Improving a Reinforcement Learning Algorithm for Resource Scheduling

Elin Wilson Andersson

Johan Håkansson

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6162
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

© 2022 by Elin Wilson Andersson & Johan Håkansson. All rights reserved.
Printed in Sweden by Tryckeriet i E-huset
Lund 2022

Abstract

This thesis aims to further investigate the viability of using reinforcement learning, specifically Q-learning, to schedule shared resources on the Ericsson Many-Core Architecture (EMCA). This was first explored by Patrik Trulsson in his master thesis *Dynamic Scheduling of Shared Resources using Reinforcement Learning* (2021).

The shared resources complete jobs assigned to them, and the jobs have deadlines as well as a latency. The Q-learning based scheduler should minimize the latency in the system. Most importantly, it should avoid missing deadlines. In this work, the Q-learning algorithm was tested on a simulation model of the EMCA that Trulsson built. Its performance was compared to a baseline and random scheduler.

Several parts of the Q-learning algorithm were evaluated and modified. The action and state space have been made smaller, and the state space has been made more applicable to the real system. The reward function, as well as other parameters of the Q-learning, were altered for better performance.

The result of all of these changes was that the Q-learning algorithm saw an increase in performance. Initially, it performed slightly better than the baseline on only one of the two configurations it was evaluated on, but in the end it performed significantly better on both. It also handles the introduction of noise to the simulation without a significant decrease in performance. While there are still things that might require further investigation, the algorithm always performs better than a baseline scheduler provided by Ericsson and is overall more suited for a real implementation due to the changes that have been done.

Acknowledgments

We would like to extend our gratitude to our supervisors at Ericsson, Jonas Korsell, William Tidelund, and El-Tayeb Bayomi, for being given the opportunity to work on this thesis as well as their guidance throughout it. Additionally, we would like to thank Karl-Erik Årzén, our supervisor at Lund University, for his assistance. Lastly, we would like to thank our examiner Bo Bernhardsson for his support.

Contents

Acronyms

3GPP Third Generation Partnership Project. 15

ATB Antenna to Beam. 17, 23–27, 47, 51, 52, 57–59

BAC Beamforming Accelerator Core. 11, 15, 17, 18, 23, 24, 28, 44, 46, 47, 51, 53, 56–59, 61–63

BFJ Beamforming Job. 17–19, 23–28, 32, 42, 47, 53, 54, 58, 61–63

BTA Beam to Antenna. 17, 23, 24, 26, 27, 30–32, 46, 47, 50–52, 58

cc clock cycles. 33, 34, 36, 37, 40–43, 45–47, 49, 50, 54, 55

DL Downlink. 11, 12, 15, 17–19, 23–34, 39, 41, 43, 46–48, 51–62

DQN Deep Q-learning Network. 11, 24, 25

EMCA Ericsson Many-Core Architecture. 3, 11, 12, 15, 22

FDD Frequency Division Duplex. 12

LTE Long Term Evolution. 15, 16

MDP Markov Decision Process. 19

NR New Radio. 15, 16

OFDM Orthogonal Frequency Division Multiplexing. 15–17, 23

PRB Physical Resource Block. 17, 23–26, 63

TD Temporal Difference. 20, 21

TDD Time Division Duplex. 12, 17, 23, 27, 29, 30, 44

UL Uplink. 11, 12, 15, 17, 18, 23–34, 39, 46–48, 51, 53, 55, 57–63

1

Introduction

The signal processing required for base stations using 5G at Ericsson uses a chip with a system called the Ericsson Many-Core Architecture (EMCA), which is a multi-core environment with concurrent jobs being executed on shared hardware resources. On the EMCA, there are one or several carriers that either send (called Downlink (DL)) or receive (called Uplink (UL)) radio signal transmissions, the process of which is called Downlink (DL) or Uplink (UL) respectively. In order to send or receive transmissions, beamforming is used, which requires large amounts of calculations to be done. These calculations can be made faster using specialized hardware, so called Beamforming Accelerator Cores (BACs). These can be modelled as a shared resource between UL and DL transmissions, the usage of which needs to be scheduled effectively in order to minimize overall system latency.

However, such efficient scheduling can be challenging to achieve due to the many different kinds of carrier configurations. These may need different kinds of scheduling behaviors in order to be considered optimal. Another thing that needs to be taken into account is that UL and DL transmissions have different latency definitions. It is also important that deadlines are met.

Using reinforcement learning to achieve this may alleviate some of these difficulties. It has earlier been investigated by Trulsson (2021) for his Master Thesis. He explored the use of reinforcement learning to schedule tasks with shared resources on the EMCA in order to minimize latency and meet deadlines. He implemented Q-learning and a Deep Q-learning Network (DQN). One of the conclusions he came to was that there were definite areas with room for improvement, some of which we looked at and tried to optimize.

Note that since our work is a continuation of that of Trulsson (2021), some parts of the report such as the Background chapter will look similar to his, since they treat the same subject matter. The report is written with the goal that you should not have to read his report to understand what we have done.

1.1 Goals

Our goal was to improve the Q-learning algorithm from Trulsson (2021) such that it outperforms a static scheduler on the massive beamforming part of the EMCA. This means that it should have a lower latency, as well as avoiding missing deadlines. The following list summarizes key points of investigation ordered by priority, the three first being the most important.

- Evaluate and improve the reward function
- Evaluate and improve the state- and action space
- Evaluate and improve exploration, e.g., exploring whether the system can protect itself from deadline misses while exploring?
- Try to reduce the agent's dependency on time in order to save memory for a real implementation.
- Evaluate if any extensions to the Q-learning need to be added, like model based reinforcement learning, decision trees, etc.

1.2 Method

We are using the EMCA simulation model that Trulsson (2021) built as a starting point, while possibly tweaking it if necessary. We will investigate different ways to improve the reinforcement learning he used based on the points above. The goal for the algorithm is to be able to schedule UL and DL types of symbols such that job deadlines are met and latency is minimized. Configurations such as state space and action space will be altered and benchmarked. Certain points of interest noted by Trulsson (2021) will be taken into account.

1.3 Delimitations

In general, we have similar delimitations as Trulsson (2021), which are that the EMCA model is somewhat simplified, see Chapter 3, and that the actions available to the scheduler are not all possible actions.

Additionally, the focus of this report is on Time Division Duplex (TDD), while Frequency Division Duplex (FDD) has not been investigated.

1.4 Report Outline

Introduction

Briefly describes the background of the problem, the purpose of the report, as well as the method.

Background

Conveys the information about the underlying system relevant to the problem, as well as background for the theory of reinforcement learning that is needed to understand the work done in this report.

Earlier Work

Contains a summary of the work of Trulsson (2021), particularly the parts that concern us and serve as our starting point as we continue to work on the problem. Also explains some bugs that were found and fixed.

Modifications

Depicts the modifications and testing that have been done in order to improve the Q-learning algorithm. It focuses on one modification at a time, being evaluated and discussed.

Results

Presents the final results in-depth, and also evaluates the Q-learning agent's robustness in more difficult situations, such as adding noise and increasing contention.

Discussion

Contains a reflection of the results, as well as potential avenues to explore for future work.

1.5 Individual Contributions

The programming work has been done together, utilizing pair programming and switching between being driver and navigator weekly. Both of us have been running tests, but Johan in general ran more.

A lot of the work on the report has been done together, because some parts required discussion as to how to best convey the information. Listed below are the sections and things that have been written/done separately, although the other person has of course read through them and given feedback.

Work by Elin:

- Created the images that were needed in addition to the ones generated from the program. She also modified images that needed larger text in order to be presentable.
- Chapter 1.
- Exploration & Exploitation in Section 2.2.
- Chapter 3
- Subsections Symbol Index and Slot Progression in Subsection 4.3
- The results and discussion regarding System 1 in Chapter 5
- Subsections Delays in the System and Configurations in Chapter 6

Work by Johan:

- Table formatting of results
- Subsection 2.2, excluding Exploration & Exploitation
- Subsections Number Jobs and Waiting Jobs in Subsection 4.3
- The results and discussion regarding System 2 in Chapter 5
- Subsections State Space and Future Work in Chapter 6

2

Background

2.1 Transmission

The EMCA chip sits in a base station with one or multiple carriers that handle incoming (UL) and outgoing (DL) signal transmissions. The carriers are responsible for generating symbols in a transmission [Trulsson, 2021]. They are divided into subcarriers [Dahlman et al., 2018]. The spacing between subcarriers affects the time between symbol arrivals [Lei et al., 2020].

An important part of handling the transmissions is beamforming, which requires complex calculations. To alleviate the burden, BACs are used for the calculations. However, these are a shared hardware resource between all transmissions and carriers, and thus need to be properly scheduled to maximize throughput and minimize latency.

Beamforming

Beamforming is the act of focusing a transmission in a specific direction, allowing a majority of the energy being sent to arrive only at the intended receiver. It can be done both ways, by having the receiver only listen to signals coming from the transmitter. This decreases the noise in the entire cell, increasing throughput [Rommer et al., 2019].

Orthogonal Frequency Division Multiplexing

New Radio (NR) is a radio-access technology released in 2017 that supports the use cases required by 5G. It builds upon its predecessor Long Term Evolution (LTE), which was released in 2009 and is associated with 4G. The organization which develops these technical specifications is called Third Generation Partnership Project (3GPP) [Dahlman et al., 2018].

In both NR and LTE the transmissions are sent and received by carriers using Orthogonal Frequency Division Multiplexing (OFDM), which is a transport technology taking advantage of several equally spaced subcarrier frequencies in order to split and transmit a high data rate stream into several parallel lower data rate

streams. The spacing between each subcarrier frequency is referred to as the subcarrier spacing. In OFDM each and every subcarrier is in theory orthogonal to each other, allowing the spectrum of each subcarrier to overlap with no risk of interfering with one another [Osseiran et al., 2016].

The subcarrier spacing of a carrier is determined by its numerology, see Table 2.1. Numerology 0 is supported by LTE, while NR supports a wider range of numerologies. The subcarrier spacing affects the duration in time of an OFDM symbol [Lei et al., 2020]. In this work, the focus is on numerologies 0 and 1.

For NR, transmissions are divided into 10 ms long frames, which consist of 1 ms long subframes. A subframe, in turn, contains a number of slots, each such slot containing 14 OFDM symbols each. The higher the numerology, the shorter the time between symbol arrivals, and the more slots a subframe contains, see Figure 2.1 [Dahlman et al., 2018].

Numerology	Subcarrier spacing (kHz)	Slot length (ms)
0	15	1
1	30	0.5
2	60	0.25
3	120	0.125

Table 2.1 Conversion table between numerology and subcarrier spacing.

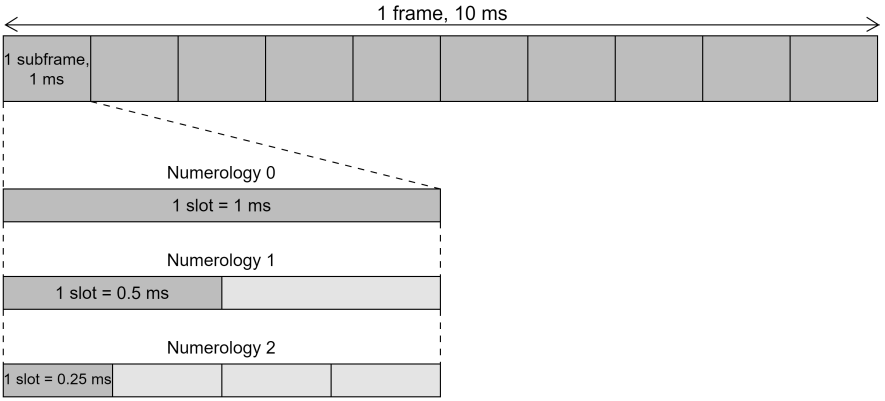

Figure 2.1 Frame structure and its relation to numerology.

Time Division Duplex

When the carriers send transmissions, this can be done in different duplex schemes, and one of them is Time Division Duplex (TDD) [Rommer et al., 2019]. TDD means that the arrival of UL and DL symbols are separated in the time domain but use the same carrier frequencies; the UL and DL symbols will arrive at different points in time [Dahlman et al., 2018]. What kind of symbol that will arrive at a given point in time depends on the TDD pattern.

One example of a pattern would be `DDSU`, where each letter is representative of a slot: D or U indicates that the entire slot consists of either only DL symbols or UL symbols respectively.

S represents the special slot, which has the format (X:Y:Z) meaning that the slot contains X DL symbols, Y GAP symbols, and Z UL symbols, sent in that order. A GAP symbol means that nothing is being sent, and a special slot always contains at least one GAP symbol, which is used to enact a guard period. This guard period ensures that there are periods where no transmissions are being sent, which is needed when the transmission switches from DL to UL in order to avoid interference [Dahlman et al., 2018].

Beamforming Jobs

Beamforming Jobs (BFJs) are the jobs in the system relating to beamforming calculations. For the purposes of this report, the terms Beamforming Job and job are used interchangeably. Note that Trulsson (2021) referred to these as Super Jobs.

A Physical Resource Block (PRB) is a unit of work, consisting of an OFDM symbol in the time domain, as well as 12 adjacent subcarriers in the frequency domain. Each BFJ is associated with a PRB size which directly affects the time a BAC takes to complete the job [Trulsson, 2021].

Each symbol results in the creation of several BFJs, the number of which is determined together with the number of PRBs such that they exploit the full bandwidth of the carrier the symbol belongs to [Trulsson, 2021].

The bandwidth of a carrier can be expressed in PRBs. Since each carrier has a subcarrier spacing and each PRB has 12 adjacent subcarriers, a carrier with 20 MHz bandwidth and a subcarrier spacing of 15 kHz will have a PRB of 100. This is because $100 \cdot 12 \cdot 15 \approx 18$ MHz. The reason that we do not end up with exactly 20 MHz is because guard bands without data transmission are needed to avoid interference [Lei et al., 2020].

In the case of TDD, different types of transmissions generate different job types in a base station. A UL transmission generates two types of Antenna to Beam (ATB) jobs, ATB1 and ATB2, whereas a DL transmission generates Beam to Antenna (BTA) jobs [Trulsson, 2021]. In this report, the term for a symbol and the term for the jobs generated from it are sometimes used interchangeably depending on context.

Resource Pool

Since there are a limited amount of BACs, not all jobs can be sent to them immediately. Instead, all jobs are first sent to a priority queue. The BACs then retrieve the jobs with the highest priority and earliest arrival, performing the beamforming calculations without interruption until finished. Together, the queue and the BACs are called the resource pool.

