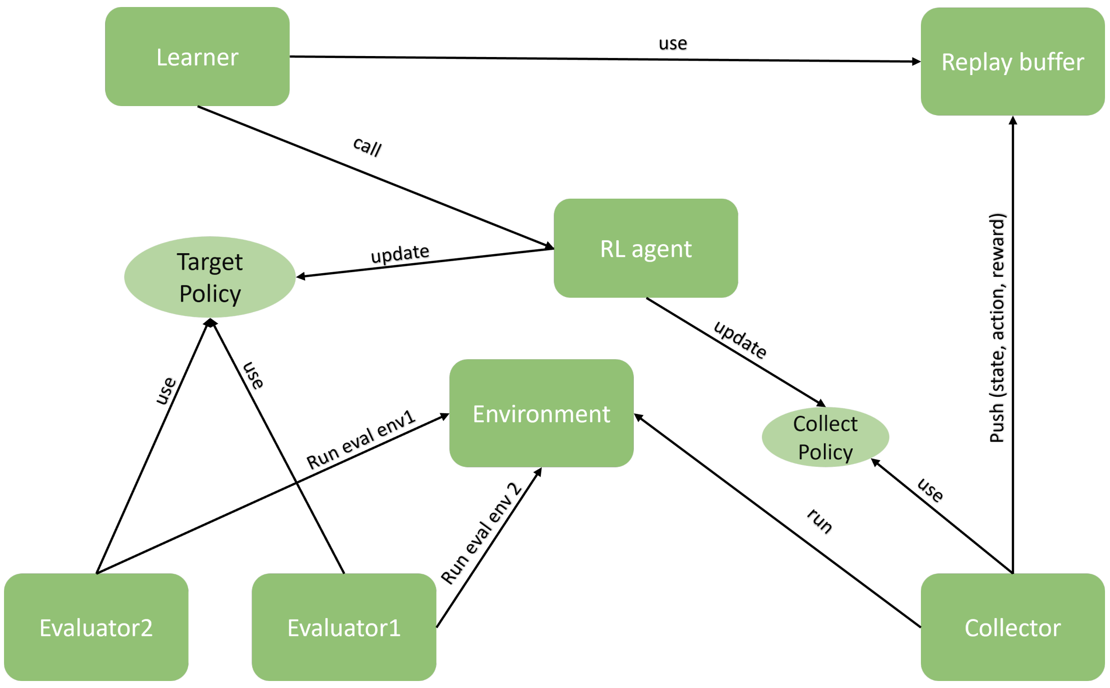


Figure 4.6: A diagram that shows the structure of the RL system

Evaluating Process

For the evaluation part and as shown in Figure 4.6 there are two different static evaluation environments specified in the main environment and two evaluators. Each evaluator calls its evaluation environment and uses the target policy to get actions, evaluate the RL agent's performance after, and log the rewards received. Each evaluation environment is implemented with constant and specific parameter values to evaluate the performance of the RL agent. The setup of the evaluation environments for each experiment is explained in section 5.2.1.

One experiment has just one evaluator and one evaluation environment called every 10 training steps. This is the experiment A2A4_1 explained in 5.2.1. In the other experiments, there are two evaluation environments. The first evaluation environment is called every 10 training steps, and the second is called every 15 training steps. The sequence of the first 40 training steps and the evaluation steps is shown in Figure 4.7.

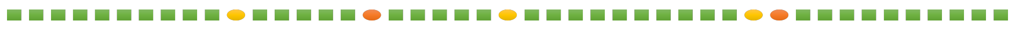


Figure 4.7: The sequence of the first 40 training and evaluation steps. The green rectangles are the training steps, the yellow circles are the steps from the first evaluation environment and the orange circles are the steps from the second evaluation environment.

Results and Evaluation Log

During every training or evaluation step, there are some values saved into the final log file to be used in analysis and exploration. These values are:

- The actions produced by the agent
- The loss value after applying each action
- The reward found after each step
- The number of handovers
- Total throughput received by all users during the simulation in this step
- Total RSRQ received by all users during the simulation in this step for A2A4 experiments
- Total RSRP received by all users during the simulation in this step for A3 experiments
- Maximum and minimum users' speed since they are changed every 50 steps in most of the experiments
- Simulation duration since it is also changing during the training.
- Total Throughput and RSRQ for each base station and its connected users after each step.

The SAC agent

The SAC algorithm is built based on two critic networks. the first critic network is implemented using tf_agents.agents.ddpg.critic_network with two layers and the state size of fully connected parameters to handle the state parameters together. The second critic network is built by copying the same weights from the first network.

The actor-network is actor_distribution_network.ActorDistributionNetwork from tf_agents.networks with two layers and the state size as layers size.

Hyperparameters used for the RL agent are shown in Table 4.2:
Actor, Critic, and Alpha learning rates are used in the actor, critic, and alpha Adam optimizers respectively.

The reward discount factor for future rewards. It's near 1 because the environment is changing in every step and the simulated network is unique in every training step to explore many network situations. Therefore the reward doesn't need to get smaller after each step.

Actor learning rate	10^{-4}
Critic learning rate	10^{-3}
Alpha learning rate	10^{-3}
γ **Reward discount factor**	0.999
τ **Target update factor**	0.005
Loss function	Squared Difference

Table 4.2: Hyper parameters for SAC agent

Target update τ is the soft update value of the target network. It slowly updates the weights in the target network by not changing them totally in every training step. Instead, the weights will be updated by 0.5% from the new values and 99.5% from the values of the old weights in this case since the Target update tau is 0.005.

The TD3 agent

The TD3 agent uses almost the same system design and hyperparameters as the SAC agent, but the TD3 agent doesn't use alpha Adam optimizers.

4.4 Experiment Processes

The RL agent has been trained on two A2A4 RSRQ and A3 RSRP handover algorithms. More experiments with more network scenarios are applied to the A2A4 algorithm. The same RL agent system and environment setup are used for both handover algorithms, but the algorithm attributes themselves are changed as mentioned in section 4.2.3. The code of experiment A2A4_7 (explained in 5.2.1) in this work is added on GitHub in [55]. This repository includes the RL environment, agent setup, and the simulation code.

4.4.1 Experiment length

As mentioned before, the simulator is called in every RL step (or episode) which means that one episode includes:

1. Deciding the handover attribute values by the RL agent's policy
2. Deciding the network scenario parameters in the RL environment
3. Calling the simulator with these parameters to build the simulated network and run it for the decided simulation duration
4. Logging the measurements produced during the simulation
5. Parsing these measurements by the RL environment to produce its observations to be used further in the RL learning process.

That means that high CPU usage is needed to run several episodes. The length of one episode ran on the local machine with an Intel Core i7 processor is around 1.5 minutes. And it needed almost one week to run one experiment with 7000 steps. In order to run as many experiments as possible, a virtual machine from the Department of Computer Science at NTNU has been used in addition to the local machine. Thus, it was possible to run 3 experiments in parallel. Two experiments ran on the local machine and one on the virtual machine. Both local and virtual machines are single-threaded.

4.4.2 The grid search experiments

The grid search experiments are done manually without reinforcement learning usage. Each grid search experiment includes a different simulated network situation with different handover attribute values. During each step, the handover attribute values change, but all network scenario parameters are the same except for the randomness in users' movements during the simulation. Each step tries a unique pair of the handover attribute values. The pairs are (ServingCellThreshold, NeighbourCellOffset) for the A2A4 handover algorithm and (TimeToTrigger, Hysteresis) for the A3 algorithm. They compare the total throughput reward received in each step to find the best attribute pair that gives the highest reward value possible. The reason of conducting these experiments is to evaluate the performance of the RL agent in the RL experiments with the two handover algorithms.

Two grid search experiments used the A2A4 handover algorithm and two others uses the A3 algorithm. Their results will be presented in section 5.2.2 in the next chapter 5. The results from the grid search experiments are compared to the RL experimental results in the next chapter too. The handover attribute values chosen to be used in the grid search experiments are:

- **A2A4 handover algorithm** ServingCellThreshold values are [1, 4, 7, 11, 15, 19, 23, 25, 29, 32, 34], and NeighbourCellOffset values are [2, 5, 9, 13, 16, 20, 23, 25, 28, 31, 33.5]
- **A3 handover algorithm** TimeToTrigger values are [0, 40, 64, 80, 100, 128, 160, 256, 320, 480, 512, 640, 1024, 1280, 2560, 5120] and Hysteresis values are [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15.5].

4.5 Summary

This whole system structure used to apply the experiments was explained in this chapter. The observation and action spaces, all reward functions used, the simulation network structure, and the RL system and algorithms are discussed. The process followed by all experiments is also mentioned in addition to an introduction about the grid search experiments applied and their main goal. The next chapter 5 will provide a wider overview of the experiments done, their purpose, and their results from different perspectives.

Chapter 5

Experiments and Results

Many experiments have been conducted and analyzed to evaluate the performance of the RL agent and the effect of changing network setup on the optimal results achieved by the agent. The discovered results will be mentioned, discussed, and analyzed in this chapter. The plan, all components for each experiment, and its results will be explained in detail.

5.1 Experimental Plan

Each experiment has its own plan and goal. The plan includes the handover algorithm, the network parameters, the evaluation environments, and the RL agent algorithm and its hyper-parameters setup. While the goal is the purpose of applying the experiment to look into the achieved results and to see if they are expected or not. Some of the results were not expected and the RL agent showed a unique behavior. The plan for each experiment will be explained and the experiment setup and its results will be discussed further in the coming sections in this chapter.

5.1.1 Different network situations

A very important part of the experiment is the simulated network which is called in every training and evaluating step, and its structure needs to be built in a rational way to analyze the RL agent behavior. The RL agent is responsible for a specific area and all BSs located in this area and the UEs moving inside it. This area is $1440m^2$. Some experiments have been done for comparing reasons to compare the performance of different RL models, reward functions, and the effect of changing the network parameters.

The network scenario parameters are changed during the training to facilitate the agent's exploration of diverse situations. The reason is to observe the agent's behavior while learning in different network situations. The simulation duration and number of users are changed in the experiments every 100 training steps. While the user's maximum and minimum speeds and starting x and y coordinates are changed every 50 training steps. The purpose of this changing method is to make the agent observe multiple situations where the number of users and the simulation duration are the same. The main goal is to check the behavior of the RL agent to find the optimal solution for each situation. Some other experiments have tested based on changing a few network parameters like the user speeds while other parameters were constant. The reason for that is to evaluate how optimal the RL agent results are for each situation specifically.

5.1.2 Handover algorithms

The two handover algorithms, A2A4 and A3 handover algorithms, are used. The most of experiments have run based on the A2A4 algorithm since this algorithm has shown different agent behavior. The same simulated network setup, observation space, and reward computing are used for both algorithms, but the attribute values are changed depending on the algorithm. The other scenario parameters don't depend on the handover algorithm itself. However, the A2A4 algorithm uses RSRQ value to decide the handover and the A3 algorithm uses RSRP.

5.1.3 RL algorithms

In the beginning, the Soft actor-critic RL algorithm has been used to find the optimal handover attribute values. The SAC algorithm works to randomize the actions chosen during the training to find the optimal solution, and that was an important part to explore since the agent observes many unique simulated network situations. The purpose was to find how optimal the values found by the RL agent are. The performance of the SAC algorithm is compared with the TD3 algorithm performance to see if they can achieve an optimal solution for each network situation specifically.

5.1.4 Optimal solution

The results found by the agent are compared to see if the results are optimal for each network situation. They are also compared with the grid search experiment results. Different network scenarios are compared to evaluate if the solution is the best for each network situation or for all situations generally.

5.2 Experimental Setup

All the experiments applied and explored are explained with their properties in this section. Each part of the experiment is covered, including the simulated network structure and parameters, the environment setup, and the evaluation environments.

5.2.1 Simulated network and environment setup

Different simulated networks have been used. They include five base stations (ENBs). Three scenarios, including five base stations in various positions, are tested to see the effect of the base station position on the optimal handover attributes. They are shown in Figure 5.1, which includes three base station position setups.

Multiple experiments have been done on both handover algorithms to analyze the agent's behavior in different network and environment situations. The general purpose was to make the agent adapt to the environmental changes and pick the optimal attribute values for each individual network scenario. The experiments have focused on different areas like changing many parameters in the simulated network many times during the training, changing just one or two parameters while others are constant, testing different base station positions in different experiments, choosing network parameters randomly, and changing the simulation duration.

Additionally, two different RL algorithms for continuous action/state spaces applications are tried, which are SAC and TD3. Each experiment has a name that will be used further in the results section. The name includes the handover algorithm name and a number, for example, A2A4_2, where A2A4 is the handover algorithm and 2 is the number.

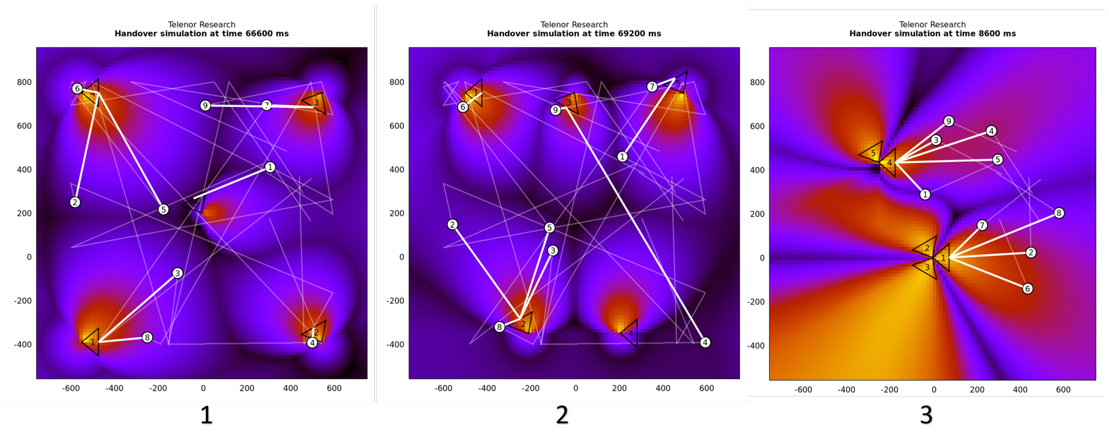


Figure 5.1: The simulated network with five base stations with different locations. The number below each network is used in future tables to refer to the network setup.

The experiments are listed first with their handover algorithm section, then their results are discussed in detail. The order of the experiments listed below is the order they were applied in. However, the results don't show the same order.

A2A4 Experiments

The work has been started with this algorithm, and most of the experiments are conducted using the A2A4 algorithm since it has shown different performances in different experiments. Multiple experiments are done, and the most important ones that have shown interesting results are presented here.

- **A2A4_1**:
  - **Training setup:** The SAC RL algorithm is used. The simulated network includes 5 BSs with positions shown in the third plot in Figure 5.1. Every 100 steps, the duration is randomly chosen from this list [60s, 70s, 80s, 90s, and 100s], and the number of users is also randomly chosen between 3 and 11. All users move at the same speed, which is 30 m/s, to analyze the effect of changing other network parameters. Users' initial positions are randomly chosen inside the simulated network limits every ten steps. They move randomly during the whole simulation episode. The reward function used is total throughput, explained in section 4.2.4.
  - **Evaluating setup:** To evaluate the performance of the RL agent, an evaluating environment has been implemented with 5 BSs, 7 UEs, users' speed is 30 m/s, and the simulation duration is 80 s.
- **A2A4_2**
  - **Training setup:** The SAC RL algorithm is used. This experiment is conducted three times with 5 BSs located differently like the three setups shown in Figure 5.1. The reason for that is to compare the behavior of the RL agent with different base station position scenarios. The names of the three experiment versions are A2A4_2a,

A2A4_2b, and A2A4_2c. Every 100 training steps. The duration is randomly chosen between [60 s, 70 s, 80 s, 90 s, 100 s, and 110 s], and the number of users is randomly chosen between 3 and 11. The users' maximum and minimum speeds and starting locations are changed every 50 steps. Maximum speed is 30 m/s or 40 m/s, while minimum speed is 20 m/s or 30 m/s. The reward function for both experiments is the total throughput explained in section 4.2.4.

  - **Evaluating setup:** Two evaluating environments have been implemented for all versions. Parameter values of both evaluation environments are displayed in Table 5.1.

ENBs	**UEs**	**Speed**	**Duration**
5	7	25 m/s	70 s
5	9	40 m/s	100 s

Table 5.1: A2A4_2 evaluation environment setups. The duration is the simulation duration

- **A2A4_4**
  - **Training setup:** This experiment has also the same training setup as A2A4_2 with 5 BSs located like the third plots in Figure 5.1. The reward function is the optimize ratio explained in section 4.2.4. The reason for this experiment is to compare the performance of this reward function and the total throughput reward and evaluate its ability to optimize the number of handovers.
  - **Evaluating setup:** The evaluation environments are also the same as the evaluation environments in A2A4_2.
- **A2A4_5**
  - **Training setup:** Both TD3 and SAC RL algorithms are used to compare their performance. This experiment has two versions with a different RL algorithm each. The simulated network had five BSs with location setup shown in the third plot in Figure 5.1. Every 100 steps, the duration is randomly chosen between [60 s, 70 s, 80 s, and 90 s], and the number of users is also randomly chosen between 7 and 9. The users' speed and users' initial positions are changed during the training every 50 steps The positions are randomly chosen inside the simulated network limits. The maximum speed is 40 m/s, and the minimum speed is 20 m/s. The reward function is the total throughput.
  - **Evaluating setup:** The setup of the two evaluation environments is displayed in Table 5.2.

ENBs	**UEs**	**Speed**	**Duration**
5	8	20 m/s	70 s
5	8	40 m/s	80 s

Table 5.2: A2A4_5 evaluation environment setups.

- **A2A4_6**

- **Training setup:** This experiment aimed to see how the agent reacts to high-speed situations with multiple users. The SAC RL algorithm is used. The simulated network had five BSs with location setup shown in the second plot in Figure 5.1. Every 100 steps, the duration is randomly chosen from this list [110 s, 120 s, and 130 s], and the number of users is also randomly chosen between 9 and 18. The users' speed is 70 m/s, which is constant during the training session. Users' initial positions are changed during the training every 50 steps. The reward function is the total throughput and RSRQ explained in 4.2.4.
- **Evaluating setup:** The setup of the two evaluation environments shown in Table 5.3. The reason for having different user speeds in the evaluation environments than the user speed in the training environment is to evaluate if the actions decided by the agent in the high-speed situation are suitable for other speed situations.

ENBs	UEs	Speed	Duration
5	9	50 m/s	110 s
5	9	40 m/s	100 s

Table 5.3: A2A4_6 evaluation environment setups.

- **A2A4_7** The code of this experiment is added in [55].
  - **Training setup:** Network parameters are constant in all training steps except for the number of users and their speed. The user speed is randomly chosen between [20 m/s, 40 m/s, and 70 m/s] every 50 steps. Minimum and maximum speeds are equal. There are 5 BSs, [7, 8 or 9] UEs, and the duration is 80 s. The reward function is total throughput.
  - **Evaluating setup:** The evaluation environments evaluate the performance in low and high-speed situations. The setup of both evaluation environments is displayed in Table 5.4.

ENBs	UEs	Speed	Duration
5	8	20 m/s	80 s
5	8	70 m/s	80 s

Table 5.4: A2A4_7 evaluation environment setups.

A3 Experiments

The A3 handover algorithm has been used in these experiments by changing different parameters to analyze the agent performance. Some experiments have shown unexpected results. The different methods used in these experiments are explained further.

- **A3_1**
  - **Training setup:** The network training setup for this experiment is the same setup in A2A4_2 to check the performance of the RL agent on the A3 handover algorithm and the same setup. The reward function is the optimize ratio.

- **Evaluating setup:** The evaluating environments are also the same as the evaluation environments of A2A4_2.

- **A3_2**
  - **Training setup:** It has 5 BSs located as shown in the third plot in Figure 5.1. The duration is randomly chosen between [70 s, 80 s, and 90 s] every 100 steps. There are eight users, and their speeds are randomly chosen from [20, 40, and 70] every 50 steps. Maximum and minimum speeds are equal. Users' initial positions are randomly chosen inside the simulated network limits every 50 steps.

    This experiment has been done with the total throughput reward function explained in 4.2.4 but, it has shown that the RL agent converges toward the maximum values for both handover attributes. That doesn't seem correct since the grid search experiments, shown in figures 5.4 5.5, show very low total throughput received when the attributes had their maximum values. The results from the experiment with the normal total throughput are shown in appendix A. After analyzing the changes in the total throughput found after each step, one solution was to make the differences between the received total throughput values in each step bigger. Therefore, the reward function used has been changed. The new reward function is a rescaled value from the total throughput and is = (total throughput -2) $\times$ 10.
  - **Evaluating setup:** Parameter values of both evaluation environments are displayed in Table 5.5.

ENBs	UEs	Speed	Duration
5	7	70 m/s	80 s
5	9	40 m/s	100 s

Table 5.5: A3_2 evaluation environment setups.

5.2.2 Results and evaluation log

Every experiment has its evaluating environments, as explained above in the previous section. The evaluation environments evaluate the learning process after applying multiple training steps. The A2A4_1 experiment has one evaluating environment and is called every ten training steps. In contrast, all other experiments have two evaluating environments to validate how optimal the actions chosen by the RL agent are in different network scenarios. The first is called every ten training steps, and the other is called every 15 training steps.

The performance is evaluated by applying the actions the RL agent decides using its policy in the evaluation environments. The actions, rewards, total throughput, total RSRP, total RSRQ, and the number of handovers that occurred after calling the evaluation environments are saved to a log file to analyze the performance of the agent after running all the evaluation steps.

5.3 Experimental Results

Multiple interesting results have been discovered after trying different situations and running all experiments. The reason behind some of the trials is to make the RL agent produce adaptive optimal handover attribute values for each simulated network to maximize the throughput received by users and to optimize the total number of handovers. Therefore, the performances of the two RL algorithms are analyzed and compared. The results after using different reward functions are compared too, and the optimal solutions produced by the agent are discussed.

5.3.1 Performance evaluation and Grid search experiments

Some grid search experiments have been done to evaluate how optimal the attribute values chosen by the RL agent are. The setup of the simulated network of all grid search experiments is shown in Table 5.6. The results of these experiments for both handover algorithms are displayed below in figures 5.2, 5.3, 5.4, and 5.5. In these figures, the values shown on the heat maps are the reward values received after running each step with the attribute values shown on the two axis.

Experiment	Duration	ENBs	UEs	Speed	Reward function
First A2A4	90 s	5	8	30 m/s	Total Throughput
Second A2A4	90 s	5	12	70 m/s	Total Throughput
First A3	90 s	5	8	30 m/s	Total Throughput
Second A3	100 s	5	8	60 m/s	Total Throughput

Table 5.6: The network setup of all grid search experiments

A2A4

The **first grid search experiment** results are shown in Figure 5.2, the maximum value is achieved when NeighbourCellOffset is seven, and ServingCellThreshold is 16. Two other high values are achieved when NeighbourCellOffset is 29, ServingCellThreshold is 4, and when NeighbourCellOffset is 21, and ServingCellThreshold is 16. Some excellent values are produced when NeighbourCellOffset is 15 and ServingCellThreshold exceeds 12. The same experiments were tried with 12 users instead of 9, and the performance was a bit different, where the maximum value received when NeighbourCellOffset is 11, and ServingCellThreshold is 16. Some good values are also received when NeighbourCellOffset is 11 or 15, and ServingCellThreshold is higher than 20. This means that the values between 11 and 15 of NeighbourCellOffset can be optimal for this experiment. From both experiments, the optimal values of ServingCellThreshold are 16, and values are between 23 and 34.

The **second grid search experiment** results are shown in Figure 5.3. The reward values in this figure are higher than those in Figure 5.2 because there are several users in this experiment, which means that the total throughput received by all users can be more significant. In both figures, lighter colors refer to higher reward values. The maximum reward for this situation is achieved when NeighbourCellOffset is 15 and ServingCellThreshold is 23. The reward values are high when NeighbourCellOffset is between 15 and 19 and ServingCellThreshold is above 16. However, they are lower when ServingCellThreshold is 23 and 31. Minimum reward values are registered when the NeighbourCellOffset is 34, regardless of how much the ServingCellThreshold is.

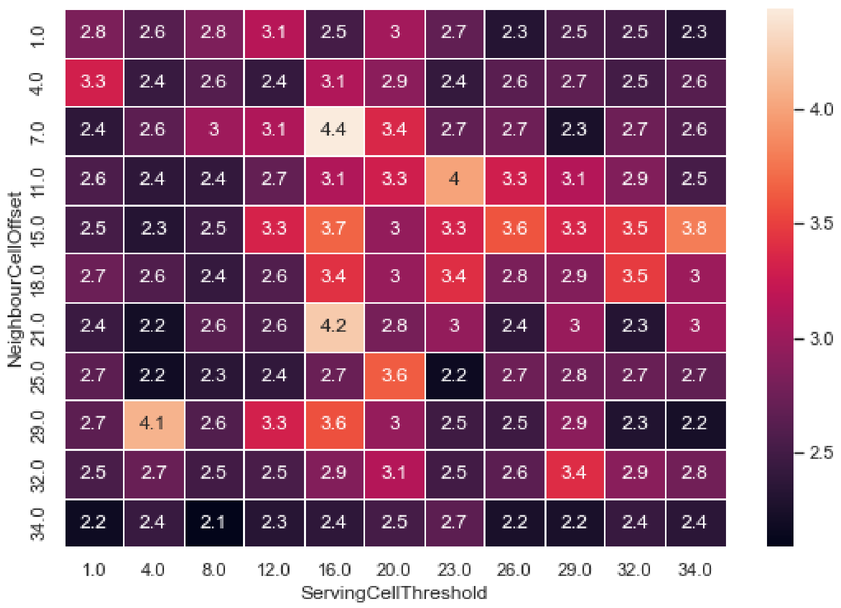


Figure 5.2: Results from the first grid search experiments. The values on the heat map represent the rewards received when applying the values of ServingCellThreshold and NeighbourCellOffset. This experiment has eight users, and their speed is 30 m/s.

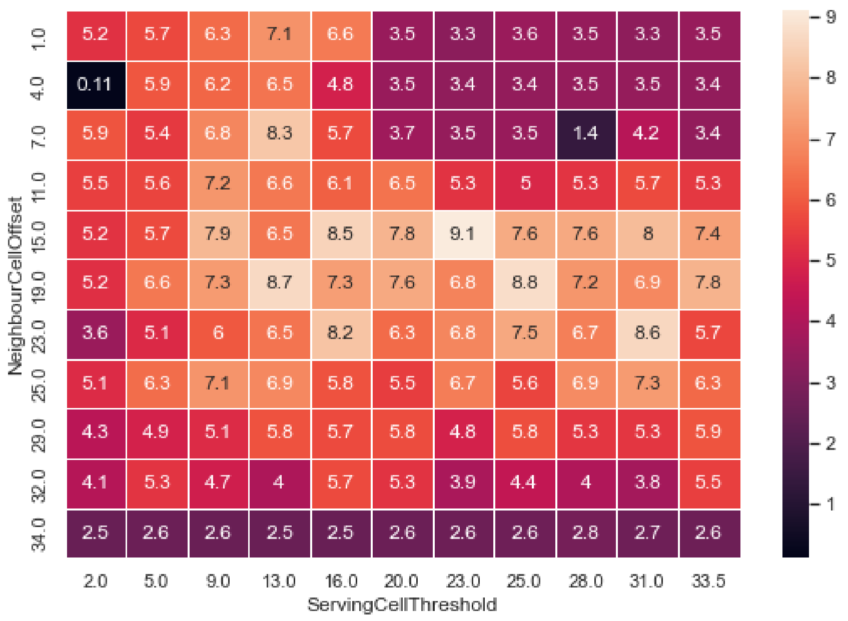


Figure 5.3: Results from the second grid search experiments. The values on the heat map represent the rewards received when applying the values of ServingCellThreshold and NeighbourCellOffset. This experiment has 12 users, and their speed is 70 m/s.

A3

The heat map of the first experiment is shown in Figure 5.4, and the second experiment is shown in Figure 5.5. Like A2A4 grid search experiments, Most reward values of the second experiment are higher than the first because the simulation duration is more prolonged. Thus, the users can receive more throughput while they move around for a more extended period.

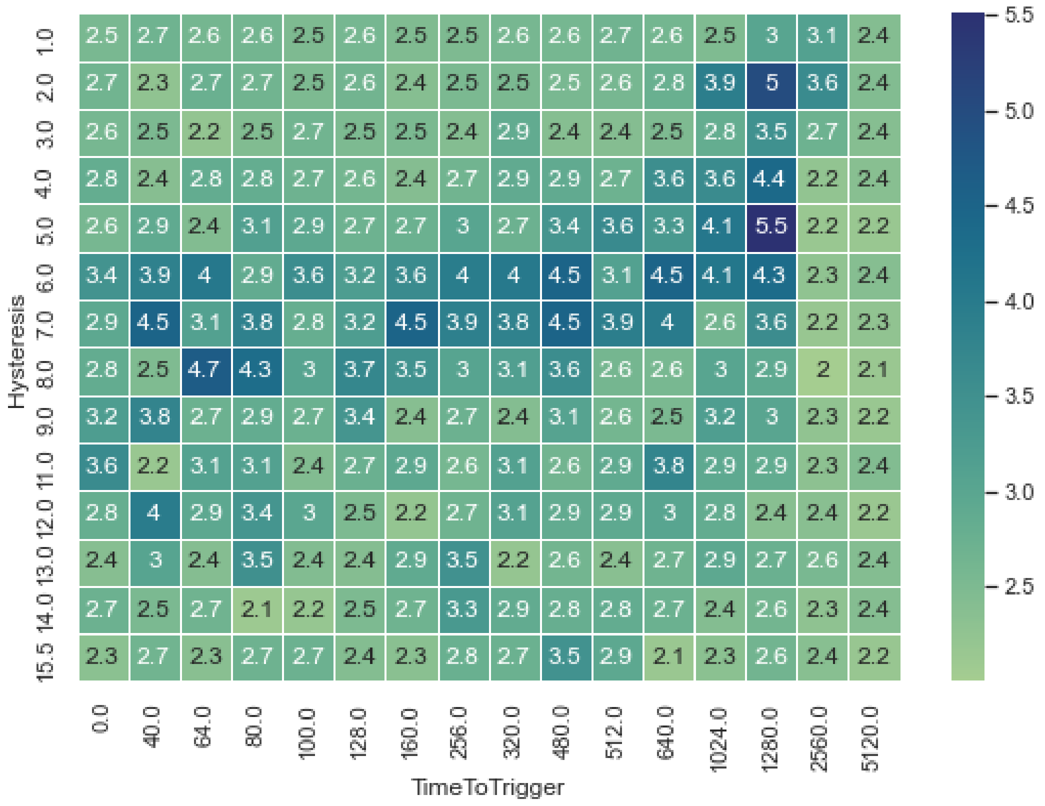


Figure 5.4: Results from the first A3 grid search experiments. The values on the heat map represent the rewards (explained in section 4.2.4) received when applying the values of ServingCellThreshold and NeighbourCellOffset. This experiment has eight users, and their speed is 30 m/s.

In both figures, darker colors refer to higher reward values, while lighter colors refer to lower values. The maximum reward value in the first experiment is received when Hysteresis is 5, and TimeToTrigger is 1280 s. The same TimeToTrigger value produced the maximum reward value, while Hysteresis is 4. However, in the second experiment, the best reward values are achieved when Hysteresis is between 4 and 6 and TimeToTrigger is 1024s or 1280s. That is also the case for the first experiment. In both experiments, some good reward values spread when Hysteresis is 6, 7, and 8, and TimeToTrigger is between 128 s and 1280 s.

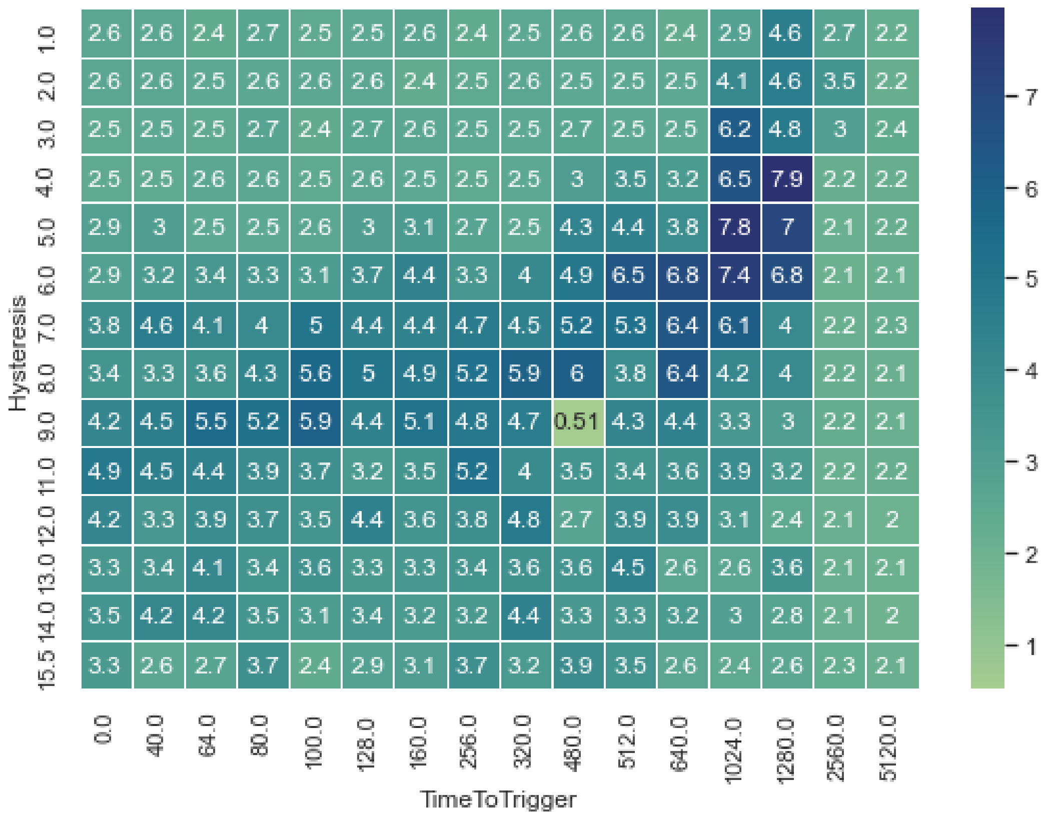


Figure 5.5: Results from the second A3 grid search experiments. The values on the heat map represent the rewards (explained in section 4.2.4) received when applying the values of Serving-CellThreshold and NeighbourCellOffset. This experiment has eight users, and their speed is 30 m/s.