System Overview

It is the task of the scheduler to handle arriving symbols such that primarily, deadlines misses are avoided at all costs, and secondarily that the latency is minimized. The arrival timing of a symbol depends on the number of carriers and their properties, and the time between symbol arrivals from the same carrier is referred to as symbol time. The deadline also depends on carrier properties, being located a set amount of symbol times after the symbol's arrival. The scheduler determines the number of BFJs, and then BFJ tokens are generated from the symbol. The scheduler also decides when these tokens should be sent to the resource pool, and what priority they all have in the priority queue. The beamforming calculations for a symbol are complete when the BACs have completed the work on all of the generated BFJ tokens.

Latency Definitions

Transmissions from a user to the base station are referred to as Uplink. For UL symbols, the latency is defined by the time between arrival to the scheduler and the completion of the beamforming calculations. This includes the time before being sent to the resource pool, the time waiting in the priority queue, and the time for the beamforming calculations, see Figure 2.2.

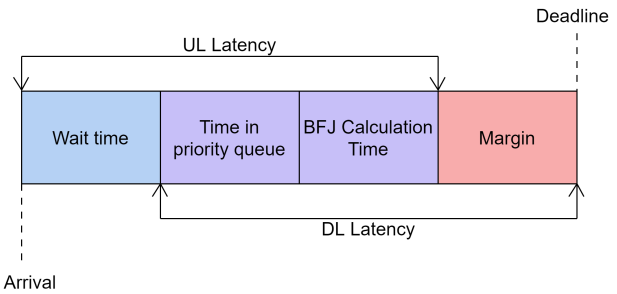

Figure 2.2 Latency definitions for UL and DL symbols.

DL symbols, where the base station is sending transmissions to the user, have a different latency definition. The transmissions have a set time upon which they are sent, i.e., their deadline, which means that having the BFJ complete too long before that time takes up memory. The latency for a DL symbol is determined by the time spent in the resource pool, and the margin between the beamforming calculations finishing and the deadline.

Missing deadlines might lead to a reduced quality of service or waste radio resources if a resubmission is needed.

2.2 Reinforcement Learning

Reinforcement learning is a subset of machine learning concerned with learning how to map certain situations into actions based on trial and error. It differs from other machine learning paradigms in the sense that the learning agent learns by directly interacting with its environment and observing the impact. This, as opposed to learning based on provided sets of situations and corresponding optimal actions [Sutton and Barto, 2018].

In order to make this possible, the agent needs a way of observing the environment, known as the state. It also needs a set of actions which can influence the environment. The situation can be described using a Markov Decision Process (MDP), which is used to formalize the reinforcement learning problem into having states, actions, and goals. The mapping from certain situations to actions then becomes a mapping between states and actions, also known as the policy. The goal of the agent is to find the optimal policy; the action for a given state that gives the best result. This is defined using rewards, signals that the environment sends to the learning agent to measure the worth of an action based on its impact, see Figure 2.3 [Sutton and Barto, 2018].

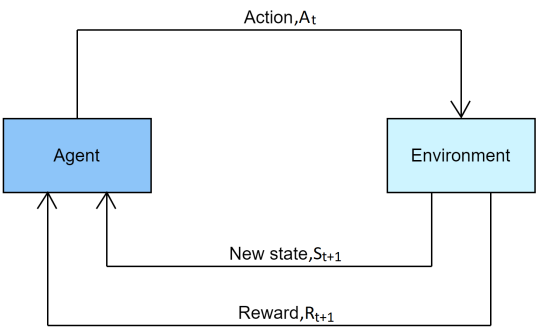

Figure 2.3 The interaction between an agent and an environment in an MDP. Note that this uses a similar denotation as Equation 2.2.

While a reward represents the short-term value of an action, the value function represents the long-term value, being a representation of cumulative future rewards. This is because even if the immediate reward of an action is not high, the resulting state might be advantageous in the sense that it offers high rewards in the long run, and vice versa. It is the value function that is used to make decisions [Sutton and Barto, 2018].

A reinforcement learning algorithm may also be model-based, while it is called model-free if it is not. A model imitates the environment and how it behaves, which means that it can be used to predict what might happen in the future and allow the agent to make decisions while taking into account what state that decision will result in [Sutton and Barto, 2018].

Temporal Difference Learning

As described by Wiering and Van Otterlo (2012), Temporal Difference (TD) learning is a branch of reinforcement learning which does not require a model. It uses a technique called bootstrapping to learn over the course of a training episode. The value function is estimated using bootstrapping, which means that values are based on other estimated values [Sutton and Barto, 2018]. This means that algorithms using TD-learning do not have to wait until the end of a training episode to update their value functions, but instead do so continuously, as opposed to other methods [Wiering and Van Otterlo, 2012].

The simplest version of TD is called TD(0), or *one-step TD*, and its value function estimation is seen in Equation 2.1. Upon transition from state S_t to S_{t+1} and receiving the reward R_{t+1}, the estimated value $V(S_t)$ is updated. The update takes into account the current estimated value, the reward, and the estimated value of the next state [Sutton and Barto, 2018].

$$V(S_t) = V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{2.1}$$

$\alpha \in [0,1]$ is the step-size parameter. Sutton and Barto (2018) show several examples where α can be either constant or decreasing over the course of training. Wiering and Van Otterlo (2012) call it the learning rate, which influences the impact of the value estimate updates. According to Even-Dar et al. (2003), the rate of convergence is dependent on how the learning rate changes throughout the training.

$\gamma \in [0,1]$ is the discount rate for future rewards. It determines the weight of future rewards, which means that depending on how far into the future a reward is received, it will matter more or less. The reward decreases each time step into the future, for example: At time step k, the reward will only be worth γ^{k-1} of what it would be if it were a current reward [Sutton and Barto, 2018].

Q-learning

Q-learning is an off-policy, model-free TD algorithm. Off-policy means that the value function is not approximated using the policy, but another policy, tending to be more exploratory [Sutton and Barto, 2018]. Model-free approaches to TD learning use state-action value functions, denoted $Q(S_t, A_t)$, to estimate values based on state and action taken [Wiering and Van Otterlo, 2012]. In Q-learning the value function is approximated in accordance with Equation 2.2 [Sutton and Barto, 2018]. The difference compared to the TD value function estimate is that the Q-learning value function also includes the action that was taken, and the estimated value of the next state is based on the action in that state that has the highest value estimate.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{2.2}$$

Given an infinite amount of visits to each state-action pair, Q-learning is guaranteed to find the optimal policy if α is reduced suitably, independently of the exploration policy [Wiering and Van Otterlo, 2012].

Exploration and Exploitation

Choosing an exploration policy is important. Exploitation of current knowledge is needed because it means performing the current best action. However, the agent should explore other actions in order to learn new things. Exploration does not always find a new best action though [Wiering and Van Otterlo, 2012].

An example of a simple exploration policy is the *greedy method*. This method entails selecting the action with the current highest estimated value, i.e., exploiting current knowledge. However, since this method never tries any other possible actions, it runs the risk that an inferior action is considered to be better than an optimal one [Sutton and Barto, 2018].

This is remedied using the ε-*greedy* method, which instead chooses to exploit with a probability of $1 - \varepsilon$, and will perform an exploratory action, trying out an action randomly, with a probability of ε [Sutton and Barto, 2018].

With ε having a high value the agent will perform more exploratory actions. This may mean that it finds the optimal action quickly, but does not choose to perform it as often. A lower value may mean that the agent takes longer to find the optimal action, but exploits it more. It is also possible to have ε decrease from a higher value to a lower in order to benefit from the advantages of both [Sutton and Barto, 2018].

3

Trulsson's Work

Trulsson (2021) set out with a similar goal to ours; to implement a reinforcement learning algorithm that would outperform a static scheduler. He also built the simulation model that we are using to evaluate the algorithms.

3.1 Schedulers

Apart from the agent itself, two other schedulers were used for comparison. One was a static scheduler provided by Ericsson which serves as the baseline, making decisions solely based on the job type of the symbol. The other was a scheduler that made decisions at random in order to showcase the challenge of achieving good scheduling.

3.2 Simulation Model

Trulsson (2021) built the simulation model mimicking a possible scenario for the EMCA using SimPy, a discrete event simulator [Scherfke et al., 2020a]. Using it, symbols can be sent to the scheduler at an appropriate time based on a traffic file, imitating how it would work in real life with symbols being sent to the system by different carriers. See Figure 3.1 for an overview.

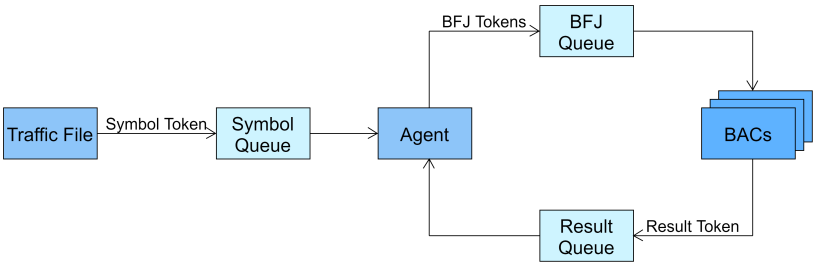

Figure 3.1 An overview of the simulation model flow.

The traffic file is generated using a given TDD pattern and based on the current configuration, discussed further below. Notably, a D slot will always result in BTA symbols being generated, whereas a U slot will result in one ATB1 and one ATB2 symbol per OFDM symbol. ATB1 symbols are hardcoded in the sense that one job is generated per symbol, which means that they have a fixed PRB size.

The simulation program reads the symbols from the traffic file, sorts them by timestamp, and puts them in the FIFO symbol queue. The timestamp represents the moment in time when the symbol will arrive to the system. A SimPy process then extracts one symbol from the queue at a time, waits until it is time to send the symbol, then sends it to the agent for scheduling.

The agent then makes its decision, determining BFJ PRB size, priority, and wait time. The wait time is decided in an interval between the symbol's arrival time and its deadline; its precision depending on a constant amount of time steps, referred to in this report as time precision. This will be further discussed in Section 5.1. Once the wait time has passed, a number of BFJ tokens representing the jobs are generated from the symbol based on the decided BFJ PRB size, and then sent to the BFJ token queue. The BFJ token queue is modelled as a SimPy priority store, which means that when tokens are extracted from it, this will be done according to the decided priority [Scherfke et al., 2020b].

The BACs are simulated as a process that pops BFJ tokens from the BFJ token queue and waits the amount of time the job would take, calculated based on the properties of the token. Then, the result is sent to the result queue. A symbol is considered to be finished when all generated jobs are done and that is when the Q-learning can learn something from the result. The process that handles the results pops items from the result queue, and updates the agent accordingly.

Note that the simulation does not include the runtime of the scheduler and the BFJ queue, which is a simplification of how it is in reality.

3.3 Configurations

The agent was trained on input consisting of 10 UL symbols followed by 10 DL symbols, which in actuality becomes 30 symbols total since UL symbols generate both ATB1 and ATB2 symbols. The main situation being highlighted was the switch from the UL to DL slot, wherein the first DL symbols start arriving before the UL symbols stop coming. Throughout an agent being trained and evaluated, the configuration remains the same. For example, there will always be the same amount of carriers sending traffic. The traffic file is not part of the configuration, even if it is based on it.

The Q-matrix is zero-initialized, meaning that at the start of the training, all actions have equal value. The size of the matrix is determined by the state space and action space. That is to say, the amount of different states, as well as the number of possible actions. The value of an action in a given state corresponds to the value in

the Q-matrix when indexing using that state and action.

Trulsson (2021) tested the algorithms on two different systems which we will also be using, presented in Table 3.1. Both systems used 40 BACs, and the total carrier bandwidth was 100 MHz. System 2 is the more challenging of the two, given that it has five times as many carriers, and thus will have more traffic to schedule, at a higher contention rate, even though System 1 has higher numerology which means that the latency requirements are harder.

Name	# Carriers	Numerology
System 1	1	1
System 2	5	0

Table 3.1 Carrier configurations.

The **state space** of the reinforcement learning algorithms in Trulsson (2021), and therefore our starting point, consisted of three variables. These were:

- Job type (ATB1, ATB2, or BTA)
- The number of jobs in the resource pool
- Symbol index

To clarify, the symbol index is a unique index for each symbol based on time of arrival.

The **action space** consisted of:

- The BFJ PRB size
- The BFJ queue priority
- The wait-time before the jobs are sent to the resource pool

There were two different implementations of reinforcement learning in [Trulsson, 2021]; Q-learning and a Deep Q-learning Network (DQN). DQN is a variant of Q-learning that instead of using a matrix to store the value $Q(S_t, A_t)$ it uses an artificial neural network [Sutton and Barto, 2018]. Both used exploration factor probability $\varepsilon = 0.1$ while training. The Q-learning scheduler had a constant learning rate of $\alpha = 0.5$ and a future rewards discount of $\gamma = 0.99$.

The reward function used in [Trulsson, 2021] is presented below, rewarding a scheduling based on a symbol's latency x. The k-values are design constants. UL and DL are rewarded differently.

$$r(x) = \begin{cases} e^{-k_{DL} \cdot x} & \text{for DL} \\ e^{-k_{UL} \cdot x} & \text{for UL} \\ -1 & \text{if deadline miss} \end{cases} \tag{3.1}$$

3.4 Results

Overall, Trulsson (2021) saw promising results. On System 1, the Q-learning scheduler had a slightly lower total latency than the baseline. The UL symbols had a higher latency than the baseline, while the DL symbols' latency was lower. In the UL/DL-switch, the scheduler managed to schedule the DL symbols a bit later than the baseline.

On System 2, the Q-learning scheduler met all deadlines but had about twice as high latency as the baseline. In general, the DL symbols were scheduled earlier than the baseline, giving reduced performance.

The scheduler using DQN had a higher latency as well as suffering from deadline misses on both systems. Therefore, we are primarily focusing on improving the Q-learning scheduler.

3.5 Our Starting Point

While we were working with the code we noticed that sometimes the algorithm did not quite behave as expected, and had results that did not make sense. An example of this was that in some situations, no matter how much the punishment for a deadline was increased the agent did not stop making the decisions that led to the misses. This was of course not in line with how Q-learning should work.

The reason why was discovered by investigating the Q-values of trained agents. An important part of the scheduler picking an action is what index the action has in the Q-matrix. Some actions are forbidden by the scheduler, for example ATB1 symbols' PRB sizes are statically decided. This necessitates being able to convert back-and-forth between actions and their respective indexes. However, there was a bug in this functionality. If an index was converted to an action and back again to an index, the end result would not always be the original index. And in the code, this conversion is always done, to account for when part of an action is statically decided.