5.3.2 Performance of TD3 and SAC RL algorithms

The performance of the Soft actor-critic and Twin Delayed Deep Deterministic Policy Gradient are compared and analyzed to see how each algorithm finds the optimal solution. Both algorithms have been applied to the same experimental setup in A2A4_5 to compare their performance. A2A4_5_td3 is the TD3 version, and A2A4_5_sac is the SAC version.

A2A4_5_td3 experiment is trained over 5560 steps while A2A4_5_sac had 6270 steps. After 5560 training steps, both agents show a reduction in the loss values as shown in Figure 5.6. The minimum loss value achieved by SAC is -27.99, and for TD3 is -3.72. The reduction ratio in the loss values is not very high in both algorithms. However, both agents could converge toward a solution and could improve (in different grades) the total throughput and RSRQ values received by all UEs in each evaluation step.

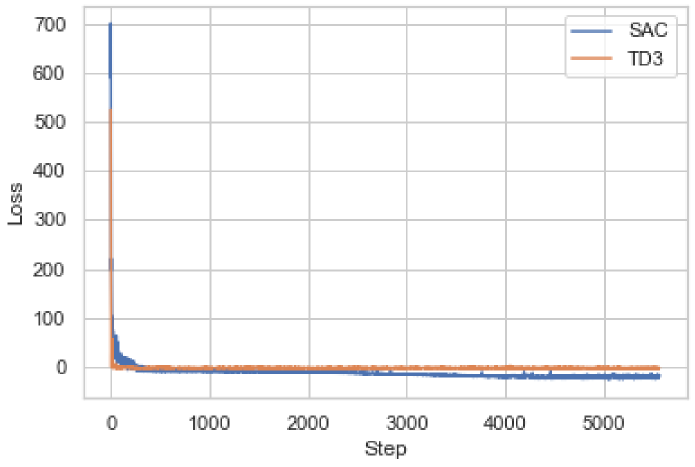


Figure 5.6: Losses for TD3 and SAC performances in A2A4_5.

Actions produced by both agents

Actions produced by both agents during the training sessions are shown in figures 5.7 and 5.8. It is clear from the plots that TD3 actions start to find the decision earlier than the SAC algorithm, which starts very randomly for more than 3000 steps before it starts to decide the best actions to be applied. From the definition of the SAC algorithm, which is explained in section 2.1.2, the agent depends on the maximum entropy besides the reward function to make its decisions which causes more randomness in the behavior. That can explain the reason for the randomness in the first 3000 steps. The randomness of the SAC algorithm and its effect on finding optimal attribute values for the handover algorithms by testing multiple unique network situations is a central part to explore in this project.

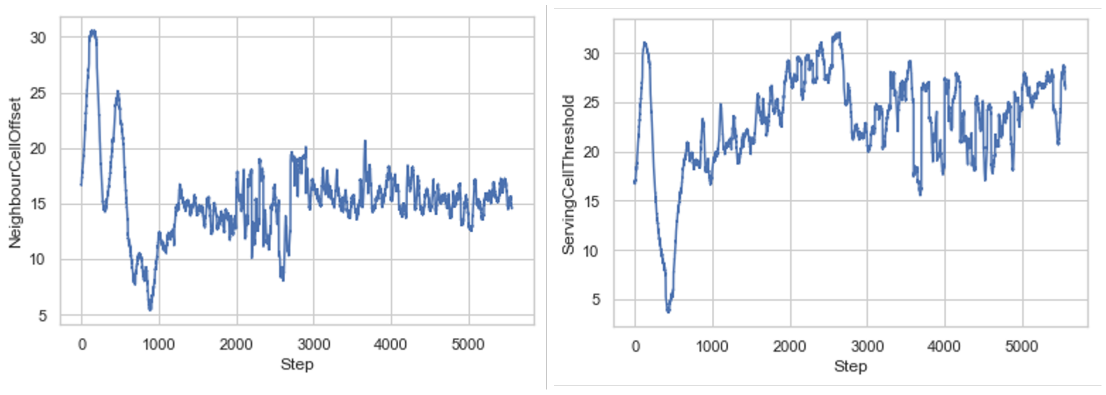


Figure 5.7: Actions produced by TD3 RL algorithm in A2A4_5.

On the other hand, the TD3 algorithm is trained on a deterministic policy, and that also can explain why the TD3 agent could decide its actions faster than the SAC agent. However, how optimal the attributes produced by both RL algorithms are, is a necessary side to analyze and evaluate.

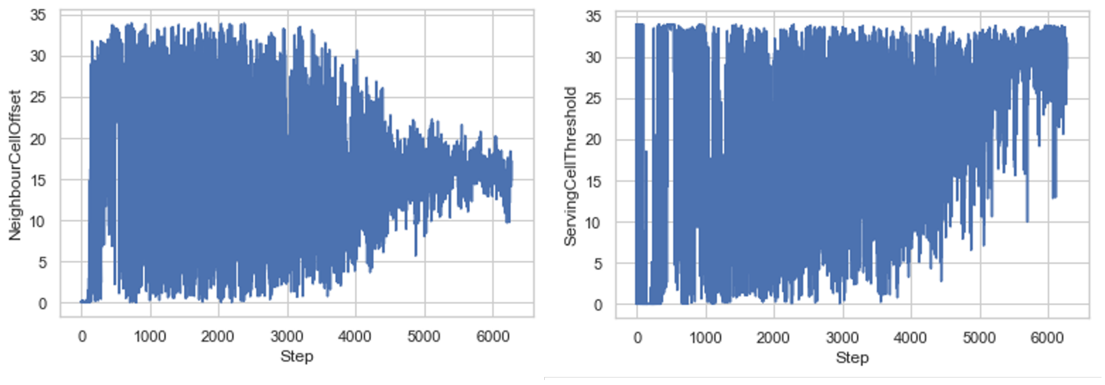


Figure 5.8: Actions produced by SAC RL algorithm in A2A4_5..

Standard Deviation (SD) and mean values of both handover attributes for the TD3 and SAC experiments from step number 4500 to the last step are shown in Table 5.7. As mentioned, the SAC experiment has more steps than the TD3 experiment. The mean value of the NeighbourCellOffset values produced by both RL algorithms is almost the same ≈ 15, but the SD of the TD3 actions is lower than the SD of the SAC algorithm, which means that TD3 is more deterministic in this situation, and could achieve the solution faster for this attribute.

As shown in the heat map in Figure 5.2, some good total throughput values are achieved when NeighbourCellOffset is 15. The ServingCellThreshold mean values are different in both experiments, and the SD value for the TD3 experiment's last steps is smaller than the SAC algorithm. That also shows that the TD3 algorithm could decide the solution faster than SAC.

By comparing the Standard Deviation (SD) value of both handover attributes in A2A4_5_td3 for the same steps, the SD of NeighbourCellOffset is lower than ServingCellThreshold. The same happens in A2A4_5_sac. That can mean that both RL agents could decide on a more accurate value of NeighbourCellOffset. However, the optimal solution differences between different situations can explain the higher SD value of ServingCellThreshold in the TD3 experiment. The agent picks different optimal attribute values since the network scenario parameters change every 50 steps. This means that the optimal value of the ServingCellThreshold attribute can vary considerably in every network setup situation.

_	NeighbourCellOffset	ServingCellThreshold
Mean TD3	15.2	24.2
SD TD3	1.04	2.9
Mean SAC	15.6	28.1
SD SAC	2.3	5.2

Table 5.7: The mean and standard deviation for the actions taken by the RL agent about both handover attributes from step number 4500 to the last step in TD3 and SAC experiments.

Figure 5.9 shows the handover attribute values chosen by the agent during the training from step number 4500 to 5560. It also shows maximum and minimum user speed changes and simulation duration between these steps. The performance differences are apparent when the network parameters are changed. That improves the ability of the TD3 agent to find adaptive handover attribute values based on the network situation.

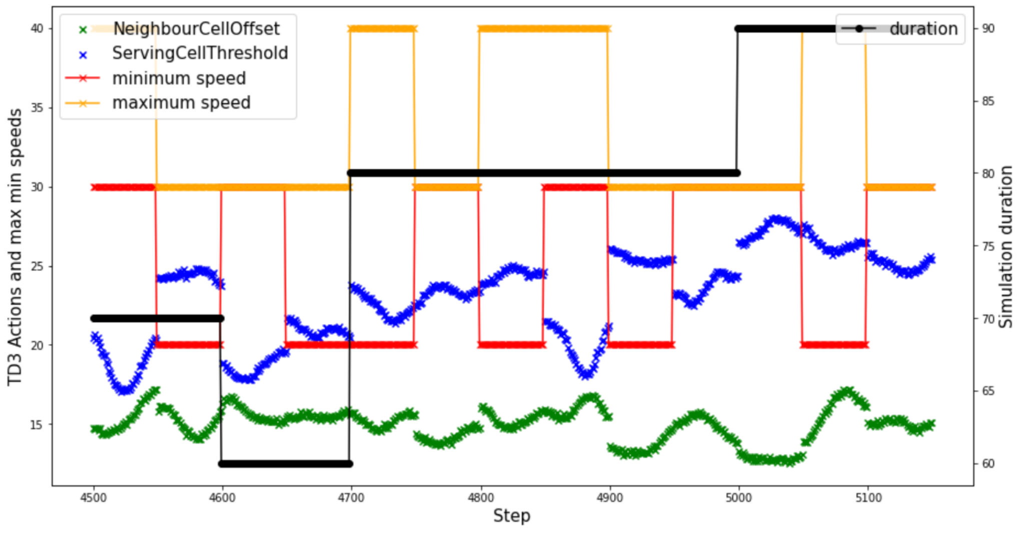


Figure 5.9: Actions produced by TD3 RL algorithms between step 4500 and 5560 compared with the changes in minimum and maximum speed values and the simulation duration. The duration is shown in black, the minimum user speed is shown in red, and the maximum speed is orange. The green and blue curves are the actions that are the handover attribute values.

At step 4600, the simulation duration changes from 70s to 60s, and the minimum speed changes from 20m/s to 30m/s. The maximum speed is not changed. At this step, it's visible that both NeighbourCellOffset and ServingCellThreshold values jump in their values based on the network changes that happened. The attribute values chosen after this step are Neighbour-CellOffset ≈ 15.5 and ServingCellThreshold ≈ 18. This situation has many similarities with the first grid search experiment, and by comparing with the heat map in Figure 5.2 a good value of NeighbourCellOffset can be 15 if ServingCellThreshold is more extensive than 16 but not the best one. ServingCellThreshold has not been tried to be 18, but 16 performed well.

The same plot is made for the SAC experiment in Figure 5.10 to see how adaptive the actions taken by the SAC agent are. It is visible that there is no explicit dependency on the network setup. The SAC agent prefers to find one direction of both attribute values.

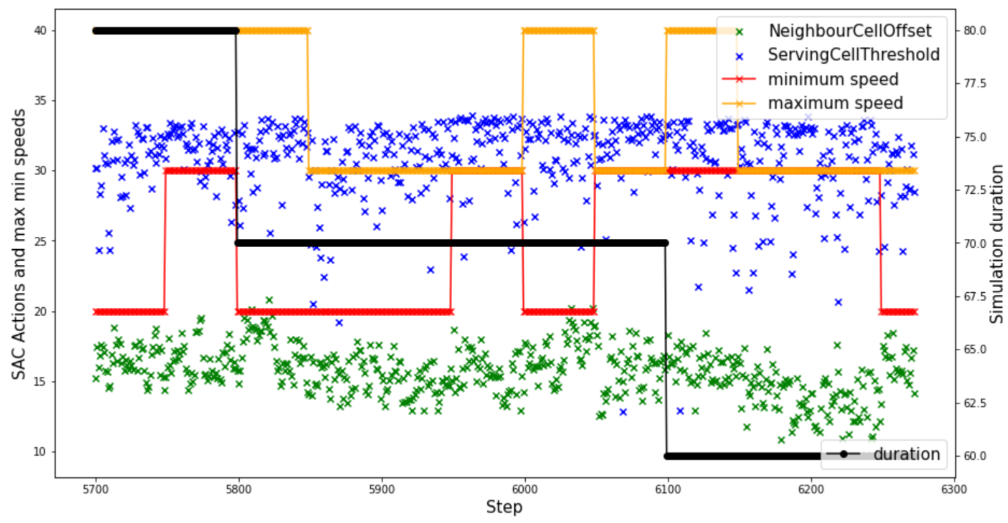


Figure 5.10: Actions produced by SAC RL algorithms in a specific period compared with the changes in minimum and maximum speed values and the simulation duration. The duration is shown in black, the minimum user speed is shown in red, and the maximum speed is orange. The green and blue curves are the actions that are the handover attribute values.

Improvements in total throughput and RSRQ

The evaluation environments are used to evaluate the performance of the RL policy after it is trained to find the optimal handover attribute values for each network situation. As explained, each evaluation environment is constant with the same network setup during all steps. After each evaluation step, the total throughput and RSRQ values are measured for both environments.

The total throughput values received after calling the second evaluation environment in both experiments are shown in Figure 5.11. From the right plot and in the SAC experiment, the throughput value improves after 300 evaluation steps which are 300x15=4500 training steps. On the other hand, the improvement in the TD3 experiment is visible from evaluation step number $\approx$ 190, equal to 190x15=2850 training step. The plots of the first evaluation environment are shown in appendix B.

The total RSRQ values measured after calling the first evaluation environment in both experiments are shown in Figure 5.12. The improvement of the RSRQ value in the TD3 experiment is clear after evaluation step 300, which is 300 x 10 = 3000 training step. From the SAC plot, the improvements begin to be visible after evaluation step number 450, which is equal to 4500 x 10 = 4500. That shows that the SAC algorithm could optimize the throughput value after almost 4500 training steps while the TD3 algorithm could do that after almost 3000 steps.

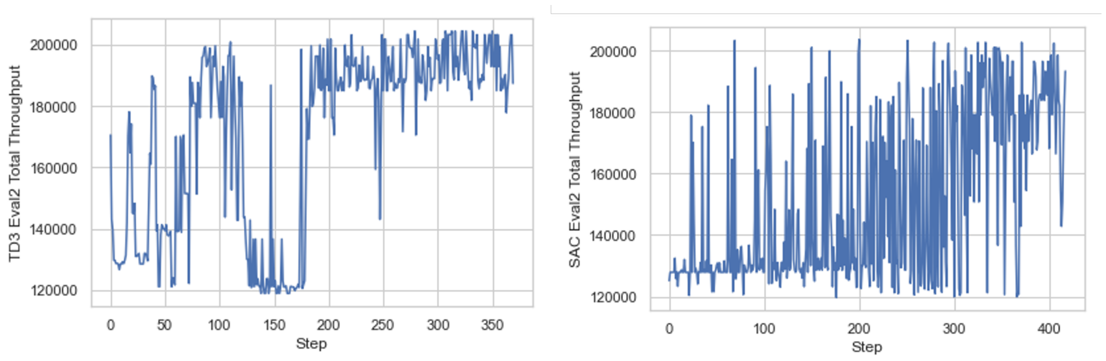


Figure 5.11: The total throughput measured from the second evaluation environment of TD3 and SAC experiments. The curve to the right is from A2A4_5_sac, and the curve to the left is from A2A4_5_td3.

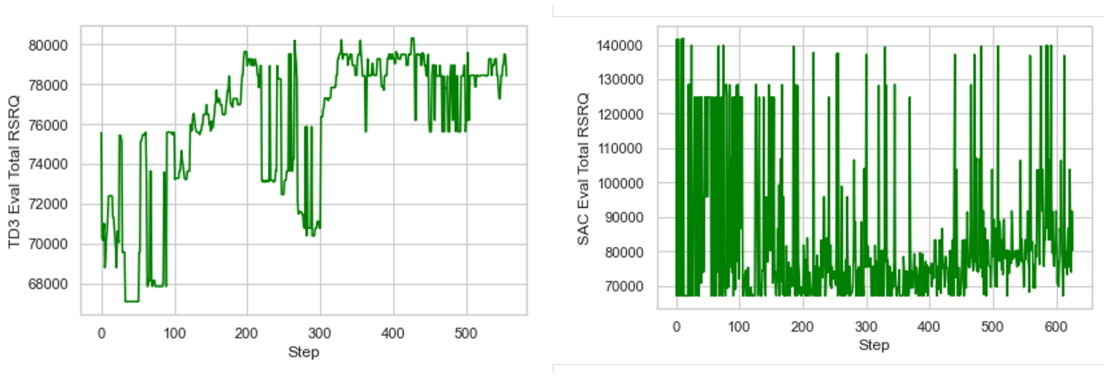


Figure 5.12: The total RSRQ measured from calling the first evaluation environment of TD3 and SAC experiments. The curve to the right is from A2A4_5_sac, and the curve to the left is from A2A4_5_td3.

Table 5.8 shows the mean of the total throughput and RSRQ values received by all UEs during all steps for both TD3 and SAC experiments. The mean value is measured for the first 100 evaluations, the middle, and the last 100 steps. The number of middle evaluation steps depends on the number of all evaluation steps and the whole number of training steps since the SAC experiment has more steps than the TD3 experiment. From this table, the mean values of total throughput and RSRQ seem to increase in the last steps more than in the first and middle steps. For example, in the second evaluation environment in the TD3 experiment:

$$MeanTotalThroughput_{first100steps} < Mean_{middlesteps} < Mean_{last100steps}$$

$$155,023 < 166,087 < 194,326$$

_	**First 100 steps**	**middle steps**	**Last 100 steps**
TD3 Eval1 throughput	139,829	153,713	153,104
SAC Eval1 throughput	134,182	140,322	149,802
TD3 Eval1 RSRQ	70,623	77,083	77,983
SAC Eval1 RSRQ	95,713	76,240	85,132
TD3 Eval2 throughput	155,023	166,087	194,326
SAC Eval2 throughput	133,505	151,933	176,557
TD3 Eval2 RSRQ	87,769	93,588	94,330
SAC Eval2 RSRQ	83,958	93,519	131,181

Table 5.8: Mean values for first and last 100 steps and the middle steps of total throughput and RSRQ from both evaluation environments for both SAC and TD3 experiments.

However, the mean total RSRQ of the first 100 steps in the first evaluation environment in the SAC version is higher than in the middle and last 100 steps. At the same time, the mean value of the last 100 steps is higher than the middle steps. That means that the RSRQ can be further optimized to be equal to or higher than the first 100 steps.

The number of handovers

The average number of handovers after calling the first 100 evaluation steps in the first evaluation environment in the SAC version is 20 handovers, and for the last 100 evaluation steps is 24. For the TD3 version, the average number of handovers in the first 100 steps is 21, and for the last 100 steps is 26. By comparing these two numbers in the last 100 steps in both experiments, the SAC algorithm could slightly optimize the number of handovers since $25 < 26$. However, that is not clear enough from these experiments. After calling both eval environments in both experiments, the plots of the handovers are shown in the appendix C.

5.3.3 Different network setup results

Some experiments have been done on multiple network situations to evaluate the effect of changing different network parameters on the performance of the RL agent. Experiment A2A4_1 had a constant user speed during the training session, while experiment A2A4_2 had a maximum and minimum user speed variation. It was repeated three times to evaluate the effect of changing the base station locations. A2A4_6 focused on high-speed situations, and A2A4_7 has no variation in the number of users, and the maximum and minimum user speeds are equal. All these experiments had the total throughput as a reward function.

Constant speed situation

After running the A2A4_1 experiment and training the SAC agent over almost 7000 steps. The agent started to converge toward a specific value for both attributes after 2000 steps, as shown in Figure 5.13. This figure shows the actions taken by the agent in every training step from the beginning to the end of the training session.

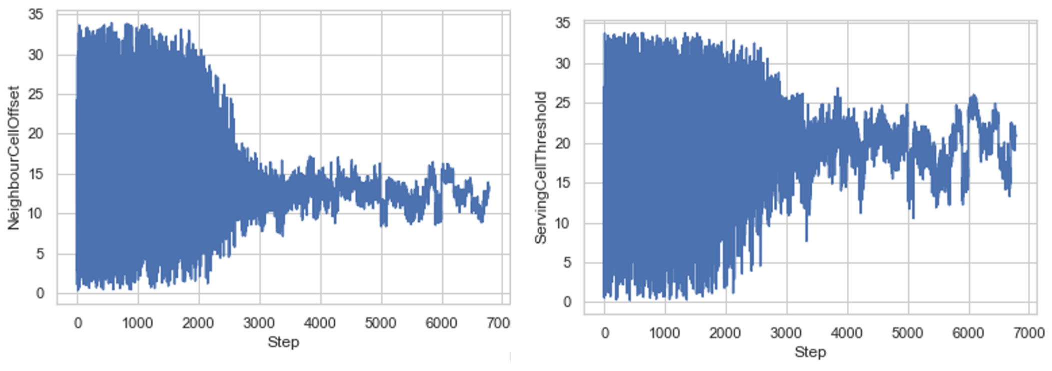


Figure 5.13: Actions taken by the RL agent in A2A4_1.

The agent begins to be deterministic after 4000 steps, where it seems to pick a specific value for each situation. Figure 5.14 zooms the actions picked from steps 4000 to 6500. The actions are changed when changing the environment parameters since the simulation duration and the number of users are changed every 100 steps, and the users' start positions change every ten steps. The shift between the attribute values is clearly shown in Figure 5.15, where the values are mainly changed every 100 steps. The NeighbourCellOffset value is between 8 and 16, and the ServingCellThreshold value is between 15 and 25.

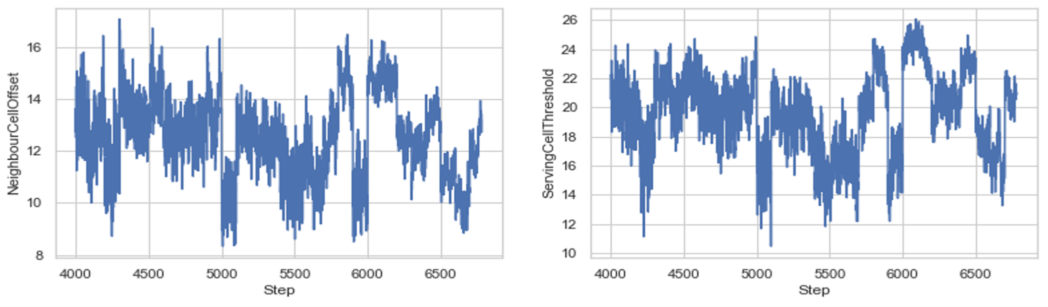


Figure 5.14: Actions between 5000 and 6500 steps in A2A4_1.

Different base station positions

The training actions produced by A2A4_2a and A2A4_2b are shown in Figure 5.16. In this figure, A2A4_2a plots are for the experiment done on the base station positions shown in the third plot in Figure 5.1 and A2A4_2b plot is for the experiment with base station positions shown in the first plot in Figure 5.1.

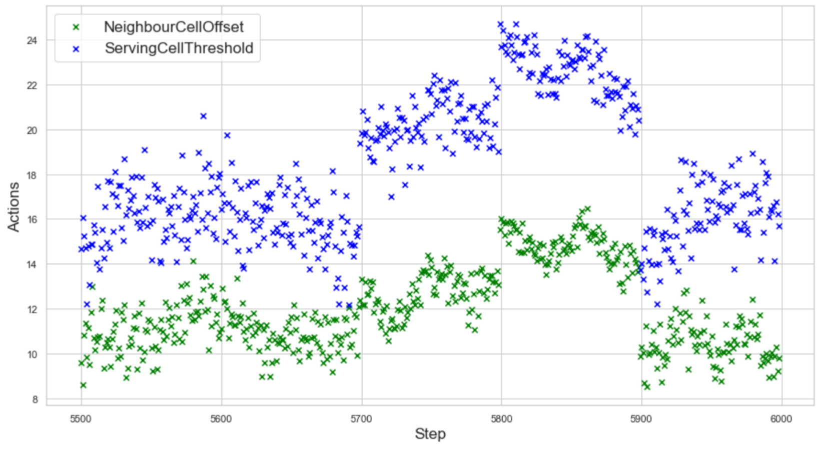


Figure 5.15: Actions between 5500 and 6000 steps in A2A4_1.

The Standard Deviation (SD) for NeighbourCellOffset in A2A4_2a is 6.63, and for A2A4_2b is 6.97. While SD for ServingCellThreshold in A2A4_2a is 7.86 and for A2A4_2b is 8.92. That means that $SD_{A2A4\_2a} < SD_{A2A4\_2b}$ thus A2A4_2a converges before A2A4_2b. In addition, for both experiments:

$$SD_{NeighbourCellOffset} < SD_{ServingCellThreshold}$$

Which means that the agent decided the direction it will choose for NeighbourCellOffset before ServingCellThreshold.

The actions chosen by the agent in both cases are different for both experiments. From step number 4000 to the last step, the mean value of the values produced by the agent are shown in Table 5.9

_	**NeighbourCellOffset**	**ServingCellThreshold**
A2A4_2a	14.47	26.22
A2A4_2b	9.69	31.67

Table 5.9: A2A4_2a and A2A4_2b mean values from step 4000 to the last training step.

The evaluation rewards of the first evaluation environment in A2A4_2b are improved, but the improvement is unclear in A2A4_2a after the training. All rewards received from the evaluation environment for both experiments are shown in Figure 5.17. The average of the rewards for the first and last 50 evaluation steps and their differences for both experiments and their evaluation environments are presented in Table 5.10.

For the first evaluation environment, the difference of the A2A4_2b averages is higher than it is in A2A4_2a, and that shows how the throughput value is better improved in A2A4_2b. On the other hand, the difference of the averages in the second eval environment in A2A4_2b is more extensive than in A2A4_2a. The minimum reward value in A2A4_2a is higher than the minimum reward value in A2A4_2b.

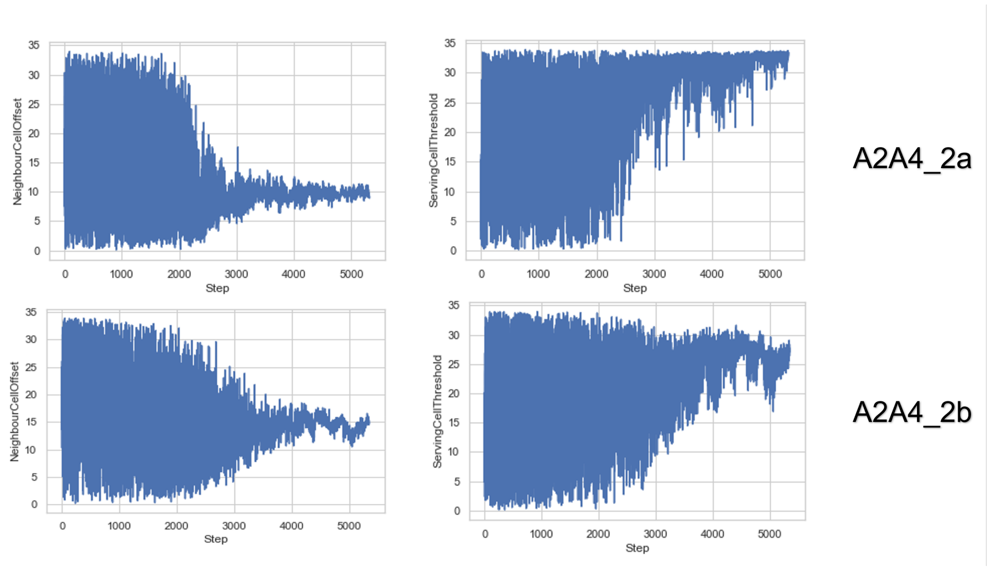


Figure 5.16: Actions for A2A4_2 experiment. The plots to the left are the values of NeighbourCellOffset, and the plots to the right are ServingCellThreshold values.

_	First 50 steps	Last 50 steps	Last - First
A2A4_2a Eval1	2.11	2.18	0.07
A2A4_2b Eval1	1.93	2.42	0.49
A2A4_2a Eval2	3.63	4.19	0.57
A2A4_2b Eval2	2.42	4.25	1.83

Table 5.10: A2A4_2a and A2A4_2b reward averages for the first and last 50 steps and their differences.

In A2A4_2b, the agent found a solution that improved the performance of both evaluation environments, but that was not the case for A2A4_2a, where the first evaluation environment didn't show a real throughput improvement. The second one showed a better improvement. That shows that changing base station positions can lead to different agent performance, even though all other network parameters are the same. It has affected the attribute values chosen by the agent, the convergence speed, and the change of evaluation rewards during the training in these two experiments.

High speed situations

The A2A4_6 experiment ran on longer simulation duration, a bigger number of users, and higher user speed. This experiment had just 2422 steps, where the agent could converge toward a specific value earlier than the other experiments. The actions chosen by the agent during the training step are shown in Figure 5.18. The NeighbourCellOffset action curve converges before the ServingCellThreshold curve and the SD for both steps between 1500 and 2422.

$$SD_{NeighbourCellOffset} = 1.74 < 4.24 = SD_{ServingCellThreshold}$$

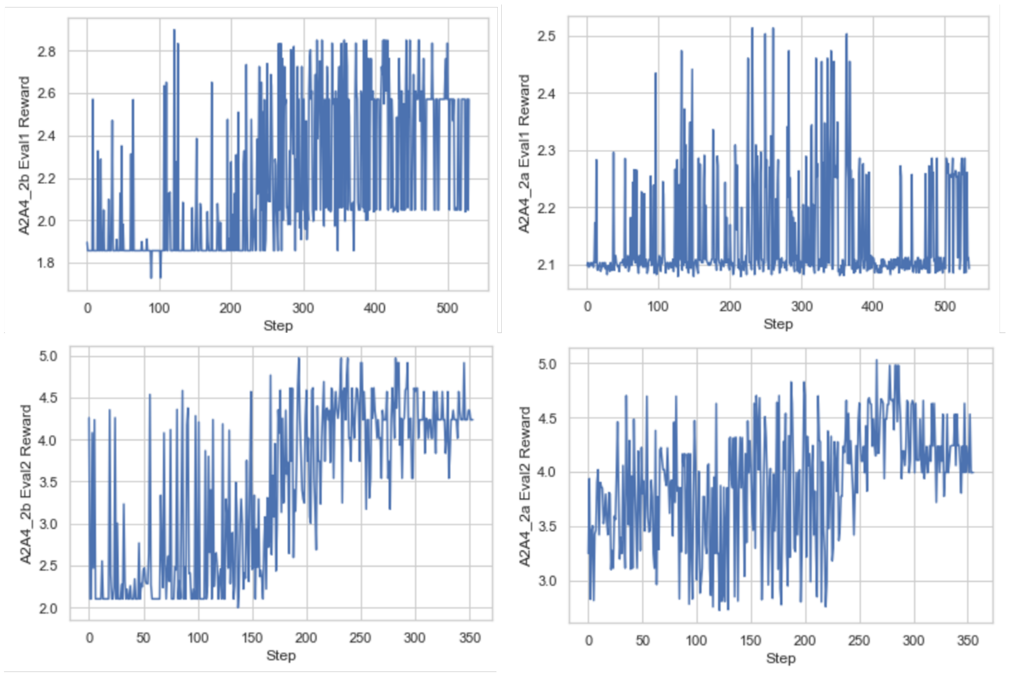


Figure 5.17: Rewards from the evaluation steps for A2A4_2a and A2A4_2b experiments.

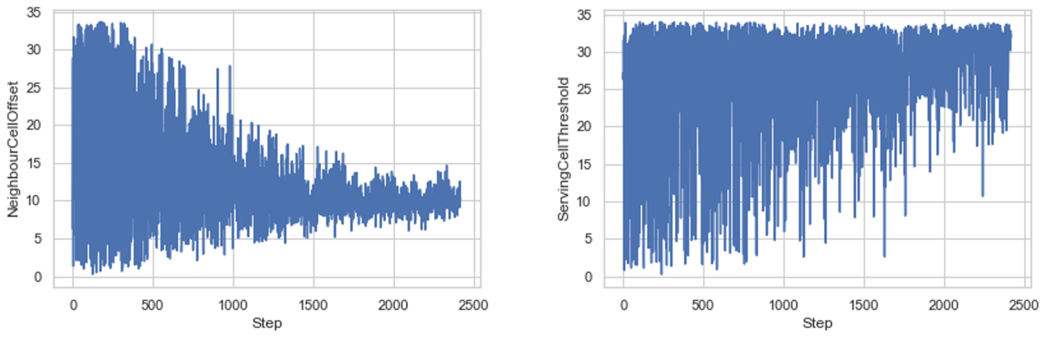


Figure 5.18: The training actions for A2A4_6.

More zoomed view on the actions between step number 2000 and 2422 in Figure 5.19. The NeighbourCellOffset seems to converge to 10, and the mean value for this period is 10.11. ServingCellThreshold has a lower determined solution of around 30, and its mean value for this period is 29.85. The longer simulation duration can be the reason that makes the RL agent explore more results from the simulator, like finding more accurate throughput and RSRQ values. The reward function of this experiment depends on the total throughput and the total RSRQ received by all UEs in each step. That could also be a reason for faster guidance.