This meant that if an action was chosen, that action would always be executed, but the corresponding index was not always correct. In that situation, when an action got its reward, or punishment, the reward was not contributed to the actual action that had been executed, but instead to the one connected to the incorrect index. Because of this, an action could have a high value in the Q-matrix on unfair grounds, or a lower value than it should. After fixing the bug, the conclusion was that this was the reason for the agent behaving in unexpected ways.

It can not be said for certain how large of an issue this was for Trulsson (2021) since there were other areas to explore in order to improve the Q-learning, but it likely had some impact.

Additionally, the simulation's BFJ queue did not take the arrival time into account for jobs with equal priority. This was also fixed.

4

Modifications

Presented in this chapter are the modifications that have been tested out on the Q-learning algorithm with the ultimate goal of minimizing latency and avoiding deadline misses for both systems. Another factor to keep in mind is that the algorithm should be viable for adaption to a real implementation.

Note that the tests presented in this chapter have been performed on System 2, and evaluated on the `DDSU` evaluation pattern. System 2 was chosen since it has five carriers, and thus requires the scheduler to make good decisions in situations with higher contention. System 1 has a lower degree of contention even with its higher numerology, meaning that it is easier for the agent to find the optimal policy. Additionally, since System 2 is more complicated, some of the changes have a larger impact on that system. For tests performed on System 1, see Chapter 5.

To avoid repetition, deadline misses are mentioned whenever they occur, but are omitted if all deadlines are met.

4.1 Initial Action Space

To keep things simple to start with, the action space was scaled down. Instead of deciding BFJ PRB size, priority, and wait time, the agent only decides the wait time, and only for DL symbols. The other actions are instead replaced with how the baseline scheduler would act.

Recall that UL symbols generate ATB jobs and DL symbols generate BTA jobs. While ATB jobs ideally should be scheduled to start as soon as possible, there are situations where it might be beneficial if they are not. In the UL/DL switch, a later scheduling of a UL symbol could allow a DL symbol to meet its deadline.

However, having the wait time for the ATB job be dynamic might not necessarily help. ATB jobs are preferably sent to the BFJ queue as soon as they enter the system, whereas BTA jobs ideally wait a while. If ATB and BTA jobs exist in the system simultaneously and affect each other's results, this means that the BTA jobs were likely scheduled a couple of symbols back in comparison to the ATB jobs. If ATB jobs cause BTA jobs to miss a deadline, for example by starting right before when

the BTA jobs would need to start in order to meet their deadline, this information is in no way propagated to the ATB jobs' reward process. The Q-learning update function takes future rewards into account, but if the scheduling of a symbol causes another symbol scheduled earlier to miss, it will not learn of this. This issue would not exist if the symbols were worked on in the same order as they arrived to the system. Thus, letting ATB wait time be decided dynamically has a limited benefit.

The assigned priority for the BFJ queue is also statically decided, and there might not be a reason to decide that dynamically again either. During traffic with only one job type, priority does not matter, since all symbols have the same ideal scheduling. However, in the UL/DL switch, it is important that BTA jobs that have been scheduled close to their deadlines get retrieved from the BFJ queue in a timely manner, such that they actually meet their deadlines. This might not be the case if ATB jobs are sent to the system at the same time with with an equal or higher priority. Statically assigning higher priority to BTA jobs will ensure that this does not happen.

4.2 Input Data

Training and evaluating on the right kind of data is important, since it directly affects the agent's behavior. Therefore, the TDD pattern used in the traffic generation was expanded from the TDD pattern `UD` used by Trulsson (2021) for both training and evaluation, to instead use the pattern shown in Table 4.1 for training. Since UL symbols have their wait times statically assigned, the special slot has been chosen without any UL symbols to increase the possible training for DL symbols in the special slot.

	Special slot		
Pattern	#DL	#GAP	#UL
DDSUUDDSUD	11	3	0

Table 4.1 The pattern used to train the agent.

The patterns used for evaluation of the agent have also been updated, shown in Table 4.2. A situation that is of particular interest, as also seen in [Trulsson, 2021], is the switch between UL and DL symbols. The `DDSU` pattern is repeated in order to evaluate this switch.

		Special slot		
Pattern	Repetitions	#DL	#GAP	#UL
DDSU	2	11	3	0
DDDSUUDDDD	1	3	8	3
DDDSUDDSUU	1	10	2	2

Table 4.2 The patterns used to evaluate the agent.

These new training and evaluation patterns better reflect situations that might occur in a real scenario, and not just the UL/DL-switch. Not all of these situations require the same kind of scheduling, or have the same difficulties. The UL/DL-switch is difficult because the end of the U slot coincides with the beginning of the D slot. This means that for a period of time, both kinds of symbols are arriving, which also increases the contention in the system. Then, a DL symbol might not be able to be scheduled too close to its deadline, because a UL symbol might arrive shortly before the DL symbol is scheduled. They might utilize the BACs in such a way that not all of the generated DL jobs will finish in time. This requires the DL symbol to be scheduled earlier.

In contrast, when a UL/DL-switch is not occurring, scheduling DL symbols becomes easier. Since all symbols have the same optimal behavior, that is to say finishing close to their deadline, symbols that arrive at different points in time will not interfere with each other. This is because they will have different deadlines, and if they are both scheduled optimally they will be entering the resource pool at different times. As for UL symbols, all actions for those are still statically decided, being sent to the BFJ queue as soon as they arrive to the system.

It is advantageous for the agent to be able to handle both of these types of situations, and schedule them differently.

Additionally, the traffic file has been extended in the sense that it also contains information about certain state events to support some of the updated parts of state space described below. During the simulation, this information is added to the symbol queue in order to retain information regarding the order of events and symbol arrivals. Thus, it might not be accurate to call it a symbol queue, since it contains not only symbols, but also events. The simulation as a whole still functions the same, though.

4.3 State Space

Presented below are modifications to the state space that have been investigated, as well as why they are interesting to explore.

Symbol Index

As noted by Trulsson (2021), the symbol index gives the algorithm a sense of time. Each symbol in the simulation will be given a unique symbol index. This, however, can be detrimental in the sense that as the time during which the agent is trained continuously increases, so will the symbol index.

This will cause memory issues, since the size of the Q-matrix is dependent on the amount of possible values of the state space, which increases with the number of symbols. This is especially a problem when testing System 2, which has five times as many symbols as System 1. Additionally, it would not be possible to train and evaluate on different traffic files if they do not have the same size.

The symbol index also has some flaws when it comes to generality. Because of the way symbol index is assigned, the agent will learn which action is optimal at a **certain point in time** during the training episode, based entirely on the traffic file. For different types of TDD patterns of the same length this is not generally applicable, because behavior that was optimal at a certain point in time for one pattern might not be optimal for another. Additionally, two symbols in different patterns may have the same symbol index, but not the same job type; they will have different states. When training on one of the patterns and evaluating on the other, the agent might encounter states which it has not trained on.

Assuming that the agent is being trained and evaluated on the same file, there is no point in having both symbol index and job type, since any symbol index will always correspond to the same job type. Therefore, having job type and symbol index simultaneously is a waste of memory. This issue does not arise when it comes to the number of total jobs in the system, since exploration means that this number might vary at a given point in time.

There is also the risk of the agent learning something that does not reflect the real world, which is that it may start scheduling jobs towards the end of a traffic file later, since there will be no more arriving symbols to contend with. This is obviously not true to reality, where the patterns could repeat indefinitely.

Another issue with symbol index is that there is no common denominator between different symbols with similar properties. For example, if a UL/DL switch occurs twice in a traffic file, there would perhaps be a benefit if the symbols were scheduled in similar ways. But if the symbol index is a part of state space, the agent will not be able to take advantage of this.

The Solution The symbol index needs to be replaced, and its replacement should be taking advantage of symbols that have similar properties. It would make sense to introduce a state variable which is supposed to convey a sense of surroundings as

opposed to a sense of time.

This state will be referred to as **slot state**, and it keeps track of the most recent as well as the current slot. Some other versions of this have been considered, such as also keeping track of the upcoming slot, or only keeping track of the current and future slot. Ultimately though, the past and current slot are enough to serve our purpose.

The idea is that by giving the agent an idea of what has happened recently in the past and what is happening now, there will be a distinction between different states that need different scheduling considerations. If the current slot is D, then the symbols need to be scheduled differently depending on if the last slot was U or D, since if U was the latest slot, UL symbols may still be arriving.

TDD-patterns have restrictions on the order in which different types of slots can be arranged. Therefore, while one might think that if a slot can be either D, S or U, it would necessitate $3 \cdot 3 = 9$ different slot states. However, the actual number of valid slot states are smaller, at five.

With this, the size of the Q-matrix can be significantly reduced, as well as not depend on how many symbols the file it gets its traffic from contains. This means that the agent can be trained and evaluated on different files without any restrictions on file size. It also solves the potential memory issues that arise as the symbol index grows larger. Additionally, the time axis for a real implementation is reduced.

Another effect of the change is that each state can be trained multiple times during one training episode. This was not possible before, when symbol index was a part of state space. This results in the agent learning more per training episode, which makes the training process faster.

Slot Progression

It is important that the state space is precise enough that symbols that require different scheduling behaviors have different states. One such situation is in the UL/DL-switch. The arrival of both kinds of symbols in the system makes scheduling more difficult, since there are more jobs in the resource pool that need to be worked on. This may mean that BTA jobs are not able to fully minimize their latency without risking missing deadlines. However, the UL symbols stop coming before the DL symbols do, which means that not all DL symbols should need to be scheduled as if there were also UL symbols in the system.

In other words, these situations need to have different states, and it should be connected to how far into the slot the symbols are. A new state space variable is introduced, called **slot progress**, which keeps track of this.

It is important to consider how many different values slot progress needs to have in order to fulfill its purpose. The different scenarios are shown in Figure 4.1. Ideally, slot progression is divided as seen in scenario a). A symbol that needs a harsher scheduling, harsher in this context meaning that it needs to be scheduled further away from its deadline in order to meet it, should not have the same state,

and therefore scheduling, as one that can be scheduled closer to its deadline. If they do, all the symbols will either be scheduled closer to their deadline in scenario b), resulting in deadline misses, or scheduled earlier in scenario c), meaning an increased latency for some symbols. The latter situation is not as dangerous as the former.

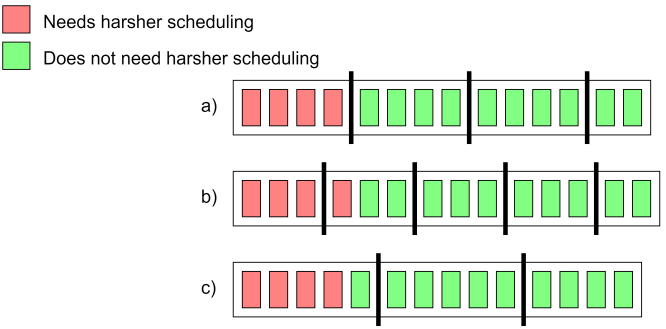

Figure 4.1 A simplified view of **slot progress** and the different scenarios that occur when scheduling BTA symbols in the UL/DL-switch. The black lines divide the slots into different states. Situation a) describes the ideal case where symbols that require a harsher scheduling are in a state separate from those that do not. Situation b) represents the undesired case where a symbol that requires a harsher scheduling has the same state as ones that do not, possibly risking deadline misses. Situation c) describes a situation that is undesirable but not as bad as situation b). A symbol that does not need a harsher scheduling has the same state as the ones that do need it, resulting in higher latency for this symbol.

However, the added precision that slot progress gives is not always a benefit. In fact, its use is limited to the UL/DL-switch, where symbols in the same slot should be treated differently. In pure DL or special slots, this is not the case. All symbols should have the same scheduling, and introducing slot progress will only increase the training time the agent needs to find the optimal scheduling. Therefore, slot progress is only used in the UL/DL-switch.

Number Jobs

Currently, the state variable keeping track of the number of jobs in the resource pool tracks the exact number of jobs. This means that two otherwise similar symbols with a very small difference in this part of state will be treated differently from each other. The precision is too high, because the behavior of the agent should not differ if there are for example 17 current jobs in the system as opposed to 18.

That is why that part of the state space has been reworked to instead reflect contention in proportion to the number of cores. Important to consider here is the degree of precision, since being either too precise or imprecise will render the state

worthless. Too high precision does not eliminate the issue, while a too low precision might mean that there is no distinction between varying contention levels.

Lowering the precision also serves to further reduce the size of the state space, and therefore the Q-matrix, while also making the learning process faster for the same reason as implementing slot state does.

Waiting Jobs

An important insight is that there is a delay between a decision being made and the decision being reflected in the system. This is because when the scheduler determines the wait time for a symbol, the jobs are not sent to the BFJ queue immediately, but instead only when the wait time has passed. Currently, there is no way to keep track of these jobs, even though information regarding the contention in the near future could be available to the scheduler, since it was the scheduler that made the decision in the first place.

The potential issue lies in the fact that there might be jobs that will alter the system's contention in the future, which could affect the optimal scheduling of the current symbol.

With all this in mind, a new state variable is introduced that represents future contention, referred to as waiting state. As with the variable keeping track of current contention, it should be represented with appropriate resolution, for the same reasons.

Final State Space

Not all of the modifications listed above were kept, because they were not all needed. The final version of the state space consists of:

- Slot State
- Slot progress
- Job type

Keep in mind, if DL symbols were to continue being the only symbols scheduled dynamically, the job type should eventually not be needed either. Contention as well as waiting state have been removed.

Contention, as it turns out, has little to no value. Firstly, for an optimal scheduling, there should be no contention in the system at the time of a symbol arriving. This is because symbols arrive at certain intervals, which also coincide with other jobs' deadlines. Any BTA jobs in the system should optimally be scheduled close to their deadline and thus already have finished or not entered the resource pool yet. Therefore, the contention would be zero. Furthermore, and more importantly, why should the **current contention** matter to how a DL symbol should be scheduled? Jobs that affect the current contention have ideally already finished by the time the symbol enters the resource pool. For UL symbols it does not matter either, given

that the jobs should be sent to the resource pool as soon as possible regardless of contention.

The number of jobs waiting to be sent to the resource pool might be more useful in that case. However, it turned out during testing that it was too imprecise, resulting in the state variable having the same value a majority of the time. Additionally, it turns out that it was not needed to get a good behavior from the agent for the configurations it was tested on. The main advantage of keeping track of the amount of jobs waiting to be sent is mainly when there are several carriers sending symbols at the same time. It could allow the agent to differentiate between the first and third symbol that arrived simultaneously, allowing them to be scheduled differently, which could be useful in the UL/DL switch. This is useful because if there is a high degree of contention with a risk of deadline misses, all symbols would otherwise have to be scheduled earlier.