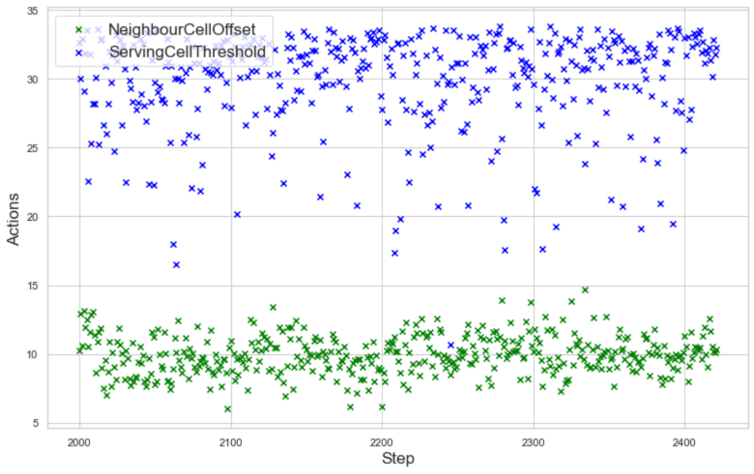


Figure 5.19: Zoom over actions for A2A4_6 between steps 2000 and 2422.

The actions chosen by the agent in this experiment are not the best attribute values compared to the second A2A4 grid search experiment shown in Figure 5.3. That can be a result of the changed reward function in this experiment, which includes the total RSRQ value besides the total throughput, and that means that the best scenario is affected by factors other than the total throughput alone. Another reason is that other network parameters changed during the training, like duration, number of users, and positions, than the network setup in the second grid search experiment, which also produces different total throughput values.

Even though the speed in this experiment was constant during the training, the actions taken by the agent don't show an explicit dependency on network parameter changes like A2A4_1. The higher speed situations don't show a special agent performance since the results of this experiment look like the results in the A2A4_2a experiment. The evaluation environments differed from the constant speed in the training environment to evaluate how well the decided action values chosen in one situation are for another. The total throughput and RSRQ values for both evaluation environments are shown in Figure 5.20.

Both total RSRQ and throughput values have been improved after the training, but the first evaluation shows better improvement than the second one, where the differences between averages of the first 50 evaluation steps and the last 50 evaluation steps in both environments are shown in Table 5.11. The chosen attribute values seem to improve the performance of the evaluation environment with users' speed of 50 m/s better than the evaluation environment with users' speed of 40 m/s. That could show that the optimal values for high-speed situations are sub-optimal for situations with lower users' speeds.

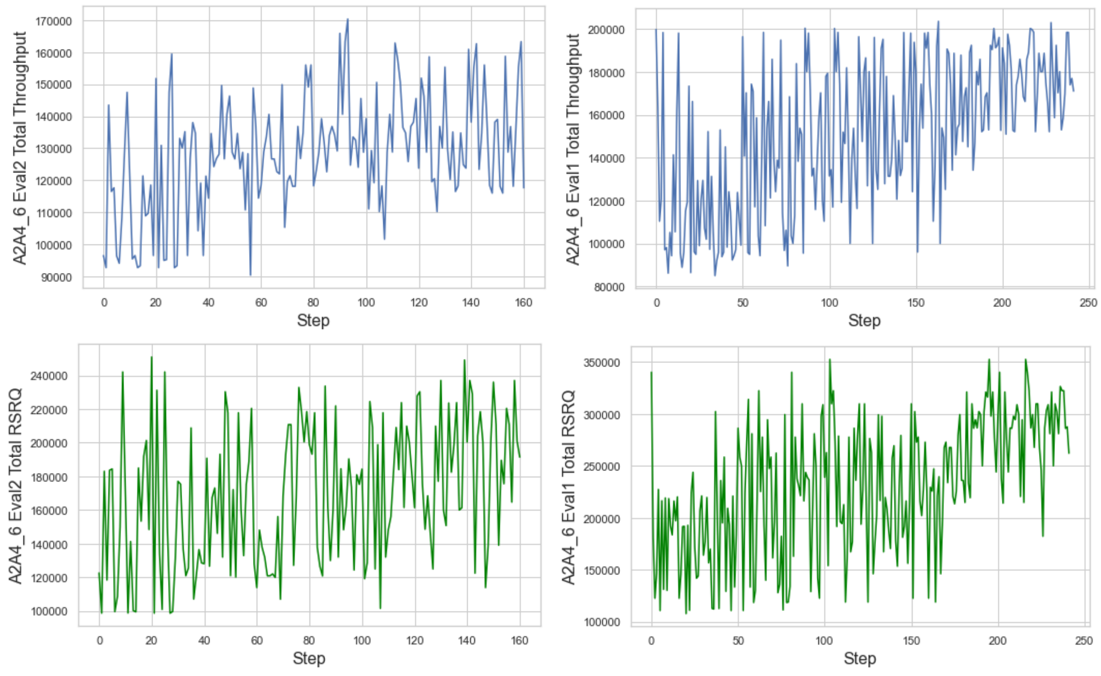


Figure 5.20: Total throughput and RSRQ for the evaluation environment in A2A4_6. The results from the first evaluation environment are shown to the right, and the second one is shown to the left. The blue plots are total throughput plots, and the green plots are total RSRQ.

_	**First 100 steps**	**Last 100 steps**	**Last - First**
Eval1 Total Throughput	117,972	178,268	60,296
Eval1 Total RSRQ	181,497	289,468	107,971
Eval2 Total Throughput	118,214	135,937	17,723
Eval2 Total RSRQ	153,195	188,520	35,325

Table 5.11: A2A4_6 mean total throughput and RSRQ values for first and last 50 evaluation steps and their differences.

5.3.4 Reward functions results

Experiments A2A4_2a and A2A4_4 had the same setup with two different reward functions: total throughput in A2A4_2a and optimize ratio in A2A4_4. A2A4_2a is trained over 5354 steps, and A2A4_4 is trained over 7219 steps. Two main factors can be compared after using these two reward functions. They are the total throughput and the number of handovers. They are measured in each evaluation step to validate the policy found by the agent. Both experiments' training and evaluation actions are shown in appendix D.

The number of handovers happening in each evaluation step in both experiments are shown in Figure 5.21. The average numbers of handovers for the first and last 100 steps are shown in Table 5.12. Since there are more training and evaluation steps in A2A4_4 than A2A4_2a, the last 100 evaluation steps in the first evaluation environment are the steps between 435 and 535 in A2A4_2a, while they are between 620 and 720 in A2A4_4. Therefore, the average number of handovers for the eval steps between 435 and 535 will also be compared in this table as the middle steps. After the same number of steps, both experiments had almost the same number of handovers. In addition, the experiment with the optimize ratio as a reward had several handovers after more steps.

In the second evaluation environment, the steps between 256 and 356 in A2A4_4 (which are the last 100 steps in A2A4_2a) are the middle steps. The experiment with the total throughput as a reward could improve the number of handovers better than the experiment with optimize ratio after the same number of steps. However, the number of handovers is decreasing after several steps in A2A4_4, which uses optimize ratio. This shows some optimization in the number of handovers, but that is not improved before analyzing the changes in the throughput values.

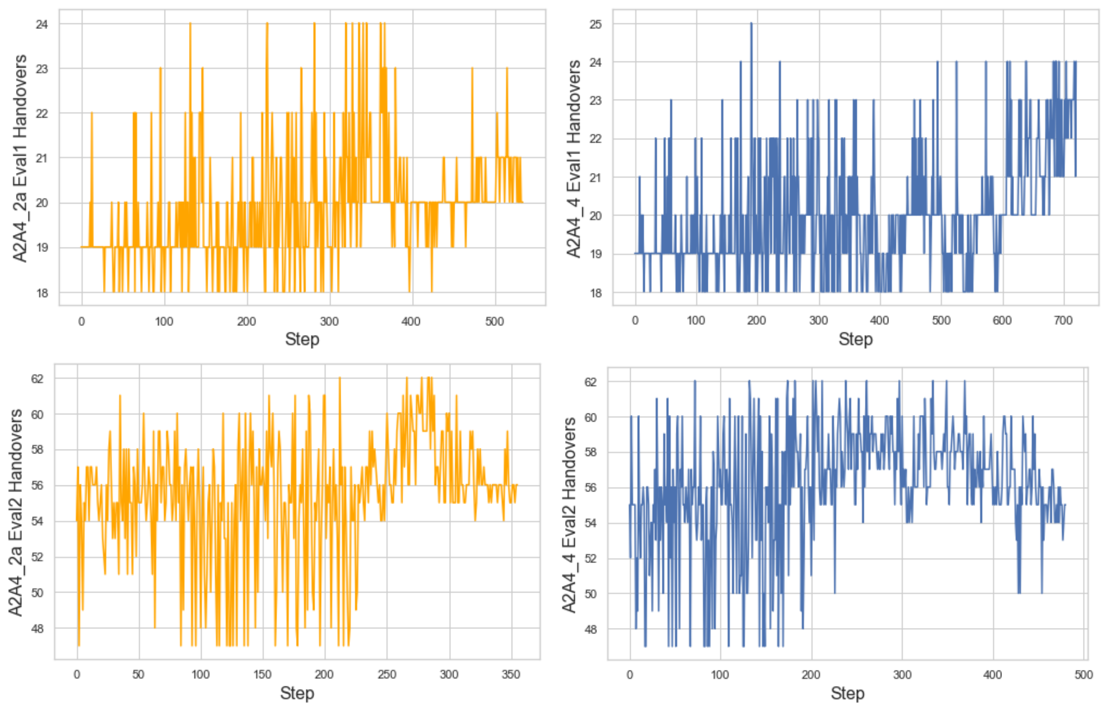


Figure 5.21: The number of handovers in A2A4_4 (blue curves) and A2A4_2a (Orange curves) in both evaluation steps

_	First 100 steps	middel steps	Last 100 steps
A2A4_4 Eval1	19	20	22
A2A4_4 Eval2	54	58	56
A2A4_2a Eval1	19	-	20
A2A4_2a Eval2	55	-	57

Table 5.12: Mean value of Handover measured from the evaluation environments in A2A4_4 and A2A4_2a experiments for the first 100 eval steps and last 100 eval steps. The middle steps are the steps between 435 and 535 for A2A4_4 eval1 and between 256 and 356 for A2A4_4 eval2

Figure 5.22 shows the maximum throughput values received from a base station to its connected users in each eval step of the first evaluation environment for both experiments. From the plots, the maximum throughput is improved from step number 590 in A2A4_4, while it is improved after step number 320 in A2A4_2a. That means it has been improved using the total throughput reward function faster than the optimize ratio experiment. Table 5.13 shows the average of the maximum throughput received. As a result, the maximum throughput is improved in A2A4_4 with the optimize ratio as a reward better than A2A4_2a with total throughput as a reward in this experiment. That means that the total throughput reward function can maximize the maximum throughput value.

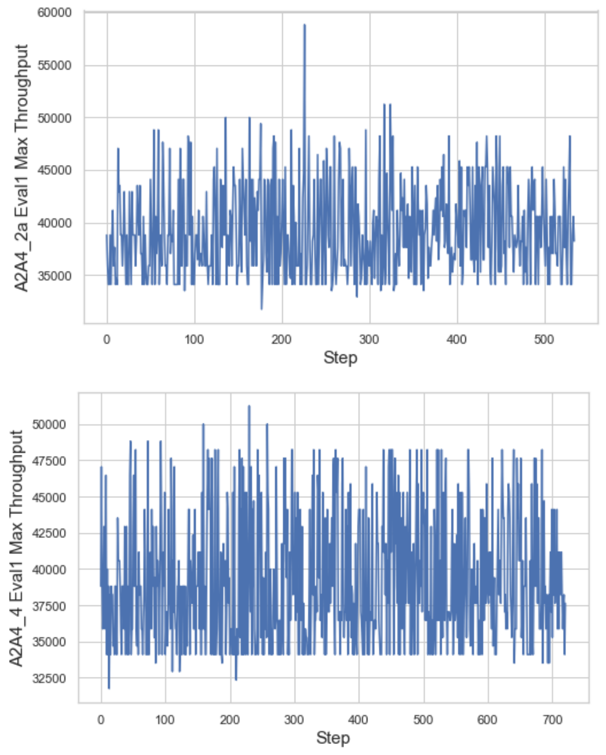


Figure 5.22: The maximum throughput received from one base station to its connected UEs in each evaluation step at the first evaluation environment for A2A4_4 and A2A4_2a

_	**First 100 steps**	**Steps between 400 and 535**	**Difference**
A2A4_4	38,655	39,905	1,250
A2A4_2a	38,502	39,857	1,355

Table 5.13: Maximum BS throughput averages for the first 100 steps and steps between 400 and 535 of A2A4_4 and A2A4_2a first evaluation environment

The number of handovers has increased in both experiments after the same number of steps with almost the exact grade, but the maximum throughput received from a base station to its connected UEs is better improved with the optimize ratio as a reward function than the experiment with total throughput as a reward function. This means that the optimize ratio is a better reward function in this case.

5.3.5 A3 results

In **A3_1**, the actions taken by the agent during the training steps are shown in Figure 5.23. They show that the agent picks the maximum values for Hysteresis and TimeToTrigger, which means that the agent waits as long as possible to trigger the handover. By comparing the results with the grid search heat maps shown in figures 5.4 and 5.5, these values are not the optimal values.

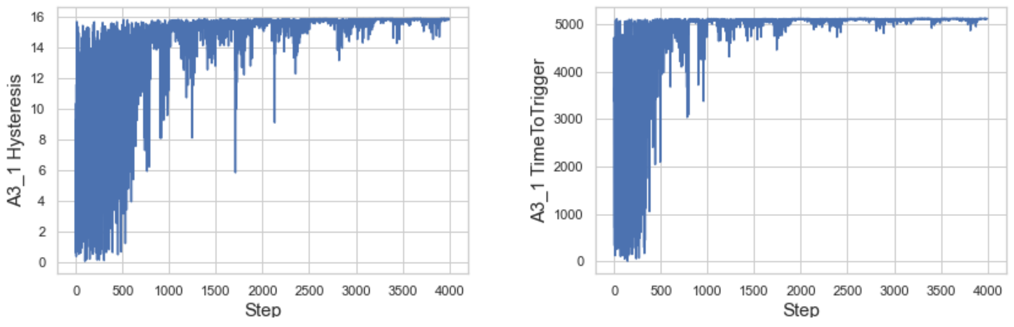
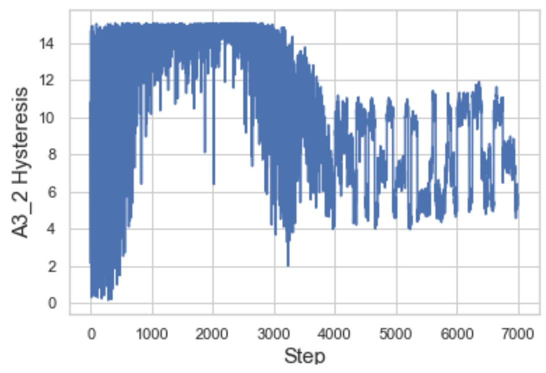


Figure 5.23: A3_1 Training actions (Hysteresis and TimeToTrigger)

On the other hand, the Hysteresis and TimeToTrigger actions taken by the agent during the training in **A3_2** are displayed in Figure 5.24. The agent has been trained on 6995 steps. The lower value of TimeToTrigger is picked from step number 2000 to the end, while the agent shows an adaptive behavior in the Hysteresis value after 4000 steps. The evaluation Hysteresis actions for both evaluation environments are shown in Figure 5.25. This figure shows that the trained policy picks different Hysteresis values for different network setups. There are several steps in the first evaluation environment than in the second one because the first is called every ten training steps, and the other is called every 15. In the first evaluation environment, the mean Hysteresis value is 10.65 for the last 100 steps, while it is 6.68 for the last 100 steps of the second evaluation environment. Even though the agent could pick an adaptive Hysteresis action for each situation, it is not clear enough that this solution is optimal.



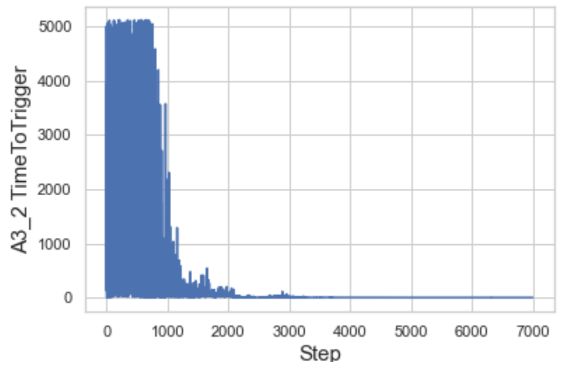


Figure 5.24: A3_2 Training actions (Hysteresis and TimeToTrigger)

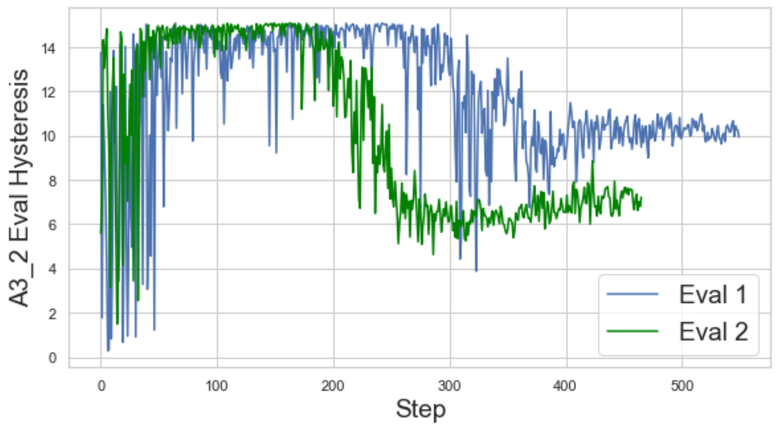


Figure 5.25: Evaluation Hysteresis values produced after calling both evaluation environments for 550 evaluation steps

Figure 5.5, which represents the second A3 grid search experiment, shows some good total throughput achieved when Hysteresis is 5 or 6, and TimeToTrigger is between 512s and 1280s. The minimum TimeToTrigger value is not the optimal value. In addition, a meager value of TimeToTrigger causes several handovers since it doesn't wait any time to check that the RSRP is stable and more significant than the neighbor base station RSRP.

Even though the chosen actions in A3_2 are not optimal in comparison with the heat map in Figure 5.4 too, they have shown some improvements in the total throughput and RSRP values in both evaluation environments. The Total throughput and RSRP and the number of handovers are displayed in Figure 5.26. The three plots have almost the same pattern and show that the agent achieved a stable behavior after 400 evaluation steps which are 4000 training steps. The comparison between the average total throughput for the first 100 eval steps, the last 300 eval steps, and the differences between them are shown in Table 5.14. The number of handovers increased by 7.3, but the total throughput and RSRP have increased.

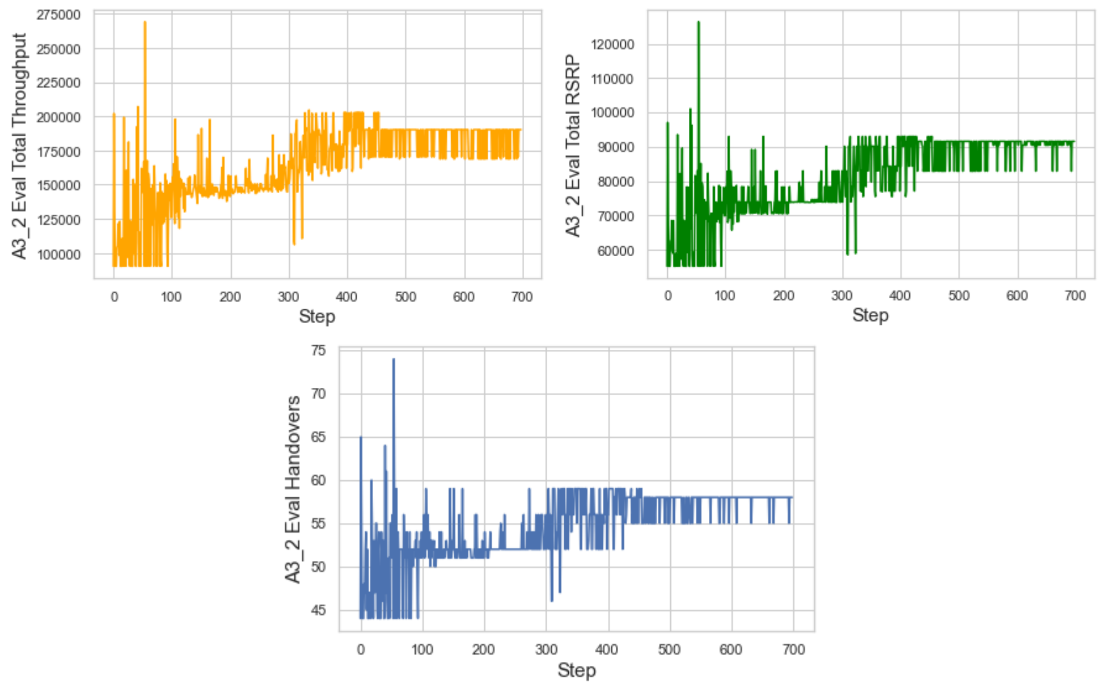


Figure 5.26: Number of handovers, total throughput and RSRP measured from the first evaluation environment

_	First 100 steps	Last 300 steps	Last - First
Total Throughput	125,625	183,554	57,929
Total RSRP	67,986	89,974	21,988
Handovers	50.2	57.5	7.3

Table 5.14: Mean value of handovers, total Throughput and RSRP measured from the first evaluation environment in A3_2 experiment for the first 100 eval steps and last 300 eval steps

5.3.6 General and Adaptive optimal solutions

Some experiments have shown that the RL agent could find optimal handover attribute values for each network scenario. Some other experiments showed that it couldn't pick a specific value for each scenario, but it could converge toward general attribute values, improving the performance of evaluation environments.

General optimal solution

Some experiments showed that the RL agent policy is trained to choose one general action for all network situations like A2A4_2c. A2A4_2c has shown that the agent policy is built to converge toward almost one general attribute value for both handover attributes in all network situations. It has also shown that the total throughput and RSRQ are improved in both evaluation environments. That general action could be a good solution for all network situations but not the optimal one for each situation.

The actions found for both evaluation environments in A2A4_2c are shown in Figure 5.27 where the average of ServingCellThreshold of the last 100 evaluation steps in both evaluation environments is almost the same. They are $32.89 \approx 32.8$. They are 9.29 and 9.11 for NeighbourCellOffset. By comparing with Figure 5.2, when ServingCellThreshold is 32 or 34, the total throughput is good with NeighbourCellOffset as 15. It is lower when NeighbourCellOffset is 7 or 11 since this grid search experiment does not test nine. That is also the case in Figure 5.3. This means that for these two situations, the value of NeighbourCellOffset chosen by the agent in A2A4_2c is not optimal.

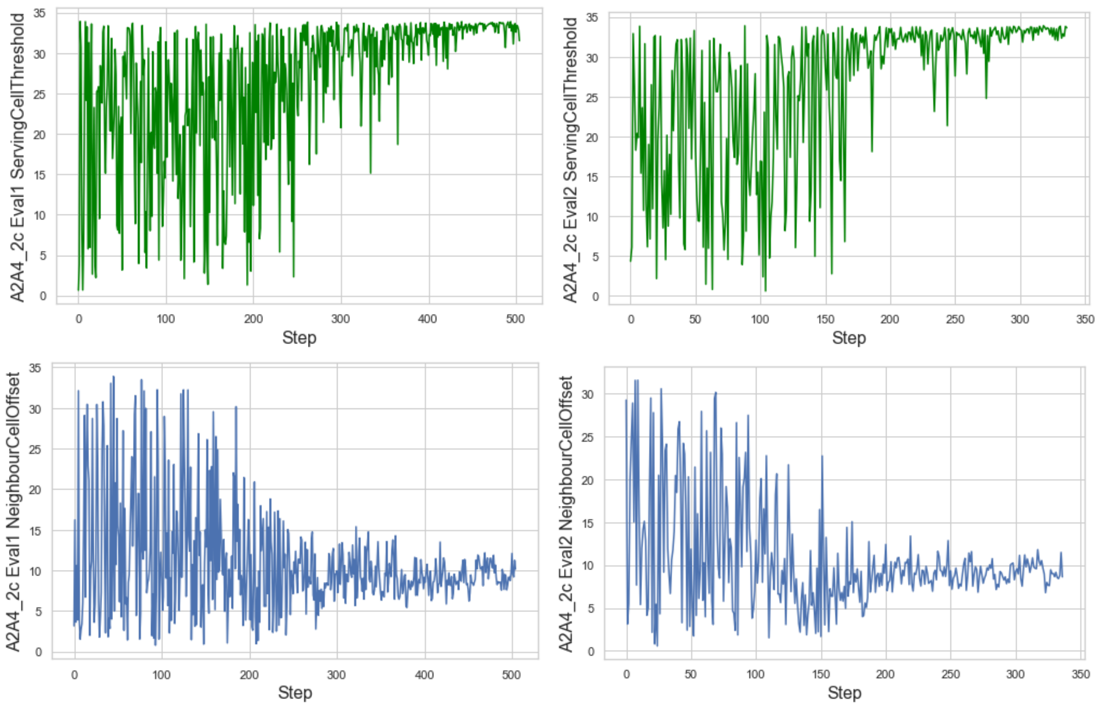


Figure 5.27: A2A4_2c evaluation actions from both evaluation environments

The SAC version in A2A4_5 also converged toward general action values: ServingCellThreshold ≈ 15.6 and NeighbourCellOffset ≈ 28. These values produce good total throughput values, as shown in the heat maps in 5.2 and 5.3.

Adaptive optimal solution

Some other experiments have shown adaptive behavior for different network situations. That is shown clearly in the TD3 version of the A2A4_5 experiment in Figure 5.9, which shows the adaptive actions when changing the network parameters like users' maximum and minimum speeds and simulation duration. The A2A4_1 experiment, which uses SAC, also showed an adaptive agent behavior for both handover attributes. That is shown in Figure 5.15. The adaptivity is also apparent in A3_2 for the Hysteresis attribute. The agent could find a different Hysteresis value for different network scenarios, as shown in Figure 5.25. This figure shows different Hysteresis values chosen by the RL agent in the different evaluation environments with two network scenarios.

To analyze how adaptive the actions are depending on the network parameter changes, experiment A2A4_7 is done with three different speeds and three numbers of users while the other parameters are constant. NeighbourCellOffset and ServingCellThreshold actions taken by the agent in different speed situations are shown in Figure 5.28. This figure shows that the agent picked different NeighbourCellOffset in different speed situations, while the ServingCellThreshold value is almost the same in all situations, which is ≈ 31.

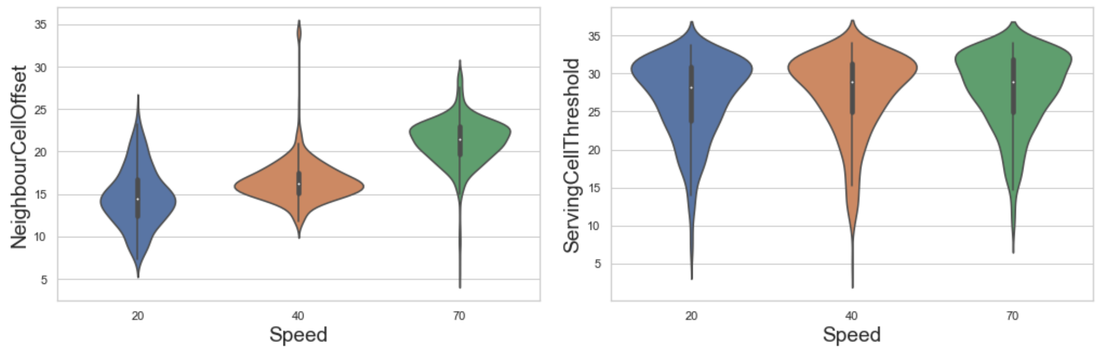


Figure 5.28: A2A4_7 training actions for different speed situations for the last 1500 training steps

For the high-speed grid search experiment shown in Figure 5.3, the chosen actions by the agent, which are NeighbourCellOffset ≈ 23 and ServingCellThreshold ≈ 31, produced a very high total throughput, which means that this is an optimal solution for this situation.

For lower speed situations, the chosen NeighbourCellOffset ≈ 15 and ServingCellThreshold ≈ 31 also gave a high total throughput value. That means the agent could find the optimal values for different network situations. Additionally, the max total throughput values shown in figures 5.2 and 5.3 are not constant since the network setup for each step is unique because the user movements are random in each step. When the grid search experiments were repeated, the heat map produced had different maximum total throughput values, but the pattern of values was almost the same. The heat maps from the repeated grid search experiments are shown in appendix E.

5.3.7 Results summary

Both A2A4 and A3 handover algorithms are used in the SAC experiments. The handover algorithm that showed a more reasonable agent behavior is the A2A4 algorithm. Its attribute values chosen by the agent were compared with the best attribute values achieved in the grid search experiments. The agent showed different behavior for both attributes in A2A4. In some of the A3 experiments, the agent picked the maximum attribute values while it could find an adaptive Hysteresis value with the minimum TimeToTrigger value in other experiments.

When SAC and TD3 algorithms have been applied to the same experiment with the A2A4 handover algorithm, the TD3 had shown a faster and more adaptive performance than the SAC algorithm since it had tested a wider variety of the actions at the beginning before it converged toward the specific handover attribute values. The TD3 algorithm could adapt to the scenario changes better than the SAC algorithm. However, in other experiments, SAC showed an adaptive behavior, indicating that it could also follow the simulated network changes. In other situations, the SAC agent converged toward specific attribute values for all network scenarios. After comparing with the grid search experiments, these values generally give good throughput values but not specifically for each situation.

The attribute values chosen by both RL algorithms showed a good performance in the grid search experiments but not the optimal one. The reason for that can be that the simulated network is unique in every step because of the random user movements. Therefore, the chosen attribute values can be the best regardless of how the users move since the total evaluation throughput and RSRQ are improved. The reward functions used have shown an increase in the total throughput and RSRQ values, but the optimize ratio reward function has shown better improvements. However, the number of handovers is increasing when using both reward functions, which can be acceptable since the user is getting higher data throughput and better signal quality.

Chapter 6

Conclusion and future work

6.1 Conclusion

The main goal of this project is to investigate the potential of RL to optimize mobile network performance. The main direction chosen is to optimize the performance of the handover algorithm. That means optimizing the handover algorithm attributes to deliver the best data throughput, signal power, and quality to the users while moving around between different base stations with different speeds. Since it is difficult to apply the research on real-world networks, using a network simulator was necessary. The open-source ns-3 simulator has been used to build the simulated networks used in this project.

The Telenor research team has started to work on optimizing mobile networks using RL. They have not specifically focused on the handover algorithm in their work but have implemented a constant simulated network for handover using the ns-3 simulator. In this project, this simulated network has been adjusted to be a dynamic simulated network. Its structure and parameters are changeable and can be updated from the RL environment.

Two handover algorithms are used, and they are A2A4 RSRQ and A3 RSRP. The handover decisions made by them are taken based on different criteria. The A2A4 algorithm takes its decisions based on comparing the RSRQ value received by the user from the current serving base station to the neighbor base station with the best RSRQ. It uses two values for that comparison which are ServingCellThreshold and NeighbourCellOffset. These values are optimized by the RL agent in this project. On the other hand, the A3 RSRP algorithm compares the RSRP, which is the signal power using the TimeToTrigger and Hysteresis attributes. These attributes are also optimized by the RL agent.

6.1.1 The first research question

Which handover algorithm and attribute values will be used for the best optimization?

After some experiments on both handover algorithms, the A2A4 RSRQ handover algorithm has shown more adaptivity in its attributes than the A3 RSRP algorithm. Some of the experiments done on the A3 RSRP algorithm showed that the RL agent converges toward the maximum values for both its attributes, and some of them showed that it converges toward the smallest value for TimeToTrigger and it picks more adaptive value for Hysteresis. The difference between these results was the reward function used by the RL agent. The experiment that converged toward maximum TimeToTrigger and an adaptive Hysteresis showed an improvement in the total throughput and RSRP values after some training steps. In contrast, the other experiment had not shown any improvements. That means that the RL agent could not improve the performance of the A3 RSRP algorithm in all cases.

On the other hand, the RL agent could show visible improvements in total throughput and RSRQ values in almost all A2A4 RSRQ experiments that use different reward functions. It had adaptive behavior in some cases and general behavior in other experiments. The attribute values chosen by the agent are compared to those that gave the best performance in the grid search experiments. These different attribute values produced a good total throughput value in low and high-speed situations. They were not the values with the best total throughput, but that can be explained by the uniqueness of the simulated scenarios in each step because of the random users' movements and positions. The optimal total throughput value can vary between the different unique network scenarios, and the handover attributes value can be the best for all users' movement scenarios.

The RL agent has shown a better performance using the A2A4 RSRQ algorithm than the A3 RSRP in the experiments done in this project. There are different optimal handover attribute values for every network situation depending on the network parameters like the number of base stations, their positions, the number of users, their speed, their positions, and other network parameters.

6.1.2 The second research question

Does the soft actor-critic model provide advantages over the Twin-Delayed Deep Deterministic model for tackling the handover problem?

The Soft actor-critic (SAC) model has been used in different experiments, and its performance has been compared with the performance of the Twin-Delayed Deep Deterministic (TD3) model. They have been applied in two experiments with the same RL environment setup using the A2A4 RSRQ handover algorithm. The TD3 algorithm was faster than SAC and could decide the handover attribute values for each scenario after fewer steps than SAC. In this experimental setup, the SAC algorithm could not find an adaptive solution for every network situation. It converged toward one ServingCellThreshold value and one NeighbourCellOffset value for all network situations. On the other hand, the TD3 algorithm showed an explicit adaptive behavior and chose different attribute values for every network situation. The total throughput and RSRQ measured from the evaluation environments showed that TD3 could improve them faster than SAC with more stable behavior. Both could improve the total throughput and RSRQ to almost the same level, but TD3 achieved that value earlier.

In other experiments, the SAC algorithm showed an adaptive behavior for different network scenarios. The adaptive behavior includes handover attribute values in some experiments or just the NeighbourCellOffset value in others. That means that both RL models can optimize the handover attribute values in different network scenarios. However, the TD3 model is faster and more stable to make its decisions than the SAC model. The randomization used by the SAC agent to choose its actions can be a reason for its delayed decisions since it tries different actions before it makes its decision. The TD3 agent is more directive and can decide earlier in this case.