Since the size of the Q-matrix is dependent on the number of states, these changes have impacted it. The number of different values for each state space variable is listed in Table 4.3 along with the total number of combinations of state. Note that the number of values for the symbol index may vary depending on the training/evaluation file, which means that the size of the Q-matrix also may vary.

Modifications	State	Number different values
Before	Symbol index	134 (varying)
	Number jobs	401
	Job type	3
	Total combinations	161 202
After	Slot state	5
	Slot progress	4
	Job type	3
	Total combinations	60

Table 4.3 State space and the number of combinations of state before and after the modifications. The symbol index is assumed to have 134 different values, which is the case when evaluating on the `DDSU` pattern in System 1.

The results when running tests with the parameters as defined in Chapter 3 are shown in Table 4.4. Note that the baseline's performance on the same traffic file is 28 942 268 clock cycles (cc). Although UL symbols' wait times are statically decided, they are still taken into consideration when calculating the total latency. This is because they still coexist with the symbols that are dynamically scheduled, and may be affected by or affect other symbols.

The results are somewhat inconsistent in the sense that two agents trained with the same settings might have differing performances. For this reason, when evalu-

ating the algorithm several agents have been trained with the same settings to estimate the average performance. Aside from that, the results seem promising since the agent is capable of performing with a lower latency than the baseline. Especially when considering that quite a bit of the precision that Trulsson (2021) had has been stripped away, while also generalizing the state space.

Min	Max	Average	Median
18 330 765	57 632 881	31 256 031	30 499 538

Table 4.4 Latencies when training 20 agents 5000 episodes each on System 2, and then evaluating them on the DDSU pattern. The baseline's performance on the same traffic file is 28 942 268 cc. It seems promising that the agent is sometimes able to perform better than the baseline.

The inconsistency sometimes leads to deadline misses, see Figure 4.2. Looking closer at these misses, the issue seems to be that the symbols have simply been scheduled too close to their deadlines, resulting in guaranteed misses. The other symbols are scheduled close to optimally, however, even in the more challenging UL/DL switch.

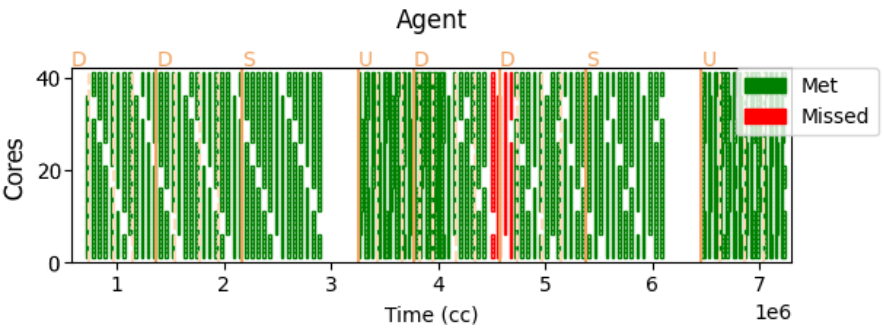

Figure 4.2 Deadline status for System 2 on the DDSU pattern, where each rectangle represents a job. The yellow vertical lines represent slot state changes, and the dashed ones represent slot progress.

The fact that the agent sometimes misses deadlines is highly undesirable. A possible reason for it could be a lack of training. In general, it can be difficult to determine when a reinforcement learning algorithm has converged. However, the Q-values of the state-action pairs change over time as the agent learns which actions are good and which are not. Eventually, the average values should stabilize, at which point it can be concluded that the agent will not learn anything new if the training is continued.

The average Q-values for this run are presented in Figure 4.3. Compared to Trulsson (2021), whose Q-values can be seen in Figure 4.4, the agent now seems to converge faster and also more convincingly, likely being a result of the smaller state and action space. The updated agent converges to a smaller value than that of Trulsson (2021), which is because of the statically decided actions.

The average Q-values for runs without deadline misses look similar to those presented in Figure 4.3. Based on the figure, the Q-values quickly shoot up, and then seem to fluctuate between values in a small interval, indicating that the agent has learned all it can. However, the oscillation means that the agent's behavior is still changing. Coupled with the fact that not every trained agent will have deadline misses, this could indicate that drastically different decisions (good ones, ones that guarantee deadline misses, and those that do not miss but have poor latencies) end up with similar Q-values.

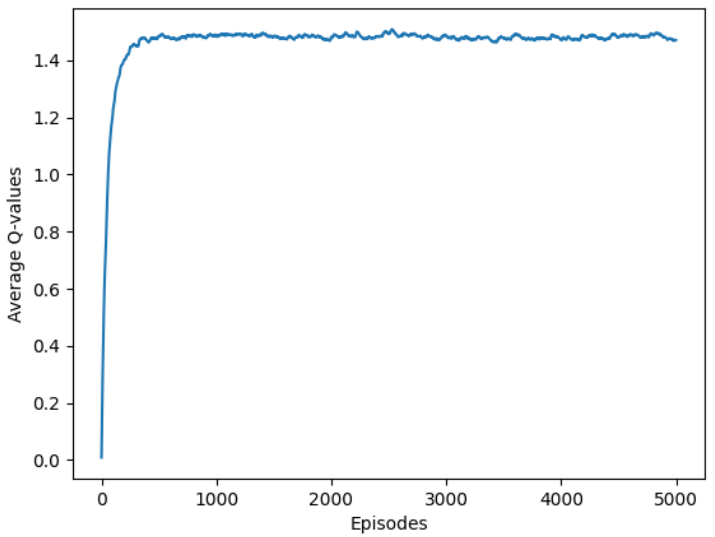

Figure 4.3 The average Q-values of an agent trained on System 2 with the `DDSUUDDSUD` pattern. It rises in value quickly in the beginning and then seems to stagnate with a fluctuation, indicating that the agent largely has finished learning. Note that the values by themselves are not interesting, what matters is that the values no longer change significantly. A low average could simply mean that there are many actions with poor rewards, which still means that the agent has learned something.

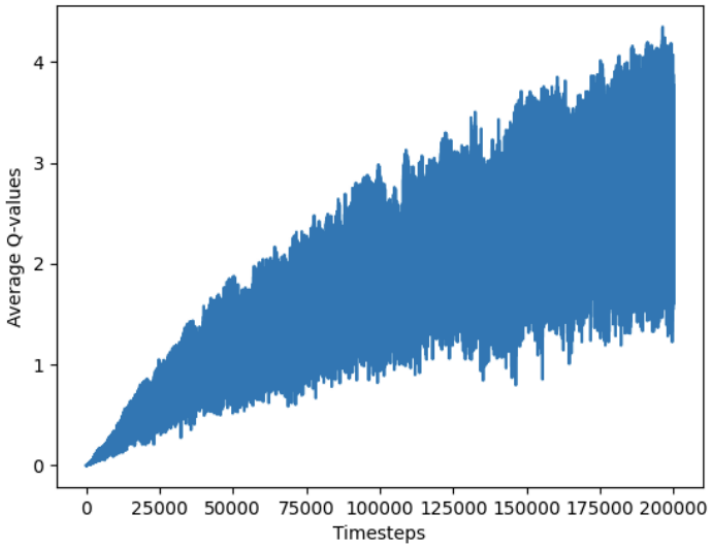

Figure 4.4 The average Q-values of the agent from [Trulsson, 2021] trained on System 2 with the `UD` pattern. In comparison with Figure 4.3 the average value still seems to be rising towards the end, indicating that there is more to learn.

This inconsistent behavior is of course undesirable, and thus different α configurations will be investigated in order to see if it is possible to reduce this variation.

4.4 Learning Rate

Different configurations of α were tested, constant as well as linearly decreasing, in order to find the ones that lead to the best behavior. This was both in terms of lower latencies and increased consistency. Having a linear decrease means that further into the training, newer rewards will affect the Q-values less. In other words, past experiences matter more than what was recently learned. The results can be found in Table 4.5 as well as Table 4.6. Note that the results for $\alpha = 0.5$ are taken from Table 4.4. For comparison, the baseline's performance on the same traffic file is 28 942 268 cc.

	Latencies (cc)			
α	Min	Max	Average	Median
0.5	18 330 765	57 632 881	31 256 031	30 499 538
0.3	17 419 195	53 783 425	33 828 235	33 071 261
0.1	17 438 055	53 903 011	30 010 086	26 801 543

Table 4.5 Latencies (cc) when evaluating 20 trained agents using the DDSU pattern. All 20 agents were trained for 5000 episodes each on System 2 with different constant α configurations.

Note that for both $\alpha = 0.5$ and $\alpha = 0.3$, deadline misses occurred during evaluation. The misses in question were due to two different problems, one of which is that some symbols were being scheduled far earlier than they should have been. This caused them to interfere with and cause misses for symbols that were scheduled closer to their deadline. The other misses were because of a late scheduling, similar to those seen in Figure 4.2. These kinds of schedulings should not be able to have the highest Q-value. Moreover, the symbols that miss should have been adjusted by the scheduler not to, which could possibly be solved by punishing a miss more harshly, see Section 4.5.

Looking at Table 4.5, it seems like the lower α values are generally associated with a decrease in latency, although $\alpha = 0.3$ has a higher average and median than $\alpha = 0.5$. There is still a high variation in latency though, at least for every constant α that has been tested. The window of variation seems to diminish when using a linearly decaying α, as can be seen in Table 4.6.

On average, all configurations from Table 4.6 outperform the baseline, although $\alpha = 0.1 \rightarrow 0.01$ has a higher max value than the baseline. Both $\alpha = 0.5 \rightarrow 0.01$ and $\alpha = 0.1 \rightarrow 0.01$ had runs that failed due to deadline misses. These misses were of the same type as for the constant $\alpha = 0.3$. $\alpha = 0.3 \rightarrow 0.01$ had no misses. Given that deadline misses can be rare, this is not necessarily a guarantee that $\alpha = 0.3 \rightarrow 0.01$ is sufficient for removing deadline misses.

	Latencies (cc)			
α	Min	Max	Average	Median
$0.5 \rightarrow 0.01$	16 306 815	28 280 631	17 974 625	17 190 288
$0.3 \rightarrow 0.01$	16 383 945	28 056 285	18 344 790	16 753 218
$0.1 \rightarrow 0.01$	16 409 655	32 138 639	19 169 745	16 836 514

Table 4.6 Latencies (cc) when evaluating 20 trained agents using the DDSU pattern. All 20 agents were trained for 5000 episodes each on System 2 with different linearly decaying α configurations.

In general, the difference between different α configurations in Table 4.6 is not

large, and it is not easily discernible which one is best. However, when $\alpha = 0.1 \rightarrow 0.01$, the latency is higher in general, in particular the max value and the average. It performs the worst out of the linearly decaying configurations.

The other two configurations, on the other hand, have very similar minimum and maximum latencies. The issue that remains is that there seems to be a tendency for a trained agent to sometimes have a significantly higher latency than what is typical, which is seen in the difference between the max values and the medians. When $\alpha = 0.3 \rightarrow 0.01$, the tendency seems to be somewhat higher, which is the reason for the higher average in comparison to when $\alpha = 0.5 \rightarrow 0.01$. Still, $\alpha = 0.3 \rightarrow 0.01$ has a lower median, which indicates that when it does not have these outliers, it has lower latency in general. For this reason, both configurations will be tested in Section 4.5.

It can also be seen from Figure 4.5 that the resulting average Q-values no longer contain the fluctuation that was brought up in Section 4.3. The other linearly decaying configurations of α exhibit similar behavior.

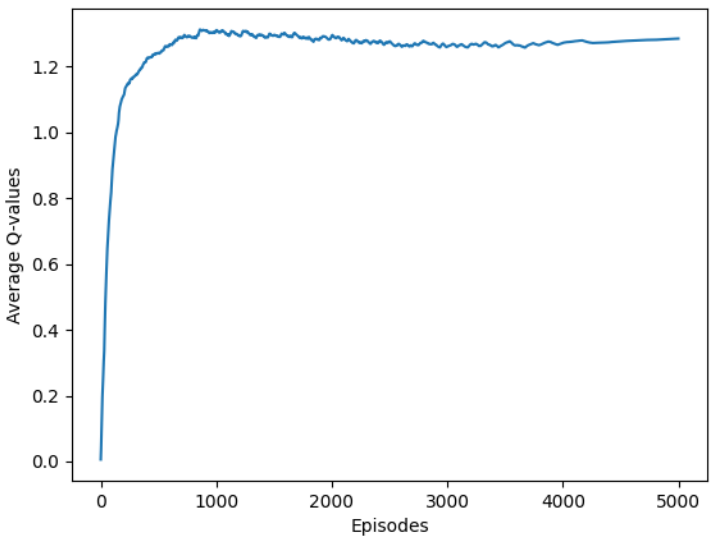

Figure 4.5 The average Q-values of an agent with a linearly decaying α from 0.3 to 0.01 trained on System 2.

4.5 Reward Function

Another factor that affects the Q-values is the reward function. While switching to linearly decaying α values decreased the latency and made the results more consistent, the fact that there may still be differences in how two trained agents behave indicates that maybe some actions are considered to be of similar or equal value. Of course, exploration/exploitation as well as future rewards could also affect this.

There are some things that are important to consider when designing a reward function for this problem. The primary one is the goals: strongly discourage deadline misses while minimizing the latency. Minimizing the latency, when scheduling DL symbols, means that the symbols should be scheduled as close to their deadline as possible. However, the closer the symbol is to its deadline, the higher the risk of a deadline miss. In other words, the goals are somewhat conflicting. Also important to keep in mind is that the punishment for a deadline miss should be harsh. No action which causes some symbols to miss their deadline but otherwise offers high long-term rewards should be determined to be the best one.

The reward function used thus far has been the one Trulsson (2021) used, as defined in Equation 3.1. Although, since the actions for UL symbols are currently statically decided, the equation has been somewhat simplified, see Equation 4.1.

$$r(x) = \begin{cases} e^{-k_{DL} \cdot x} & \text{for DL} \\ -1 & \text{if deadline miss} \end{cases} \tag{4.1}$$

Currently, the rewards are given using an exponential function. This means that the difference in how two symbols are rewarded is not proportional to their differences in latency. Also, a larger difference between rewards would allow the agent to more easily differentiate between the worth of different actions. Hence, an alternative reward function is defined in Equation 4.2, where k is a design constant and symbol time is the time between symbol arrivals. It has been designed with the factors discussed above in mind. In comparison to Equation 4.1, there will be a larger difference in reward between different latencies.

$$r(x) = \begin{cases} 5(1 - \frac{x}{k \cdot \text{symbol time}}) & \text{for DL} \\ -25 & \text{if deadline miss} \end{cases} \tag{4.2}$$

Additionally, the punishment for missing a deadline has been increased, and is five times as high as the maximum possible reward. This serves to further lower the probability of actions that cause deadline misses having high long-term Q-values. A situation that is important to take into account is when there are actions where some but not all symbols miss their deadlines, either regularly or randomly, as shown in Section 5.1. In these cases, a punishment of -1 may not be enough to bring down the Q-value of the action in the long run. The action may still be chosen through exploration and have positive results if it rarely results in misses, as well as increasing in value due to future rewards.