6.1.3 Summary

The Soft actor-critic algorithm has been applied in most of the experiments conducted in this project to analyze its behavior to optimize the performance of the handover algorithm. Optimizing the handover attribute values and applying several handovers could improve the total data throughput and signal quality, and power received by all UEs in each simulation step. SAC has been compared with the TD3 algorithm, which showed a better and faster performance. The two handover algorithms have been applied in the experiments and analyzed. The RL agent could optimize the A2A4 RSRQ more clearly than the A3 algorithm. The attribute values chosen by the agent could improve the total throughput, total RSRQ, and total RSRP values.

6.2 Future Work

There are many directions to be chosen further to continue working on this research. It is possible to continue the work on optimizing the handover process or optimizing other directions in mobile networks like scheduling, resource allocation, load balancing, beam management, and power saving.

6.2.1 The handover process

The A2A4 RSRQ and A3 RSRP algorithms have been used in this work, but it is possible to build a new handover algorithm in the ns-3 simulator with different attributes. A relevant idea can be to combine both A2A4 and A3 handover algorithms. That can happen by making the handover decision based on the values of both RSRQ and RSRP. Thus the four handover attributes will be used in this algorithm, which can be optimized using reinforcement learning. A new implementation of the handover algorithm can also be built to include the throughput value as a decision criterion inside the handover algorithm.

6.2.2 Other RL models

Several RL models can also be used and compared to the SAC and TD3 models. Some of the RL algorithms that can be used instead are DQL or DDPG. The RL system structure itself can also be modified with multiple critic networks or implementing new critic and actor neural networks that are designed to fit the observation and give more weight to some important values. These networks can include several layers depending on the importance of each value. A custom critic network was built and tested during this project but was not used in the experiments since it used more CPU capacity.

6.2.3 More simulated networks

It is also possible to try other simulated network scenarios with different numbers of base stations and with a dynamic number of users. The dynamic number of users means a variation of the number of users in one simulation. That can happen when some users leave the simulated area, some new users come in, or when the same user goes away and then comes back. Currently, the experiments have a constant number of users in each simulated step, which means that the user does not leave the simulated area and no new uses come in. All users now have the same speed in each simulated step, but it is possible to vary their speed in one simulated step to include all cases in the same step. That means that some users move slowly, and others move faster simultaneously.

6.2.4 Two RL agents for two different simulated areas

One other idea to be applied further is to have two RL agents for two different simulated areas and analyze the handover process in these two areas. These two areas can be distinct, and they can also overlap. If they overlap, it will be important to observe the agents' performances and how they are affecting each other decisions.

Bibliography

[1] Vegard Edvardsen, Gard Spreemann, and Jeriek Van den Abeele. FORLORN: A framework for comparing offline methods and reinforcement learning for optimization of RAN parameters. In *Proceedings of the 18th ACM International Symposium on QoS and Security for Wireless and Mobile Networks on 18th ACM International Symposium on QoS and Security for Wireless and Mobile Networks*. ACM, oct 2022.

[2] Azza H. Ahmed and Ahmed Elmokashfi. Icran: Intelligent control for self-driving ran based on deep reinforcement learning. *IEEE Transactions on Network and Service Management*, 19(3):2751–2766, 2022.

[3] Bohdan Shubyn, Nazarii Lutsiv, Oleh Syrotynskyi, and Roman Kolodii. Deep learning based adaptive handover optimization for ultra-dense 5g mobile networks. In *2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, pages 869–872, 2020.

[4] Kang Tan, Duncan Bremner, Julien Le Kernec, Yusuf Sambo, Lei Zhang, and Muhammad Ali Imran. Intelligent handover algorithm for vehicle-to-network communications with double-deep q-learning. *IEEE Transactions on Vehicular Technology*, 71(7):7848–7862, 2022.

[5] Xingqiang Cai, Cheng Wu, Jie Sheng, Jin Zhang, and Yiming Wang. A parameter optimization method for lte-r handover based on reinforcement learning. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1216–1221, 2020.

[6] Delin Guo, Lan Tang, Xinggan Zhang, and Ying-Chang Liang. Joint optimization of handover control and power allocation based on multi-agent deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69(11):13124–13138, 2020.

[7] Md Mehedi Hasan, Sungoh Kwon, and Sangchul Oh. Frequent-handover mitigation in ultra-dense heterogeneous networks. *IEEE Transactions on Vehicular Technology*, 68(1):1035–1040, 2019.

[8] Vijaya Yajnanarayana, Henrik Rydén, László Hévizi, Ankit Jauhari, and Mirsad Cirkic. 5g handover using reinforcement learning. *CoRR*, abs/1904.02572, 2019.

[9] Piotr Gawlowicz and Anatolij Zubow. ns3-gym: Extending openai gym for networking research. *CoRR*, abs/1810.03943, 2018.

[10] Hao Yin, Pengyu Liu, Keshu Liu, Liu Cao, Lyutianyang Zhang, Yayu Gao, and Xiaojun Hei. ns3-ai: Fostering artificial intelligence algorithms for networking research. pages 57–64, 06 2020.

[11] Mariam Aboelwafa, Ghada Alsuhli, Karim Banawan, and Karim Seddik. Self-optimization of cellular networks using deep reinforcement learning with hybrid action space. 01 2022.

[12] Yunhui Qin, Zhongshan Zhang, Wei Huangfu, Haijun Zhang, and Keping Long. Cooperative resource allocation based on soft actor-critic with data augmentation in cellular network. *IEEE Wireless Communications Letters*, pages 1–1, 2022.

[13] Alisson Henrique Kolling Ricardo Bedin Grando Marco Antonio de Souza Leite Cuadros Daniel Fernando Tello Gamarra Junior Costa de Jesus, Victor Augusto Kich. Soft actor-critic for navigation of mobile robots. 2021.

[14] nsnam.org. ns-3 manual, `https://www.nsnam.org/docs/models/html/lte-design.html#overall-design`, 2022.

[15] Chao Yu, Jiming Liu, and Shamim Nemati. Reinforcement learning in healthcare: A survey. *CoRR*, abs/1908.08796, 2019.

[16] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space, 2015.

[17] Mauricio Fadel Argerich. List of reinforcement learning environments. `https://medium.com/@mauriciofadelargerich/reinforcement-learning-environments-cff767bc241f`, 2019.

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

[19] RICHARD BELLMAN. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[20] Ishii H. Barron, E. N. The bellman equation for minimizing the maximum cost nonlinear analysis: Theory, methods applications, 13(9), 1067–1090. 1989.

[21] Sanchit Tanwar. Bellman equation and dynamic programming. `https://medium.com/analytics-vidhya/bellman-equation-and-dynamic-programming-773ce67fc6a7`, 2019.

[22] Elmar Diederichs. Reinforcement learning - a technical introduction. *Journal of Autonomous Intelligence*, 2:25, 08 2019.

[23] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016.

[24] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[25] Yan Ma, Wenbo Zhu, Michael G. Benton, and Jose' Romagnoli. Continuous control of a polymerization system with deep reinforcement learning. *Journal of Process Control*, 75:40–47, 2019.

[26] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[27] Tanuja Joshi, Shikhar Makker, Hariprasad Kodamana, and Harikumar Kandath. Twin actor twin delayed deep deterministic policy gradient (tatd3) learning for batch process control. *Computers Chemical Engineering*, 155:107527, 2021.

[28] Kristian Hartikainen Aurick Zhou Murtaza Dalal Tuomas Haarnoja, Vitchyr Pong and Sergey Levine. Soft actor critic deep reinforcement learning with real-world robots. `https://bair.berkeley.edu/blog/2018/12/14/sac/`, 2018.

[29] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 1433–1438. AAAI Press, 2008.

[30] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[31] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018.

[32] Hamid Ali, Hammad Majeed, Imran Usman, and Khaled Almejalli. Reducing entropy overestimation in soft actor critic using dual policy network. *Wireless Communications and Mobile Computing*, 2021:1–13, 06 2021.

[33] Hado van Hasselt and Marco A. Wiering. Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279, 2007.

[34] Junpei Zhong and Yanan Li. Toward human-in-the-loop pid control based on cacla reinforcement learning. In Haibin Yu, Jinguo Liu, Lianqing Liu, Zhaojie Ju, Yuwang Liu, and Dalin Zhou, editors, *Intelligent Robotics and Applications*, pages 605–613, Cham, 2019. Springer International Publishing.

[35] Ioan Sorin Comsa, Sijing Zhang, Mehmet Aydin, Jianping Chen, Pierre Kuonen, and Jean-Frederic Wagen. Adaptive proportional fair parameterization based lte scheduling using continuous actor-critic reinforcement learning. In *2014 IEEE Global Communications Conference*, pages 4387–4393, 2014.

[36] Oğuz Sunay Larry Peterson. *5G Mobile Networks: A Systems Approach (Synthesis Lectures on Network Systems), Ebook in `https://5g.systemsapproach.org/arch.html#main-components`*. 2020.

[37] Mojtaba Vaezi and Ying Zhang. *Radio Access Network Evolution*, pages 67–86. 04 2017.

[38] nsnam.org. Ns-3 tutorial, `https://www.nsnam.org/docs/tutorial/html/quick-start.html`, 2022.

[39] nsnam.org. Ns-3 introduction, `https://www.nsnam.org/docs/tutorial/html/introduction.html#introduction`, 2022.

[40] ScienceDirect. Handover, `https://www.sciencedirect.com/topics/engineering/handover`.

[41] Hendrawan Hendrawan, Ayu Zain, and Sri Lestari. Performance evaluation of a2-a4-rsrq and a3-rsrp handover algorithms in lte network. *Jurnal Elektronika dan Telekomunikasi*, 19:64, 12 2019.

[42] Farhana Afroz, Ramprasad Subramanian, Roshanak Heidary, Kumbesan Sandrasegaran, and Solaiman Ahmed. Sinr, rsrp, rssi and rsrq measurements in long term evolution networks. *International Journal of Wireless Mobile Networks*, 7:113–123, 08 2015.

[43] Muhamad Assyadzily, Antonius Suhartomo, and Arthur Silitonga. Evaluation of x2-handover performance based on rsrp measurement with friis path loss using network simulator version 3 (ns-3). In *2014 2nd International Conference on Information and Communication Technology (ICoICT)*, pages 436–441, 2014.

[44] nsnam.org. Handover algorithms in ns-3 manual, `https://www.nsnam.org/docs/models/html/lte-design.html#handover-algorithm`, 2022.

[45] nsnam.org. A2a4 rsrq handover algorithm in ns-3 manual, `https://www.nsnam.org/docs/release/3.19/doxygen/classns3_1_1_a2_a4_rsrq_handover_algorithm.html`, 2022.

[46] Ehab Ahmed Ibrahim, M.R.M. Rizk, and Ehab F. Badran. Study of lte-r x2 handover based on a3 event algorithm using matlab. In *2015 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1155–1159, 2015.

[47] nsnam.org. A3 event handover algorithm in ns-3 manual, `https://www.nsnam.org/docs/models/html/lte-design.html#strongest-cell-handover-algorithm`, 2022.

[48] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[49] Telenor Research team. Forlorn project, `https://github.com/tnresearch/forlorn`, 2022.

[50] nsnam.org. Ltehelper, `https://www.nsnam.org/docs/release/3.29/doxygen/classns3_1_1_lte_helper.html#details`, 2022.

[51] nsnam.org. Random variables in ns-3, `https://www.nsnam.org/docs/manual/html/random-variables.html`, 2022.

[52] tensorflow.org. Sac tutorial tensorflow, `https://www.tensorflow.org/agents/tutorials/7_sac_minitaur_tutorial#policies`, 2023.

[53] tensorflow.org. Reinforcement learning tutorial, `https://www.tensorflow.org/agents/tutorials/6_reinforce_tutorial`, 2023.

[54] tensorflow.org. Pyenvironment, `https://www.tensorflow.org/agents/api_docs/python/tf_agents/environments/pyenvironment`, 2022.

[55] Huda Albizreh Sakka Amini. Rl for handover optimization (an experiment setup in the master project) `https://github.com/hudaal/handoveroptimization`, 2023.

Appendix A

A3_2 results with total throughput reward function

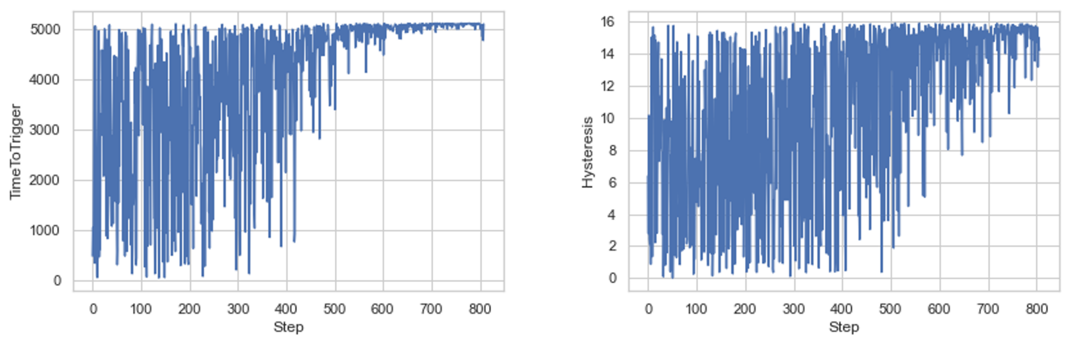


Figure 1: The training actions from A3_2 with total throughput as a reward function. The agent converges toward the maximum values for both A3 attributes.

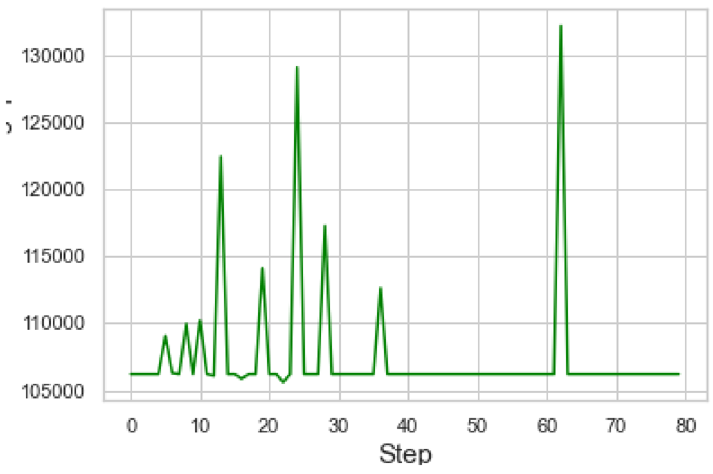


Figure 2: The total throughput from the first evaluation environment in A3_2

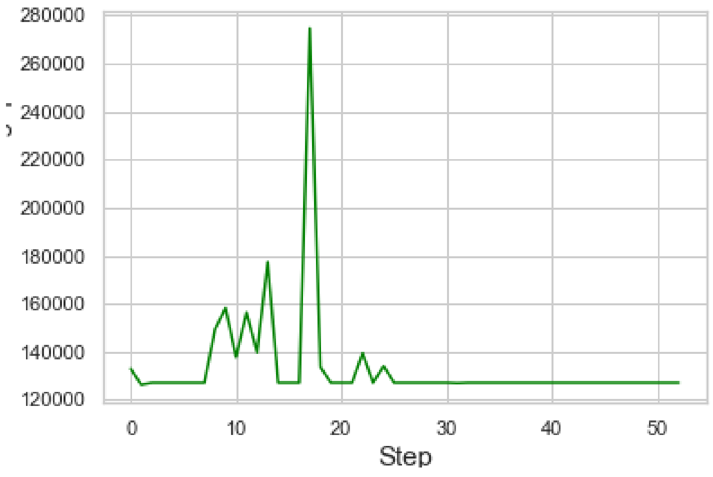


Figure 3: The total throughput from the second evaluation environment in A3_2

Appendix B

Total throughput from the first evaluation environment in A2A4_5 for both TD3 and SAC versions

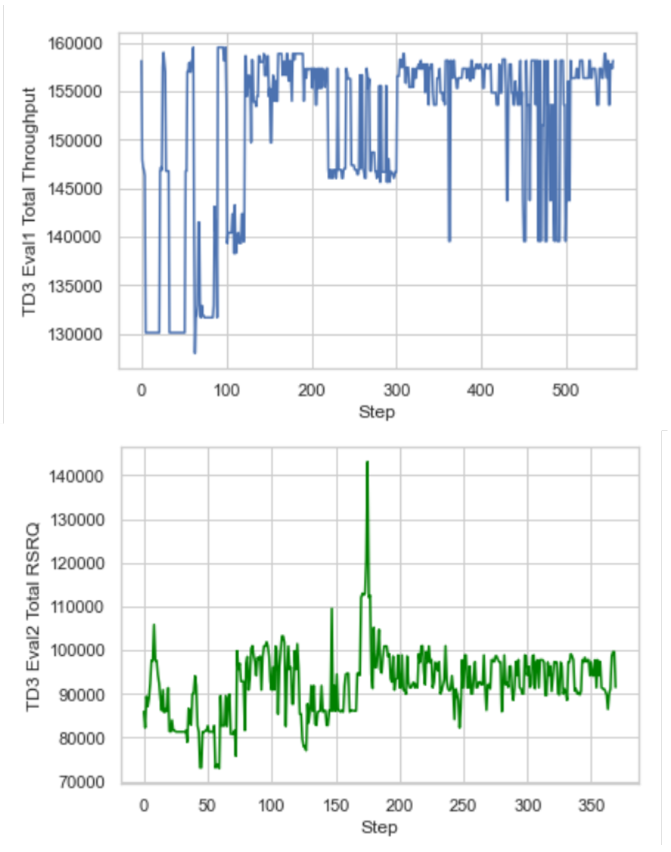


Figure 4: The total throughput and RSRQ from the evaluation environments in TD3 version of A2A4_5