Both reward functions were tested in order to properly compare them with both of the α configurations that were deemed to have potential. The results are presented in Table 4.7, note that the results for Equation 4.1 are taken from Section 4.4. There were no deadline misses using Equation 4.2.

Configurations		Latencies (cc)			
α	Reward Function	Min	Max	Average	Median
$0.5 \rightarrow 0.01$	Equation 4.1	16 306 815	28 280 631	17 974 625	17 190 288
$0.5 \rightarrow 0.01$	Equation 4.2	16 426 795	22 494 355	17 247 934	16 946 043
$0.3 \rightarrow 0.01$	Equation 4.1	16 383 945	28 056 285	18 344 790	16 753 218
$0.3 \rightarrow 0.01$	Equation 4.2	16 401 085	21 397 395	16 978 497	16 769 595

Table 4.7 Latencies (cc) when evaluating 20 trained agents using the `DDSU` pattern. All 20 agents were trained for 5000 episodes each on System 2 with different reward functions.

Overall, it seems like Equation 4.2 performs better, for both α configurations. The maximum outlier values are smaller, going from around 28 million cc to 22-21, meaning that the interval within which the results fluctuate is far smaller. The averages and median values are also lower. In addition, the difference between the values is also smaller, indicating that outlier values are also less common.

Using Equation 4.2, the two α configurations perform similarly. The $0.3 \rightarrow 0.01$ α configuration has lower values than the $0.5 \rightarrow 0.01$ configuration in general. Based on this, these are the settings that will be used going forward, i.e., $\alpha = 0.3 \rightarrow 0.01$ and the reward function as defined in Equation 4.2.

4.6 Future Rewards Discount

The reward given from an action is not the only thing that affects its Q-value. An important part of the Q-learning's update function is the future reward, reflecting the current highest possible reward for the next state. If an action results in a state that has a high long-term value, naturally this affects the value of the action. The parameter γ is the future rewards discount, deciding how much of an impact the future reward has.

Currently, γ has a value of 0.99. However, there might be reason not to let the future reward influence the Q-value to such a degree. There are two main factors within the current system that are the reason for this. The first is that while the scheduler's actions affect the environment, not all parts of the environment are reflected by the state. In fact, the current state space, consisting of job type, slot state and slot progress, is not affected by the scheduler's actions at all, only by the passage of time.

Even if this was not the case, there is another factor to consider. For optimally scheduled DL symbols the wait time before being sent to the system means that the jobs' impact on the environment is delayed. The next state, however, depends on the state of the next symbol to be scheduled, which is far sooner. It does not matter which action is chosen, the next state is the same independent of the action.

A consequence of this is that an exploratory action with a poor reward will see an increase in its Q-value based on the maximum possible reward in the next state, even though it is not deserved.

Therefore, there is reason to consider that perhaps the current value for γ does not give the best possible results. When γ is lowered, the agent cares less about the rewards of the next actions. For example, with $\gamma = 0.5$, only 50% of the maximum value for the next state is considered, which in turn only is affected by 50% of the value of its own next state, meaning that 25% of the second-next state's value is taken into account. In short, with $\gamma = 0.5$ the agent does not look very far into the future, which is why it is the lowest value that will be tested.

Presented in Table 4.8 below are the results with different values for γ, although note that the results for $\gamma = 0.99$ are taken from Table 4.7.

	Latencies (cc)			
γ	Min	Max	Average	Median
0.99	16 401 085	21 397 395	16 978 497	16 769 595
0.95	16 392 515	21 398 921	17 286 472	16 886 053
0.9	16 366 805	18 423 605	16 890 028	16 718 175
0.8	16 358 235	19 254 895	17 635 623	17 592 315
0.5	17 379 591	20 548 965	19 063 966	19 011 413

Table 4.8 Latencies (cc) when evaluating 20 trained agents using the `DDSU` pattern. All 20 agents were trained for 5000 episodes each on System 2 with different γ configurations.

Overall, there is not a large difference in the performance between varying values of γ. The two highest tested values have the largest difference between their lowest and highest latencies, while the two lowest values have the highest averages and medians, which means that their latencies are generally higher. The best-performing value for γ seems to be 0.9, since it has the lowest max latency, average and median. Its smallest latency is only slightly larger than for $\gamma = 0.8$. Therefore, $\gamma = 0.9$ will be used going forward.

The similarities between the results for different γ in Table 4.8 indicate that the reasoning for trying out different γ still holds, in the sense that it does not seem to affect the results very strongly. However, lowering γ beyond a certain point seems to yield worse results, which means that there might be something more to it. It is possible that this is largely due to the fact that while the agent does not affect the

state with its actions, it still affects the environment. A larger γ is used to propagate the future results of an action, not necessarily depending on the state to reflect the change that occurs in the environment in order to be effective.

4.7 Exploration & Exploitation

Another of our goals was to evaluate and possibly improve the exploration part of the Q-learning, particularly if it was possible to avoid deadline misses during exploration. However, at this point we have realized that the agent should not avoid deadline misses during exploration. Instead, they should be found. If a particular action has any chance of failing, even if it is not often, the agent should have witnessed that during training, so that it has learned to prioritize less risky actions.

The one exception is when different symbols' schedulings impact each other in a way that the Q-learning cannot mitigate. If the agent chooses exploitation for one symbol and schedules it close to its deadline but chooses exploration for another symbol, the jobs might be sent to the BFJ queue at the same time. This could cause the exploiting action to miss at no fault of its own. A higher ε will likely exacerbate this issue.

Thus, the balance between exploration and exploitation is still very important. Exploring is useful in order to discover potential better actions, while exploitation is good for evaluating how good the current "best" action is. Right now, this is determined using the ε-greedy algorithm with a constant $\varepsilon = 0.1$. There is still a variation in how a trained agent will perform, and this is based on when and what it chose to explore at different points during the training. Therefore, it can be valuable to investigate how different values of ε, including ones that decay over the course of training, affect the results.

As noted in Section 2.2, there are benefits of both higher and lower ε, as well as a possibility of gaining the advantages of both by decaying from a higher value to a lower. Therefore, different values have been tested in Tables 4.9 and 4.10.

	Latencies (cc)			
ε	Min	Max	Average	Median
0.3	17 283 795	21 433 201	19 404 119	19 564 178
0.1	16 366 805	18 423 605	16 890 028	16 718 175
0.01	16 375 375	17 156 771	16 743 266	16 736 841

Table 4.9 Latencies (cc) when evaluating 20 trained agents using the `DDSU` pattern. All 20 agents were trained for 5000 episodes each on System 2 with different constant ε configurations.

	Latencies (cc)			
ε	Min	Max	Average	Median
$0.5 \rightarrow 0.01$	16 289 675	17 808 091	17 001 948	17 041 076
$0.3 \rightarrow 0.01$	16 306 815	17 712 295	16 903 469	16 953 850
$0.1 \rightarrow 0.01$	16 323 955	17 259 611	16 673 717	16 658 185

Table 4.10 Latencies (cc) when evaluating 20 trained agents using the `DDSU` pattern. All 20 agents were trained for 5000 episodes each on System 2 with different linearly decaying ε configurations.

It is not entirely obvious which configuration is best. $\varepsilon = 0.3$ sticks out in the sense that it has the highest latencies across the board. $\varepsilon = 0.5 \rightarrow 0.01$ also performs slightly worse than the other configurations, in the sense that its average and median are above 17 million cc, although notably it has the lowest minimum value. This is likely connected to the issue mentioned above, as these configurations were the ones featuring the highest values of ε.

Of the constant configurations, $\varepsilon = 0.1$ and $\varepsilon = 0.01$ have very similar results. The biggest difference between them is their max latencies, where $\varepsilon = 0.01$ is over a million cc lower, making it seemingly the best performing configuration out of the two of them.

Among the decaying configurations, $\varepsilon = 0.3 \rightarrow 0.01$ has a slightly lower minimum latency than $\varepsilon = 0.1 \rightarrow 0.01$, but a higher max, average and median, meaning that $\varepsilon = 0.1 \rightarrow 0.01$ overall performs better.

Overall, $\varepsilon = 0.1$ and $\varepsilon = 0.1 \rightarrow 0.01$ have similar results, although the latter has the lower minimum, average and median latency. Because of this, $\varepsilon = 0.1 \rightarrow 0.01$ will be used going forward. Note that the differences in performance at this point are not large, but in the range of ten-thousand cc as opposed to millions of cc as seen in sections above. As such, even though a certain configuration is used going forward, this does not mean that the others are bad and would not perform well.

4.8 Recap

To recap: After the modifications have been done, the following settings are used, listed in Table 4.11. The action space has been decreased, in the sense that the agent now only dynamically decides the wait times of DL symbols. The state space consists of: slot state, slot progress and job type.

α	Reward function	γ	ε
$0.3 \rightarrow 0.01$	Equation 4.2	0.9	$0.1 \rightarrow 0.01$

Table 4.11 The settings that are used moving forward.

5

Results

It is evident that the changes done in Chapter 4 have made a difference in comparison to Trulsson (2021), both in the sense that the latency for a trained agent has decreased, as well as not having seen any deadline misses since the reward function was updated. However, the only evaluation pattern that has been looked at is `DDSU`, and the only system that has been presented is System 2.

Note that the same TDD pattern will be used for training as in Chapter 4, listed in Table 4.1, but all of the evaluation patterns listed in Table 4.2 will be used. This means that the agent is evaluated on patterns that it has not seen before.

This chapter aims to take a closer look at how both the systems perform for all evaluation files, and comparing the results to the baseline and random schedulers. Refer to Table 3.1 for the system information. The behavior of the agent when scheduling symbols in different situations is also investigated in more detail.

Additionally, in Section 5.1, the agent is stress-tested by seeing how it performs when adding noise to symbol arrival times, as well as looking at what happens when the number of BACs are decreased.

System 1

The latencies for the different schedulers using the different evaluation files are listed in Table 5.1. Note that the baseline scheduler and the agent **never had any misses**, while the random scheduler always did, for every single run. Additionally, since the baseline always makes the same decisions, it only has one possible result.

The number of training episodes used has been increased to 10 000 for System 1. This is because there is only one carrier, which means fewer symbols and the agent learning less per training episode.

DDSU				
	Latencies (cc)			
Scheduler	Min	Max	Average	Median
Baseline	5 811 988			
Agent	2 861 333	2 925 225	2 880 166	2 878 090
Random	4 212 117	4 777 840	4 515 487	4 551 304

DDDSUDDSUU				
	Latencies (cc)			
Scheduler	Min	Max	Average	Median
Baseline	7 102 564			
Agent	3 696 021	3 770 197	3 715 111	3 712 778
Random	5 356 821	6 107 715	5 706 226	5 719 695

DDDSUUDDDD				
	Latencies (cc)			
Scheduler	Min	Max	Average	Median
Baseline	7 347 081			
Agent	3 522 641	3 606 244	3 540 017	3 538 113
Random	5 448 097	6 179 565	5 748 698	5 731 179

Table 5.1 Final evaluations on System 1 for all evaluation patterns found in Table 4.2. All 20 agents were trained for 10 000 episodes each.

The agent performs better than the baseline for all evaluation files, especially in comparison to the results Trulsson (2021) saw, where the agent barely performed better. This implies that the agent has gotten better at finding schedules that result in lower latencies, and does so consistently. Additionally, since all evaluation files have good results, this means that if the agent performs well on one file, it probably performs well on all files.

Curiously enough, the random scheduler seems to generally have lower latencies than the baseline, although this is largely irrelevant due to the fact that it, due to its randomness, never is able to evaluate on a file without causing deadline misses.

The average Q-values for the agent are shown in Figure 5.1. Notably, the average value seems to shoot up very quickly in the beginning of training, and slowly stabilize throughout it. Judging by both the results and this figure, it seems like increasing the number of training episodes to 10 000 was enough to guarantee that the agent has converged.

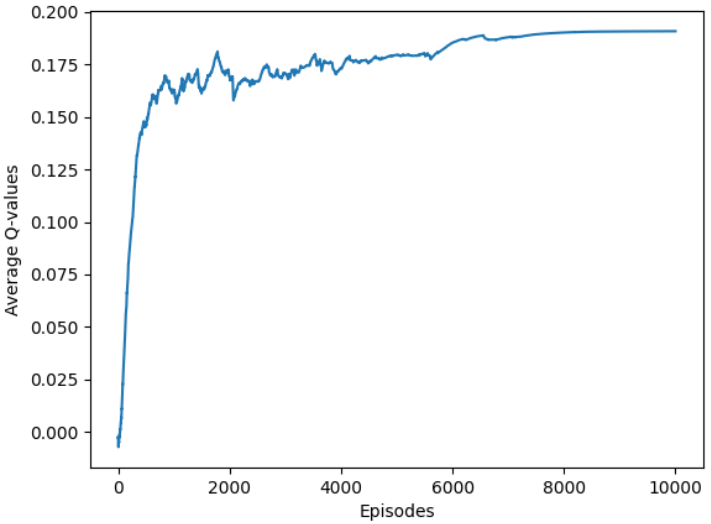

Figure 5.1 Average Q-values of one trained agent using System 1.

A Closer Look While the latencies for the agent look promising, it is also important to take a look at how it actually schedules symbols. Thus, the results of one of the trained agents has been picked to be investigated further. The results that were picked perform closely to the median latency for all evaluation files. It was picked with the assumption that there should not be a significant difference in behavior between different trained agents, given the small differences in latency. The latency with the different evaluation patterns for the chosen trained agent is listed in Table 5.2.

Pattern	Latency (cc)
DDSU	2 878 090
DDDSUDDSUU	3 712 778
DDDSUUDDDD	3 539 398

Table 5.2 Final evaluations on System 1 for one of the agents, comparing the resulting latencies (cc) for each pattern. The agent was trained for 10 000 episodes.

Figure 5.2 shows how the jobs were distributed across the BACs throughout evaluation on the `DDSU` pattern. It is visible in the beginning that the agent schedules the DL symbols closer to their deadline than the baseline does. The block of BTA jobs is further to the right for the agent than the baseline, which indicates that the jobs are scheduled later by the agent. Something similar also seems to be happening in the UL/DL-switch, but both situations warrant further investigation.

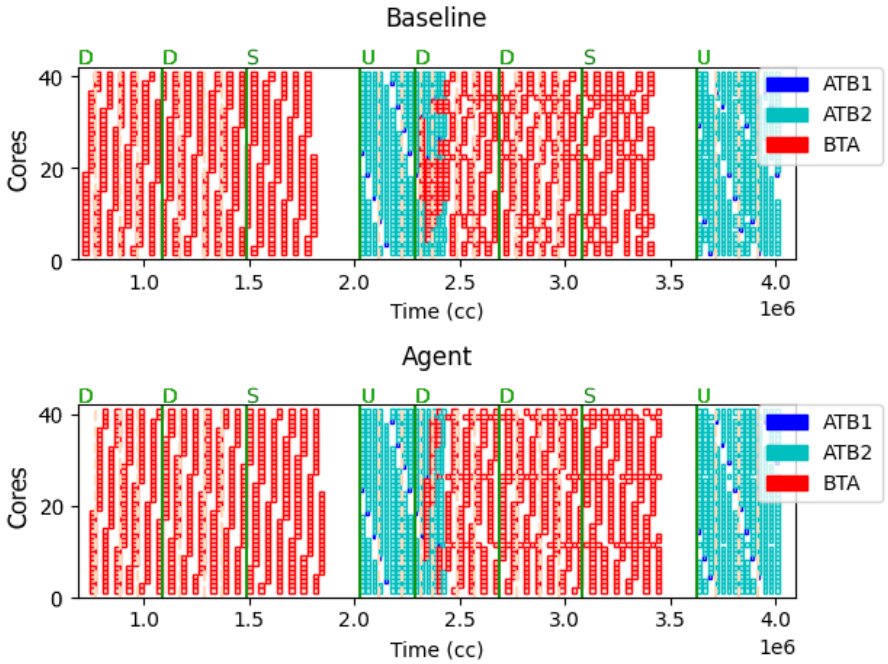

Figure 5.2 Job scheduling for the `DDSU` pattern in System 1, where each rectangle represents a job. The green lines represent changes in slot state. The y-axis represents the core upon which the job was completed, and the x-axis is time in cc. The dotted light brown lines represent the places where slot progress would shift, even though it is only actually used in the UL/DL-switch.

Figures 5.3 and 5.4 show the situations from Figure 5.2 in further detail. Figure 5.3 shows a situation where only DL symbols are scheduled, and it seems like the agent has learned to schedule the symbols as close to their deadline as possible with the given time precision. The situation that Figure 5.4 shows is the UL/DL-switch, where at times all of the BACs are at work concurrently. This means that jobs might be waiting in the BFJ queue for a while before they get processed. DL symbols cannot be scheduled as close to their deadline here, because ATB jobs might arrive and be picked by the BACs to be worked on before the BTA jobs are sent to the BFJ queue, which comes with the risk that the BTA jobs miss their deadlines. Still, the agent seems to have learned that it is able to push the jobs closer to their deadlines than what the baseline does, while still being early enough to avoid missing them.

The fact that the agent is able to schedule the symbols differently in these two scenarios points to the state space being precise enough to allow this. It schedules the symbols close to their deadlines when it is possible and schedules them further away when it is dangerous.

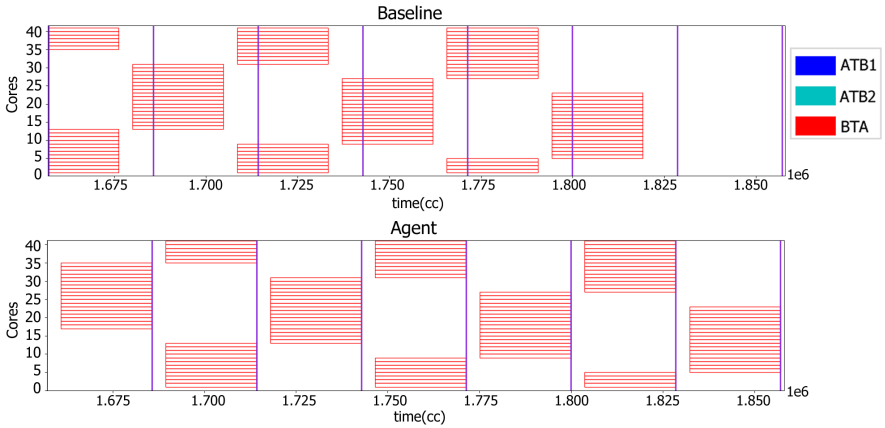

Figure 5.3 Job scheduling for a period of only DL transmissions in System 1, where each rectangle represents a job. The purple lines represent deadlines. The agent schedules the jobs later than the baseline does, and close to their deadlines.

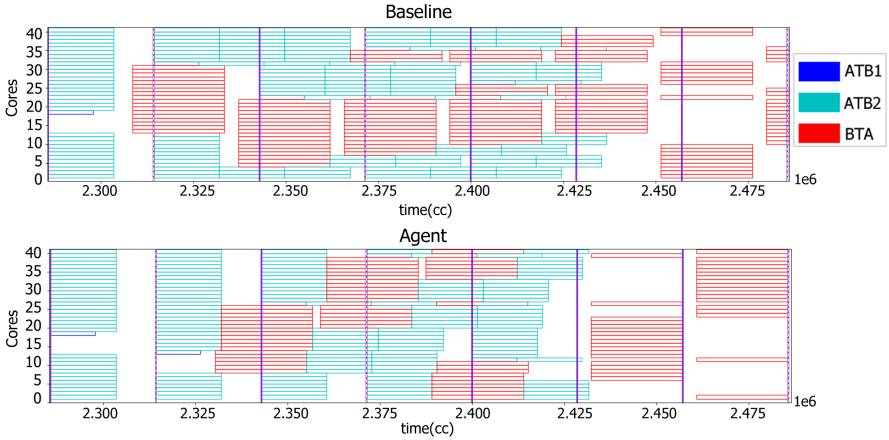

Figure 5.4 Job scheduling for the UL/DL switch in System 1, where each rectangle represents a job. The purple lines represent deadlines. The agent still schedules the jobs later than the baseline does, but not as close to their deadlines as in Figure 5.3. This shows that the agent has learned to prioritize avoiding deadline misses in high-contention phases.

System 2

Listed in Table 5.3 are the latencies for the different schedulers using the different evaluation files. Note that the baseline scheduler and the agent **never had any misses**, while the random scheduler always had, for every single run. Since most modifications were evaluated on System 2, the results shown for the pattern DDSU are the same as from Section 4.7, specifically Table 4.10 with $\varepsilon = 0.1 \rightarrow 0.01$.

DDSU				
	Latencies (cc)			
Scheduler	Min	Max	Average	Median
Baseline	28 941 268			
Agent	16 323 955	17 259 611	16 673 717	16 658 185
Random	37 837 209	42 558 802	40 432 963	40 583 508

DDDSUDDSUU				
	Latencies (cc)			
Scheduler	Min	Max	Average	Median
Baseline	36 455 900			
Agent	21 851 483	23 027 099	22 238 096	22 230 089
Random	48 361 910	51 226 598	49 987 243	50 159 505

DDDSUUDDDD				
	Latencies (cc)			
Scheduler	Min	Max	Average	Median
Baseline	36 193 620			
Agent	19 767 688	21 423 224	20 194 580	20 076 971
Random	49 562 892	53 691 296	51 464 265	51 291 535

Table 5.3 Final evaluations on System 2 for all evaluation patterns found in Table 4.2. All 20 agents were trained for 5000 episodes each.

Once again, the agent always performs better than the baseline, and does so with quite some margin on all evaluation files. There is also not a huge variation in how the trained agents perform. Compared to the results in Trulsson (2021), where the agent had a twice as high latency as the baseline, this is positive. Once again, the random scheduler shows the difficulty of finding a good schedule, although now it does perform consistently worse than the baseline, and always has deadline misses. In comparison, the baseline and agent never miss.

The average Q-values for one of the runs are seen in Figure 5.5. It looks similar to the Q-values for System 1.

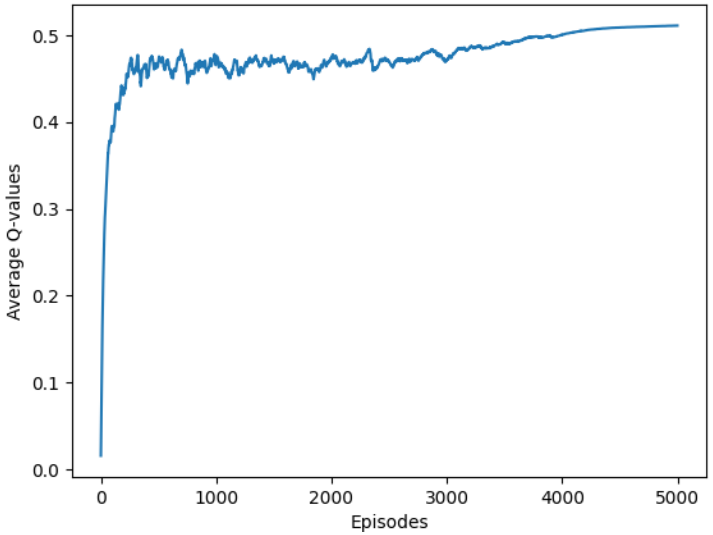

Figure 5.5 Average Q-values of one trained agent using System 2.

A Closer Look Once again, looking more closely at one of the trained agents will reveal more about the its behavior. The latencies for the different evaluation files for this trained agent are listed in Table 5.4.

Pattern	Latency (cc)
DDSU	16 649 615
DDDSUDDSUU	22 160 003
DDDSUUDDDD	19 930 518

Table 5.4 Final evaluations on System 2 for one of the trained agents, comparing the resulting latencies (cc) for each pattern. The agent was trained for 5000 episodes.

Figure 5.6 shows the schedule for the `DDSU` evaluation pattern. While it is a bit difficult to discern the details due to the number of symbols in this file, it seems that BTA jobs are generally pushed closer to their deadlines.

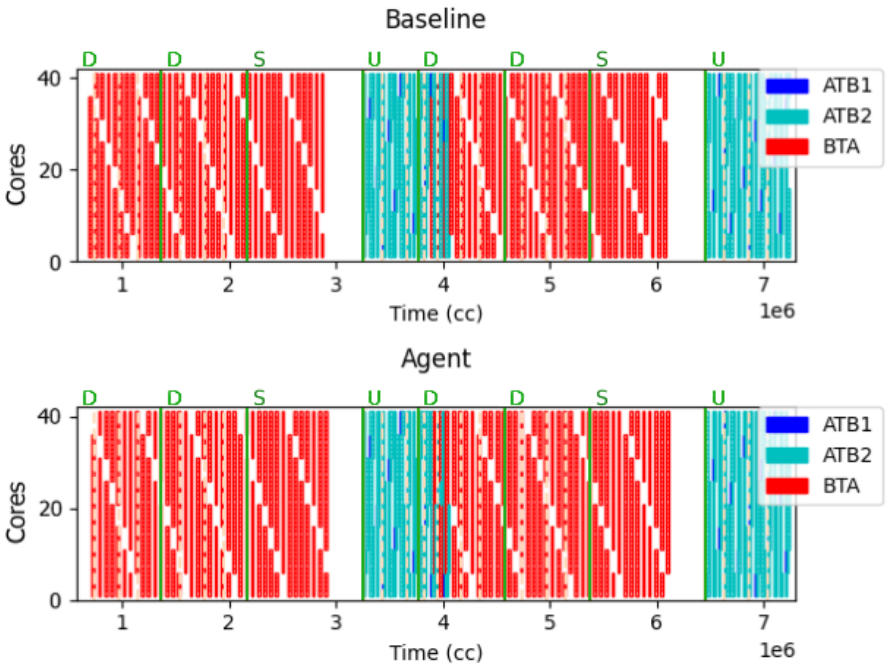

Figure 5.6 Job scheduling for the `DDSU` pattern, where each rectangle represents a job. The green lines represent changes in slot state. The dotted light brown lines represent the places where slot progress would shift, even though it is only actually used in the UL/DL-switch.

Zoomed in versions of Figure 5.6 are shown in Figures 5.7 and 5.8, to more easily investigate the agent's scheduling behavior. Figure 5.7 shows the baseline and agent's decisions in a situation where only DL symbols are arriving. The agent schedules its symbols such that the generated jobs finish as close to their deadline as possible with the given time precision, which is the desired behavior.

Figure 5.8 shows the situation in the UL/DL-switch. This is similar to System 1, with all BACs working at certain points.

Because of this, the agent seems to have learned to schedule DL symbols such that the generated jobs are processed in between the ATB jobs that arrive close to the BTA jobs' deadlines. Compared to the baseline, which schedules the BTA jobs before the ATB jobs, this leads to a lower latency. Since this is System 2, five symbols arrive at the same time. They have the same state and thus get the same scheduling decision. However, it is visible in the figure that it could theoretically be possible to minimize the latency further by scheduling some of the symbols closer to their deadline, which is not possible with the current state space.

Note also that the BTA jobs start to get scheduled closer to their deadline when there are no more ATB jobs in the system, meaning that the slot progress variable manages to create a difference between the two situations. Still, these jobs do not finish as close to their deadlines as the jobs in Figure 5.7.

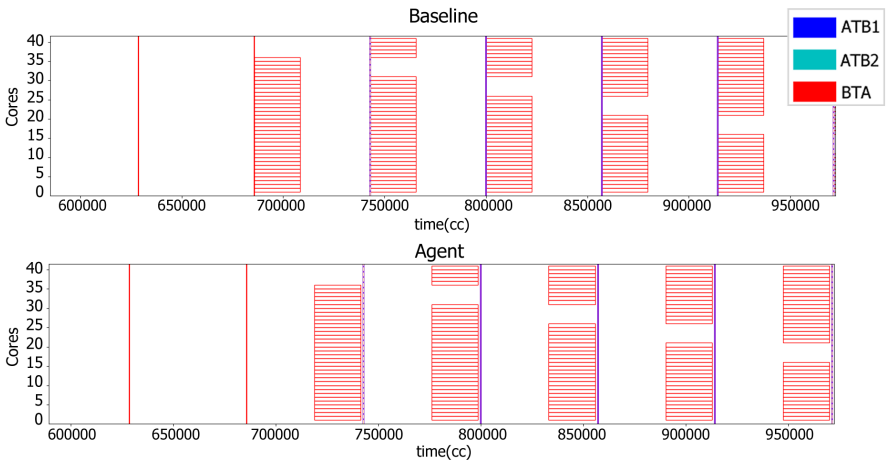

Figure 5.7 Job scheduling for a period of only DL transmissions in System 2, where each rectangle represents a job. The purple lines represent deadlines. The agent schedules the jobs later than the baseline does, and close to their deadlines.