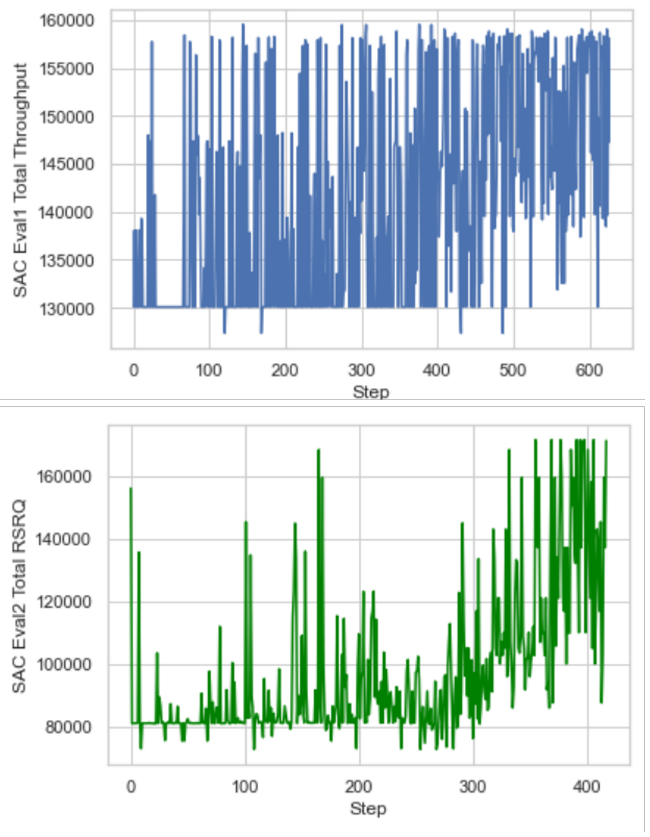


Figure 5: The total throughput and RSRQ from the evaluation environments in SAC version of A2A4_5

Appendix C

Number of handover from the evaluation environments in A2A4_5 for both TD3 and SAC versions

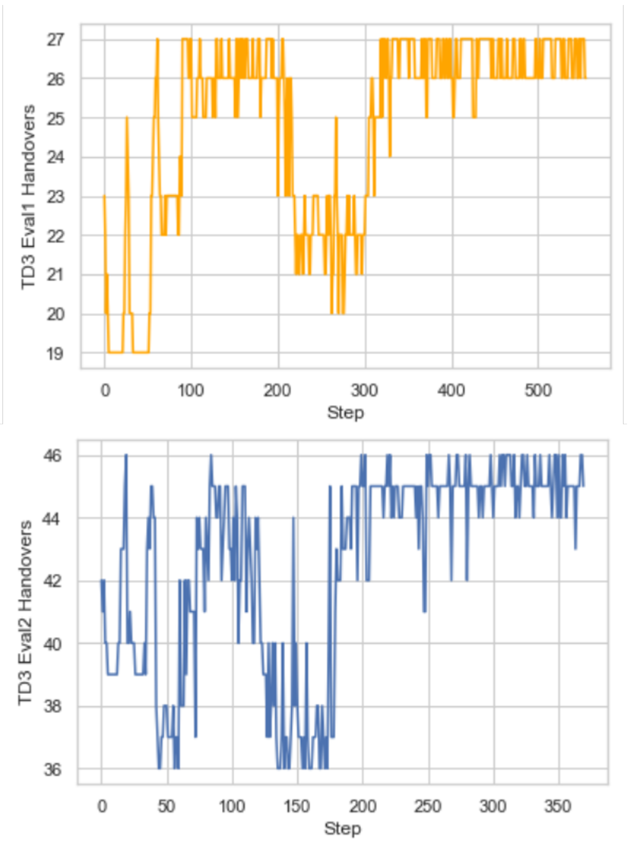
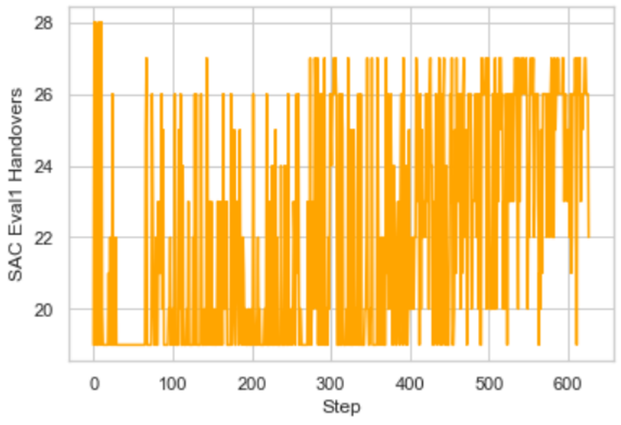


Figure 6: The handovers from the evaluation environments in TD3 version of A2A4_5



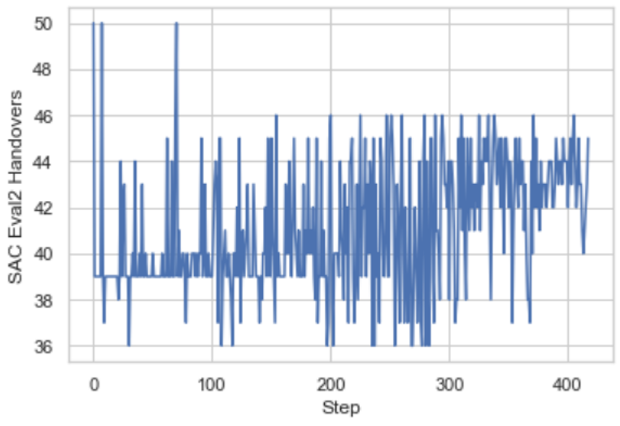


Figure 7: The number of handovers from the evaluation environments in SAC version of A2A4_5

Appendix D

The training and evaluation actions produced in A2A4_4 and A2A4_2a experiments

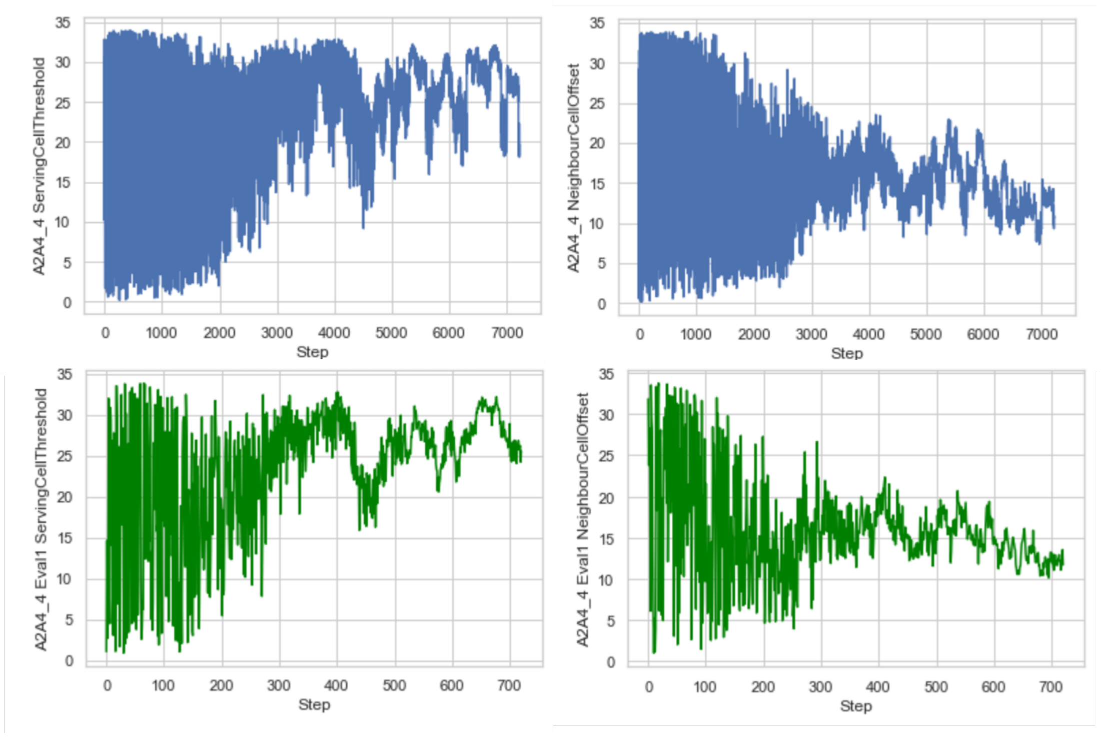


Figure 8: The training and evaluation actions produced in A2A4_4 experiment

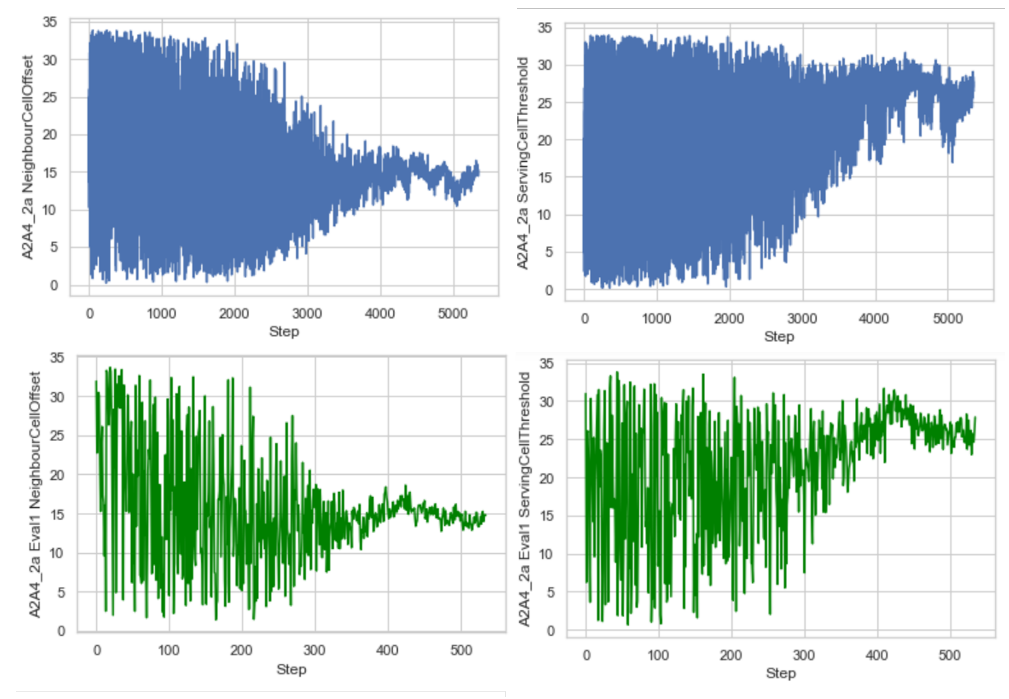


Figure 9: The training and evaluation actions produced in A2A4_2a experiment

Appendix E

The heat maps from the repeated grid search experiments

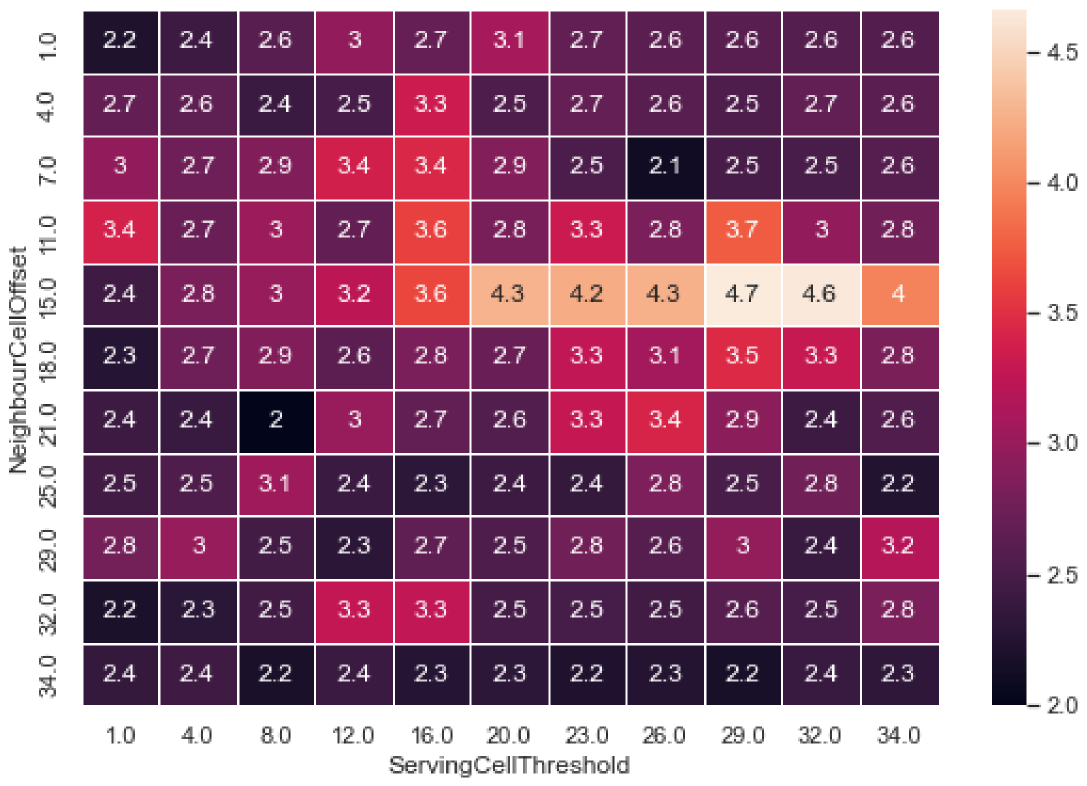


Figure 10: The heat maps from the first repeated A2A4 grid search experiments

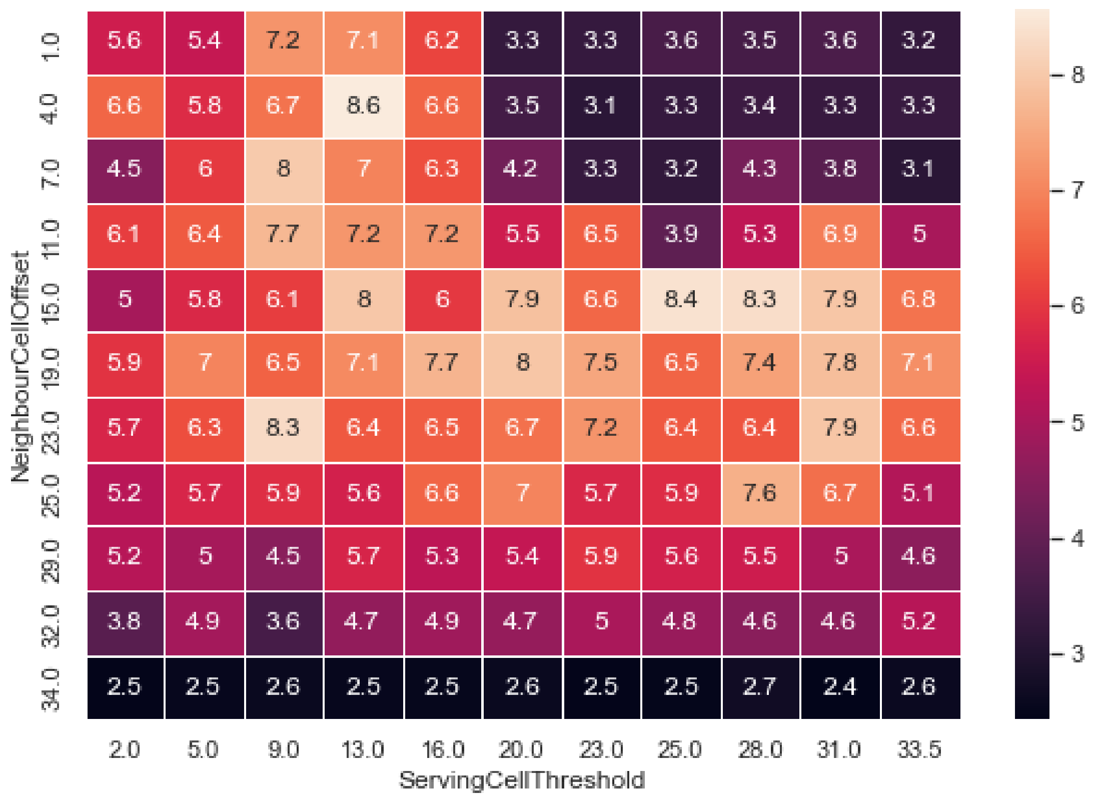


Figure 11: The heat maps from the second repeated A2A4 grid search experiments

NTNU

Norwegian University of
Science and Technology