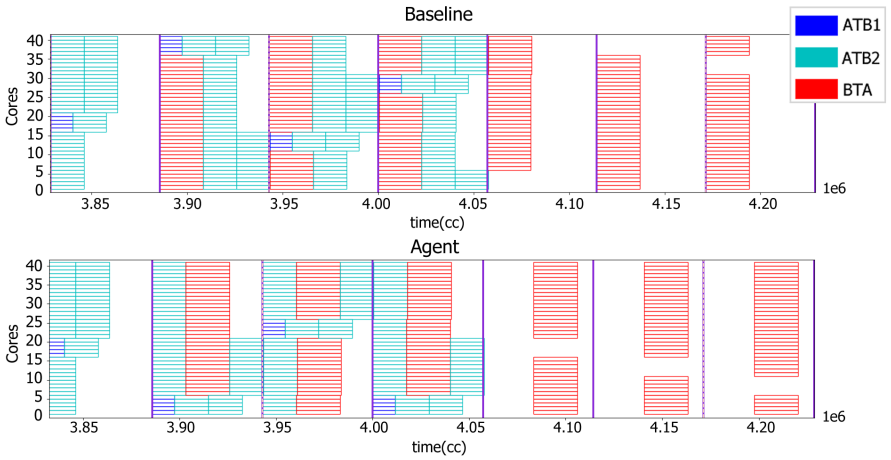

Figure 5.8 Job scheduling for the UL/DL switch in System 2, where each rectangle represents a job. The purple lines represent deadlines. The agent still schedules the jobs later than the baseline does, but not as close to their deadlines as in Figure 5.7. This shows that the agent has learned to prioritize avoiding deadline misses in high-contention phases.

5.1 Robustness

Now that the agent performs well, with low latency and avoiding deadline misses, it is time to test the algorithm's robustness and check how the agent performs in more difficult situations. The reason for this is to further investigate how fit the agent is for a potential real implementation.

One way this is done is by introducing noise in the system, such that symbol arrivals in both training and evaluation files are not as consistent. Another way to do it is by checking how the agent performs with a lower amount of cores, such that the period of high contention for the BACs becomes longer. Both of these are tested separately in the sections below.

Noise

In a realistic setting, things might happen that cause symbols to not always arrive at consistent times, which the scheduler should ideally be able to handle. For the purpose of these tests DL symbols may be either late or early, whereas UL symbols can only be late. The noise for each symbol is randomly generated within parameters chosen to accurately reflect a real situation. The deadlines remain unaffected by noise.

Deciding the wait time for UL symbols is trivial, because they are ideally sent to the BFJ queue as soon as possible regardless of arrival time. However, it can

become problematic for DL symbols. This is because the scheduling needs to be adapted to the worst possible scenario, that is, the latest possible symbol arrival. If it is adapted, then the symbols arriving earlier than that will have a wait time that is too short to fully minimize their latencies. If it is not adapted, then the wait time might be too long for the latest possible symbol arrival, instead resulting in those symbols missing their deadlines.

However, it would still be ideal to fully minimize the latency. This can be done by changing how the time when jobs are sent to the BFJ queue is decided. Instead of deciding a time to wait, which is dependent on the time of arrival, the agent can instead decide how soon before its deadline a symbol should be sent. The deadline distance is decided in the interval between the arrival time that the symbol would have had without noise, and the deadline. This way, the actual arrival time does not matter.

Of course, avoiding deadline misses is the most important part. Introducing noise causes a situation where few symbols may miss their deadlines, and is one of the reasons for punishing those misses more harshly.

System 1 Table 5.5 shows the impact that latency noise has, both for deciding wait time and for deciding deadline distance. As expected, the latencies are generally higher when adding noise and deciding wait time, whereas deciding deadline distance makes the agent perform more similar to the case without noise.

	Latencies (cc)			
Configuration	Min	Max	Average	Median
No noise	2 861 333	2 925 225	2 880 166	2 878 090
Noise (wait time)	2 995 130	3 071 707	3 029 071	3 030 370
Noise (deadline distance)	2 874 353	2 998 409	2 908 648	2 905 727

Table 5.5 Comparison with noise in System 1 when deciding wait time as well as deadline distance, evaluated on the `DDSU` pattern (result without noise taken from Table 5.1). All 20 agents were trained for 10 000 episodes each. The agents featuring noise were both trained and evaluated with noise. During each training episode, the training file was re-generated using the same training pattern but with randomized noise.

Figure 5.9 shows the latencies for some symbols for both approaches. The figure displays the issue with deciding wait times, where latencies are inconsistent since they depend on the arrival time of a symbol. This inconsistency is removed when deciding deadline distance.

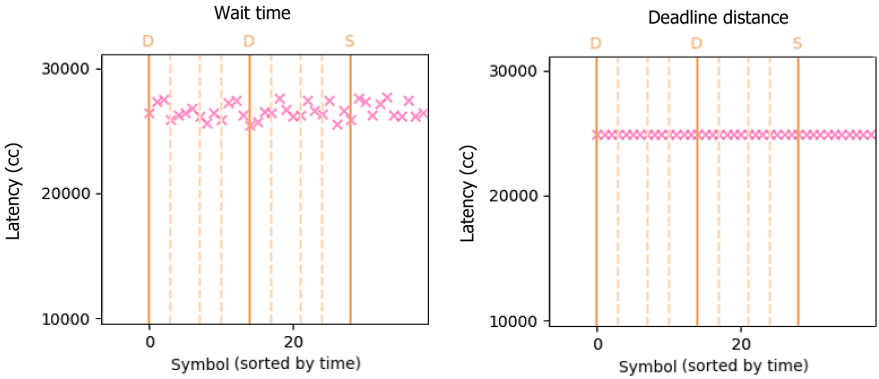

Figure 5.9 The differences in latency between deciding wait time and deadline distance for System 1 with noise. Evaluated on the DDSU pattern, showcasing situations where only DL symbols arrive. Each cross corresponds to a symbol. The y-axis is the latency, and the x-axis is the symbol sorted by time. The yellow lines represent changes in slot state. The dotted light brown lines represent the places where slot progress would shift, even though it is only actually used in the UL/DL-switch.

System 2 System 2 does not seem to be affected in quite the same way by noise as System 1, as seen in Table 5.6. Instead, adding the noise seems to slightly lower the average and median latency when deciding wait time. Deciding deadline distance has very similar results, if slightly lower average and median.

	Latencies (cc)			
Configuration	Min	Max	Average	Median
No noise	16 323 955	17 259 611	16 673 717	16 658 185
Noise (wait time)	16 152 597	17 242 334	16 556 288	16 551 730
Noise (deadline distance)	16 074 881	17 263 678	16 545 440	16 455 596

Table 5.6 Comparison with noise in System 2 when deciding wait time as well as deadline distance, evaluated on the DDSU pattern (result without noise taken from Table 5.1). All 20 agents were trained for 10 000 episodes each. The agents featuring noise were both trained and evaluated with noise. During each training episode, the training file was re-generated using the same training pattern but with randomized noise.

Despite the latencies being lower when introducing noise while deciding wait time, the same behavior seen in System 1 can be seen for System 2, see Figure 5.10. Notably, some of the latencies when deciding wait time are lower than the latencies found when deciding deadline distance, which is interesting because the latter latency is as low as possible given the time precision.

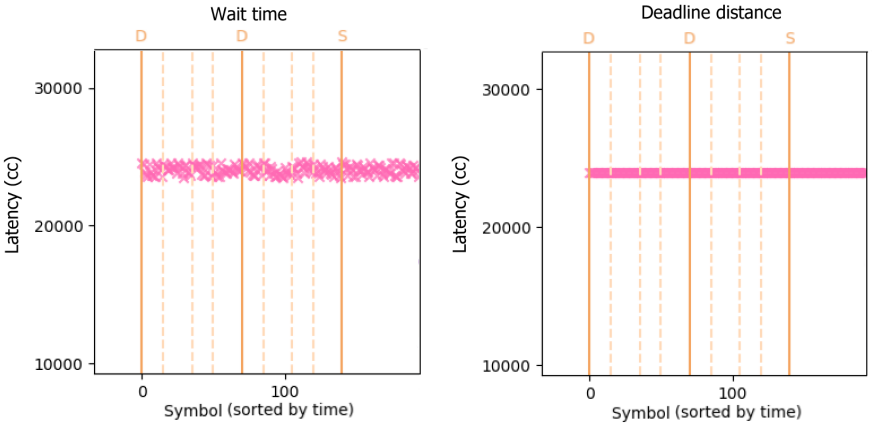

Figure 5.10 The differences in latency between deciding wait time and deadline distance for System 2 with noise. Evaluated on the DDSU pattern, showcasing situations where only DL symbols arrive. Each cross corresponds to a symbol. The y-axis is the latency, and the x-axis is the symbol sorted by time.

These results could be connected to the precision when deciding wait time/deadline distance. The deadline distance is decided using a pre-decided, never changing time interval. When deciding the wait time, the time interval depends on the arrival time, which is different for each symbol because of noise. With maximum noise, this means that the time interval in which the wait time is decided grows smaller, which increases the time precision since the number of time steps is constant. It is possible that this allows those symbols to be scheduled just a bit closer to their deadline. However, there is probably a limit for how much an increased precision can help.

The reason why this same behavior is not seen for System 1 could be connected to the differences between the systems. Since System 1 has a higher numerology than System 2, it has a higher subcarrier spacing. The higher subcarrier spacing means that the time between symbol arrivals is smaller, which leads to earlier deadlines. Thus the time interval for System 1 is shorter both in regards to deciding wait time and deadline distance. The noise would also have a larger effect.

Increasing Contention

By decreasing the number of cores, the contention for the BACs will increase. This means that scheduling symbols becomes more difficult, since there is less room for error without risking missing deadlines. There will be a point when there simply are too many symbols to schedule for every symbol to be able to meet its deadline. A similar effect could also be achieved by increasing the number of carriers.

System 1 As seen in Table 5.7, the agent manages to avoid deadline misses for a bit longer than the baseline as the number of cores decrease.

Scheduler \ # Cores	40	32	30	29	28	27
Baseline						
Agent						

Table 5.7 Deadline stress test of System 1. Green means that a run had no misses, whereas red means there were misses on at least one evaluation file.

Figure 5.11 shows the distribution of jobs when the agent starts to miss deadlines at 28 cores, and the jobs that miss in both the agent and the baseline are ATB jobs. The issue seems to lie towards the end of the UL/DL-switch, which makes sense since it does have more contention than the areas with purely DL or UL transmissions. If there are too many jobs for the BACs to work on given the time frame, it is a given that some jobs will not be picked up by them until it is too late.

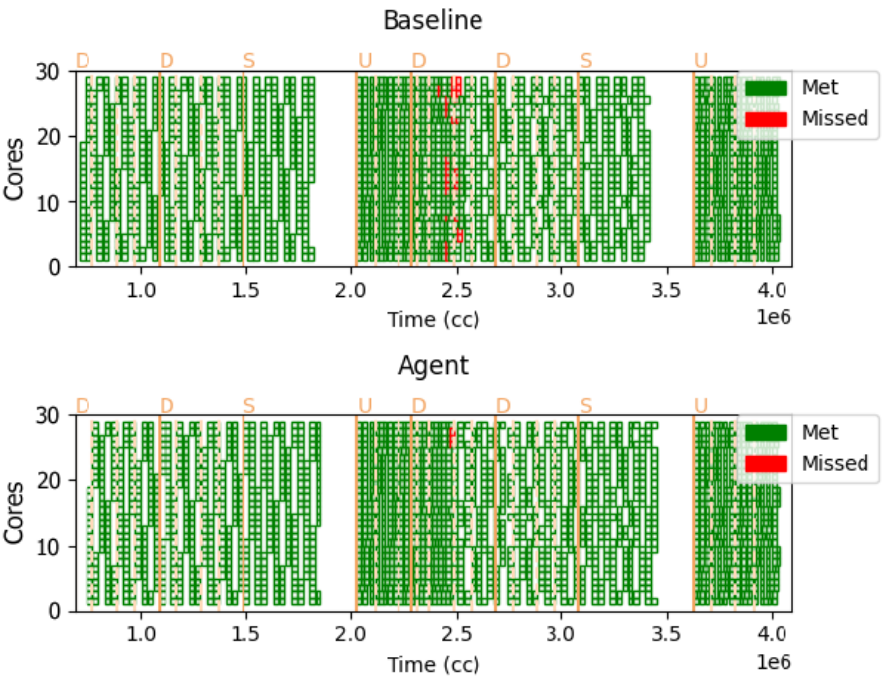

Figure 5.11 Job scheduling for the DDSU pattern in System 1 with 28 cores, where each rectangle represents a job. The yellow lines represent changes in slot state.

A closer look at the exact scheduling is shown in Figure 5.12. In general, it seems like the issue is that there simply is not enough space to properly schedule the symbols such that they meet their deadlines. Interestingly, the agent shows a slightly different behavior than the baseline. In the beginning of the UL/DL-switch, the BTA jobs are scheduled earlier compared to the baseline, while towards the end they seem to be scheduled slightly later. It is possible that this allows for more ATB jobs to be worked on compared to the baseline, in the part where no BTA jobs are worked on. This probably accounts for the differences in how many cores the baseline and agent manage respectively as well as the number of misses, since it is ATB jobs that miss.

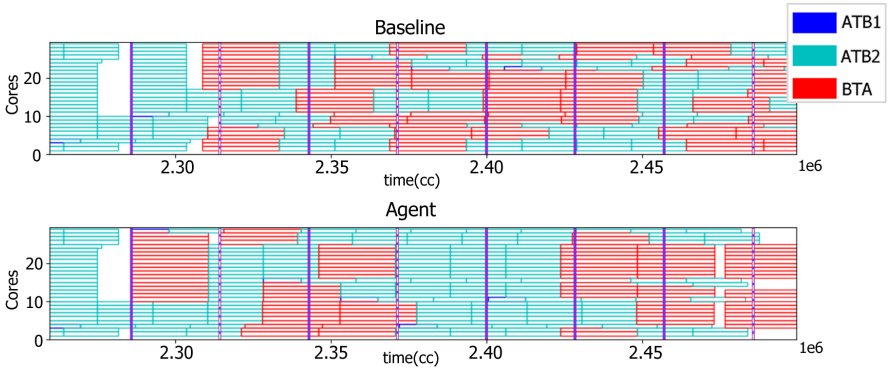

Figure 5.12 Job scheduling for the UL/DL switch in System 1 with 28 cores. Each rectangle represents a job, and the purple lines represent deadlines. For both the baseline and the agent, it is ATB jobs that miss, because they have a lower priority than the BTA jobs. The baseline has more misses than the agent does, visible in Figure 5.11, whereas the only deadlines missed by the agent are the three right-most ATB2 jobs in this image.

System 2 In System 2, the agent and the baseline have similar performance, both managing to schedule without deadline misses at 26 cores, but failing if the number of cores are decreased beyond that. Figure 5.13 shows the distribution of jobs across the BACs over time. Perhaps not unexpectedly since they start missing deadlines at the same number of cores, the baseline and the agent perform similarly. The main difference is that the agent seems to schedule BTA jobs slightly earlier than the baseline does, which results in fewer ATB jobs left in the BFJ queue towards the end of the UL/DL-switch.

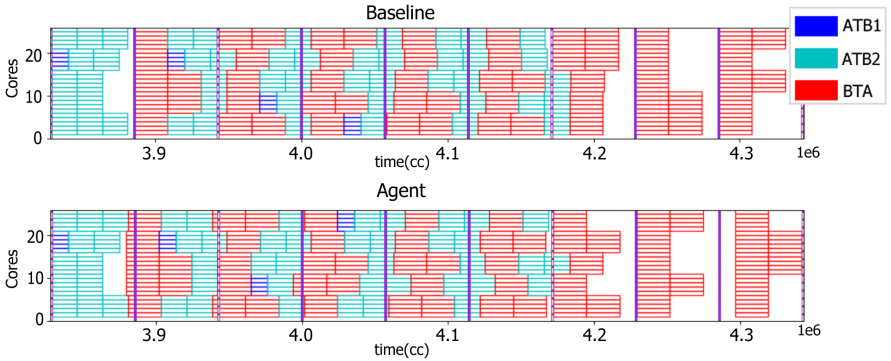

Figure 5.13 Job scheduling for the UL/DL switch in System 2 with 25 cores. Each rectangle represents a job, and the purple lines represent deadlines. For the baseline, the entire right-most block of ATB2 type jobs miss their deadline. For the agent, the two right-most ATB2 type jobs miss.

There is also an interesting thing to point out with the behavior during periods with only DL symbols arriving. With five carriers, too many jobs are generated from the five symbols for all of them too be worked on by the BACs at once. The scheduling in this situation is shown in Figure 5.14. Since all of these symbols have the same state, they need to have the same scheduling. Fortunately, the agent learns to schedule the symbols in such a way that all of the symbols meet their deadline, even though it does lead to an increase in latency, since multiple symbols are not scheduled as close to their deadline as they could be.

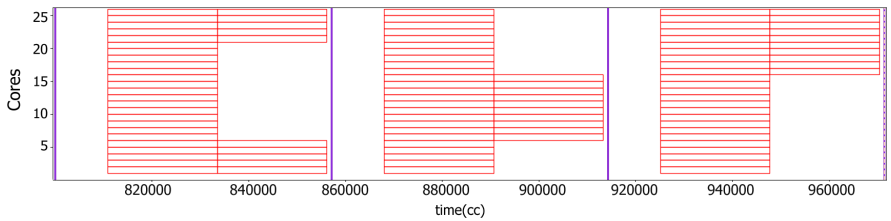

Figure 5.14 Job scheduling with 25 cores for a period of only DL transmissions in System 2, where each rectangle represents a job. The purple lines represent deadlines.

6

Discussion

Overall the agent has improved, consistently outperforming the baseline. Since the reward function was updated, it manages to consistently find low-latency schedules without missing deadlines, and it does so without needing an extreme amount of training. For System 1, the agent has an average latency over the different evaluation files corresponding to approximately 50% of the baseline's latency. The corresponding number for System 2 is 58%.

It is not entirely fair to compare the agent's new performance regarding latency with its performance in the work of Trulsson (2021). The reason for this is mainly that the agent is now evaluated using multiple patterns that more accurately reflect reality. This includes sections with only DL transmissions where there is a lot to gain latency-wise in comparison to the baseline. Trulsson (2021) does not have these sections.

The behavior, on the other hand, can be compared. The agent now manages to schedule DL symbols close to their deadline in the UL/DL-switch for both systems. For Trulsson (2021), this only occurred in System 1. The changes made to the state space guarantee a more uniform scheduling.

State Space

The state space has received heavy modifications, the only part of how it initially looked that is left being the job type, which in and of itself could be removed as long as the agent only dynamically schedules DL symbols. The new parts of state space consist of slot state and slot progress. In comparison to the old state variable symbol index, the precision has decreased, which is good in the sense that symbols with similar or the same best schedulings now have the same state, which decreases the amount of training episodes needed to find the optimal scheduling for each symbol.

However, a certain precision is still needed, because otherwise symbols that do not have the same ideal scheduling have the same state, which is not desirable. This is the reason that slot progress was introduced, to take into account the need for different schedules in the different parts of the UL/DL-switch.

In the end, there are two situations that warrant different schedules: the UL/DL-switch, that requires a more careful scheduling of symbols to ensure keeping dead-

lines, and the situation that is **not** the UL/DL-switch, where the agent can schedule symbols closer to their deadline safely.

If the state space were to be optimized to the extreme in favor of training time, this means that all that is really needed is a way to distinguish between the aforementioned situations, and use them as the two states. This would increase performance and decrease the training time, since there would be no need to distinguish between a D slot and an S slot. It can be achieved using the same underlying information that is used for slot state and progression.

However, having a smaller state space does come at the cost of some precision. It was seen in System 2 that sometimes when symbols arrive at the same time, the latency could theoretically be lowered by scheduling some of the symbols later. This does not happen because they have the same state. The state space is too small to handle this situation. Conversely, precision could be increased by introducing a state space variable that allows for the agent to distinguish between these symbols. This could be something similar to the earlier proposed waiting state. Then, the latency could be lowered although at the cost of an increased training time.

Delays in the System

An important aspect to take into account when working on this problem is the possible delay between the agent making a scheduling decision, and the generated jobs actually being sent to the system. This means that symbols that arrive to the system at similar times might not end up in the BFJ queue at the same time, and thus end up not affecting each other, even though the Q-learning will look at the state of the next symbol when updating the Q-value.

Another issue is that there is no inherent mechanism in Q-learning that will always properly blame a scheduling that causes another symbol to miss. To give a proper example: Suppose that symbol `A` arrives to the system, and is scheduled close to its deadline at time `T`. Later on, symbol `B` arrives to the system and is scheduled with a shorter wait time than `A` at time `T-1`, just before symbol `A`, perhaps as an exploratory action. As a result, not all of the jobs generated from symbol `A` will get picked up by the BACs, since they are working on the jobs from symbol `B`, resulting in symbol `A` missing its deadline. The action taken for symbol `A` will be punished for a deadline miss even though it actually had a good schedule, and it was `B` that was the true offender. Furthermore, Q-learning only looks at future rewards, and from symbol `B`'s perspective, symbol `A` is in the past. Symbol `B` will therefore not get any feedback that it caused a miss. Additionally, the Q-value of symbol `A`'s scheduling might not recover after being wrongfully punished. Then, a worse scheduling could be valued higher at the end of the training.

The idea of introducing the waiting state, i.e., keeping track of the number of scheduled jobs, came from the aforementioned issue. The basic idea still holds: Why should the agent forget that it just made a scheduling decision that will affect the viability of scheduling upcoming symbols in a certain way? Ideally, the agent

should be able to keep track of when jobs have been scheduled, and thus know that it is not a good decision to schedule more symbols at a time when the contention for the BACs is high. However, this would likely require moving away from Q-learning and implementing a more model-based method. As it stands, the agent manages to find low-latency schedules while avoiding deadline misses despite these issues, and as long as that holds there is no compelling reason to move away from Q-learning.

The precision of state space likely has an effect, because at the core of it, this is a question of symbol interplay, how the scheduling of one symbol affects another. When multiple symbols have the same state, they will have the same behavior, barring exploratory actions. When they have the same actions, they will not interfere with each other. However, symbols can also interfere with each other during exploitative actions if they do not have the same state, and therefore have different actions. Thus, the higher the amount of different states, the larger the risk of symbols interfering with each other. This could be the reason for why the symbols in the D-slot of the UL/DL-switch do not always find the optimal schedule, as seen in Figure 5.8. This is the only slot where slot progress, and therefore a more precise state space, is actually used.

Lowering ε should also have an effect on this issue, since a low ε means that the probability of an unfortunate exploratory action is smaller. It could also be possible to decrease the time interval which the agent schedules DL symbols in, since sending them to the BFJ queue with a short wait time is rarely desired, even in the more difficult scheduling situations. If symbols cannot have a short enough wait time to interfere with other symbol that arrived at different times, there will not be an issue.

However, it is not possible to lower ε all the way down to 0 and pick actions purely though the ε-greedy approach. Since the Q-matrix is 0-initialized, all actions have equal value in the beginning of the training. This means that whichever action that is picked and does not cause deadline misses will be what the greedy algorithm goes with, likely resulting in the agent getting stuck with a sub-optimal policy.

Configurations

In the interest of time, when testing new configurations they were trained and evaluated 20 times. However, given the element of randomness inherent with exploration, it is not certain that 20 runs were actually enough to properly determine which configurations were best. For certain tests, like comparing letting α decay linearly with having it constant, the results were obvious, but for the different ε configurations, the results were close to each other, meaning that while one configuration was picked to move forward with, it might not necessarily have made a difference.

Something that might have made a difference was the order in which particular settings were tested. While adjusting α made a significant difference in latency, lowering it as well as making the agent perform more consistently, the change might not have been as dramatic if, for example, ε was investigated first. It could also have lead to a bigger difference for different ε.

6.1 Future Work

Scaling down the action space for the sake of simplicity seems to have paid off. There does not seem to be any issues in assigning priority statically, and neither for deciding wait time for UL symbols statically. However, it is still possible that letting the agent decide BFJ PRB size dynamically could have benefits.

For example, lowering the BFJ PRB size leads to a lower job processing time, potentially leading to a lower latency if there is no shortage of BACs. However, this also leads to an increase in the amount of jobs generated, and therefore a larger overhead, making this a trade-off between memory and latency. Regardless, it could be worth investigating in the future.

It could also be possible to investigate γ further, since the tests of different configurations did not yield the expected results. This could include further testing of different values.

7

Conclusions

The Q-learning algorithm and its parameters have been investigated and improved. Some parts of action space have been removed to simplify the algorithm, although there is still room to re-add those actions. To be able to train and showcase the agent's performance in a more multifaceted way, the training/evaluation data has been extended to cover more scheduling situations.

The state space has received comprehensive changes, and the agent no longer depends on time with the removal of symbol index, paving the way for a real implementation. The size of the Q-matrix is now independent of the traffic file, and is also far smaller. An important aspect when discussing state space is the trade-off between precision and training time. As of right now, some degree of precision is sacrificed for training time, but there are arguments for going either way.

Several parameters for the Q-learning have been investigated. Implementing linear α decay had a large impact on the performance. It is important that ε is not too large, since exploratory actions may interfere with exploitative ones, which the Q-learning cannot always properly reward. Changing γ does not seem to make a large difference. The reward function has also been updated, mostly in the sense that the punishment for missing deadlines has increased.

There are some attributes inherent to the system that a model-based method most likely would be able to take advantage of, although the Q-learning still performs very well compared to the baseline.

In conclusion, the agent manages to schedule symbols in a way that is almost optimal, and without deadline misses, on both the different system configurations. It also performs well when introducing noise, indicating that it could work in a real implementation.

Bibliography

Dahlman, E., S. Parkvall, and J. Sköld (2018). *5G NR, The Next Generation Wireless Access Technology*. Academic Press.

Even-Dar, E., Y. Mansour, and P. Bartlett (2003). "Learning rates for Q-learning." *Journal of machine learning Research* **5**:1.

Lei, W., P. CK Soong, L. Jianghua, W. Yong, B. Classon, W. Xiao, D. Mazzarese, Z. Yang, and T. Saboorian (2020). *5G system design*. Springer International Publishing.

Osseiran, A., J. F. Monserrat, and P. Marsch (2016). *5G mobile and wireless communications technology*. Cambridge University Press.

Rommer, S., P. Hedman, M. Olsson, L. Frid, S. Sultana, and C. Mulligan (2019). *5G Core Networks: Powering Digitalization*. Academic Press.

Scherfke, S., O. Lünsdorf, and P. Grayson (2020a). *Simpy, discrete event simulation for python*. Visited on 2022-05-11. URL: `https://simpy.readthedocs.io/en/latest/`.

Scherfke, S., O. Lünsdorf, and P. Grayson (2020b). *Simpy, shared resource primitives*. Visited on 2022-05-04. URL: `https://simpy.readthedocs.io/en/latest/api_reference/simpy.resources.html`.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. MIT Press.

Trulsson, P. (2021). *Dynamic Scheduling of Shared Resources using Reinforcement Learning*. MA thesis. Lund University, SE-221 00 LUND.

Wiering, M. A. and M. Van Otterlo (2012). "Reinforcement learning". *Adaptation, learning, and optimization* **12**:3, p. 729.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
June 2022

Document Number
TFRT-6162

Author(s)
Elin Wilson Andersson
Johan Håkansson

Supervisor
Jonas Korsell, Ericsson
William Tidelund, Ericsson
El-Tayeb Bayomi, Ericsson
Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Improving a Reinforcement Learning Algorithm for Resource Scheduling

Abstract

This thesis aims to further investigate the viability of using reinforcement learning, specifically Q-learning, to schedule shared resources on the Ericsson Many-Core Architecture (EMCA). This was first explored by Patrik Trulsson in his master thesis Dynamic Scheduling of Shared Resources using Reinforcement Learning (2021). The shared resources complete jobs assigned to them, and the jobs have deadlines as well as a latency. The Q-learning based scheduler should minimize the latency in the system. Most importantly, it should avoid missing deadlines. In this work, the Q-learning algorithm was tested on a simulation model of the EMCA that Trulsson built. Its performance was compared to a baseline and random scheduler. Several parts of the Q-learning algorithm were evaluated and modified. The action and state space have been made smaller, and the state space has been made more applicable to the real system. The reward function, as well as other parameters of the Q-learning, were altered for better performance. The result of all of these changes was that the Q-learning algorithm saw an increase in performance. Initially, it performed slightly better than the baseline on only one of the two configurations it was evaluated on, but in the end it performed significantly better on both. It also handles the introduction of noise to the simulation without a significant decrease in performance. While there are still things that might require further investigation, the algorithm always performs better than a baseline scheduler provided by Ericsson and is overall more suited for a real implementation due to the changes that have been done.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-65

Recipient's notes

Security classification

http://www.control.lth.se/publications/