Dynamic Scheduling of Shared Resources using Reinforcement Learning

Patrik Trulsson

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6143
ISSN 0280-5316

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

Printed in Sweden by Tryckeriet i E-huset
Lund 2021

Abstract

The goal of the thesis is to simulate the Ericsson Many-Core Architecture, EMCA, and implement a dynamic scheduler for the system using reinforcement learning methods. The system contains shared resources that receive and complete jobs. Also, the deadlines and latency definitions can change depending on the job type. The scheduler should aim to avoid missing deadlines as well as aim to reduce the overall latency in the system. A python simulation has been implemented of the Ericsson Many-Core Architecture and two reinforcement learning based schedulers have then been developed and used for different configurations. They are evaluated by comparing their performance to a random and a static scheduler. The first scheduler uses Q-learning and the second uses a version of Q-learning with a neural net that approximates the Q-function. The results showed that the second version experienced issues with convergence which caused deadline misses and poor latency. The regular version of Q-learning showed promising results, avoiding deadline misses and was able to reduce the latency below that of the static scheduler for one of the systems. There are still some issues that could be addressed as well as avenues to explore regarding the scheduler. Furthermore, in order to apply the scheduler to the real system, some modifications are necessary. However, the simulations show that the reinforcement learning can successfully be used as a scheduler on the EMCA for different configurations.

Acknowledgements

I would like thank my supervisors at Ericsson, William Tidelund and Jonas Korsell, for the opportunity to work on this thesis and for the assistance in completing it. Furthermore, I would like thank my supervisor at Lund University, Karl-Erik Årzén, for his help on this thesis. Finally, I would like to thank my examiner Bo Bernhardsson for valuable input regarding the reinforcement learning algorithm.

Contents

Acronyms

ATB Antenna to beam 16, 20, 29, 31, 38, 40, 42

BAC Beamforming accelerator core 10, 11, 15, 16, 17, 18, 20, 26, 27, 30, 31, 35, 36, 39, 40, 42

BTA Beam to antenna 16, 20, 29, 31, 38, 40

DL Downlink 11, 14, 15, 16, 17, 18, 20, 26, 27, 28, 31, 35, 39, 40, 42, 43

DQN Deep Q-learning Network 24, 25, 26, 28, 29, 34, 35, 36, 37, 38, 43, 44

EMCA Ericsson Many-Core Architecture 3, 10, 11, 12, 14, 16, 19, 44

FDD Frequency Division Duplex 11, 15, 16, 42, 44

LTE Long Term Evolution 14, 15

MDP Markov Decision Process 21, 22, 23, 24, 26

NR New Radio 14, 15

OFDM Orthogonal Frequency Division Multiplexing 14, 15, 16, 17

PRB Physical Resource Block 16, 17, 20, 27, 31, 35, 39

RL Reinforcement learning 20, 21, 23, 26

SJ Super Job 16, 18, 19, 20, 27, 28

TD Temporal-difference 22, 23

TDD Time Division Duplex 11, 15, 16, 17, 42

UL Uplink 11, 14, 15, 16, 17, 18, 20, 26, 27, 28, 31, 35, 38, 39, 42, 43

1

Introduction

Scheduling allows for the distribution of shared resources and occurs in many different systems. However, in cases such as the Ericsson Many-Core Architecture, EMCA, there are many different configurations. This means that either a scheduler has to be designed specifically for each configuration or a general scheduler must be used that ensures deadlines are met, but might be sub-optimal for certain configurations. Reinforcement learning could make it possible for a scheduler to be generated automatically, which is of commercial interest in telecommunications where specifications are in flux. An example of this is the ongoing introduction of the new generation of wireless access technology in the form of 5G/New Radio [Erik Dahlman, 2018]. Also, using reinforcement learning could lead to schedulers that lower system latency compared to static schedulers that might be overly cautious. Furthermore, reinforcement learning is an area of active research and applications, such as the one provided by this thesis, provide challenges that could be of theoretical interest.

In order to run 5G signal processing Ericsson uses a tailormade system on a chip, the EMCA, in the base station. In the EMCA there are many concurrent jobs as well as shared resource pools. In the system, one or several carriers are used that can send and receive information from users through modulation of radio signals. Beamforming is performed when the base station sends and receives information. Beamforming requires a large amount of calculations which can increase latency in the system. This can be mitigated through the use of beamforming accelerator cores, BAC, that are part of the EMCA. These are accelerators used to decrease the computation time of the calculations. However, there are only a limited amount of these cores in the system and thus it would be optimal to share these resources as effectively as possible. The beamforming is divided into jobs which the cores pick from a priority queue.

Scheduling in this context means dividing the beamforming calculations between BAC through deciding the size of the jobs, which implicitly determines the number of jobs. The scheduler can also wait before sending jobs to the priority queue in order to increase overall latency. Finally, each job is associated with a

priority which the scheduler is able to set. This will determine which jobs are completed first if there is competition for the BAC.

Furthermore, latency definitions differ for uplink, recieving from user, and downlink, sending to user. For uplink, all calculations should be completed as soon as possible, as opposed to downlink where calculations should be completed as late as possible without missing deadlines. This is due to the different operations necessary for sending versus receiving from a user and is explained further in depth in Section 2.2.

The scheduling algorithm needs to consider that it has to run in many products and environments and has to take the following into account:

- The job execution time, periodicity, deadline, and number of jobs in the system will change based on deployment and configuration.
- The job runs in a multi-core environment with other concurrent jobs.
- Some jobs in the pool have strict deadlines and some jobs more relaxed deadlines.
- The system can run in time division duplex, TDD, or frequency division duplex, FDD. This determines when uplink and downlink beamforming work is generated by the simulation.
- Resource constraints include: number of cores, core processing power, latency and buffering.

1.1 Objectives

The thesis objectives can be summarized as follows. Implement a multi-core scheduling environment in Python which focuses on modeling the EMCA part that handles the scheduling of the beamforming computations. Schedule jobs in the simulated environment according to known Uplink, UL, or Downlink, DL, profiles and configurations at Ericsson. Implement scheduling algorithms in the simulated environment which makes sure that all the job deadlines are met with as low latency as possible for both UL and DL. The algorithm should be adaptive and possible to apply to any configuration with regards to: number of jobs, job time, job periodicity, job deadline and number of resource pool cores.

The first simulation model should focus on the beamforming accelerator resource, when this one is known, complexity of job chaining between resources can be added.

1.2 Delimitations

The reinforcement learning algorithms are limited to two versions of Q-learning. Also, while the EMCA simulator aims to model the real system, some simplifications are made, see Chapter 3. Furthermore, the possible actions that the scheduler could make do not correspond to all actions that could be taken in the system. There are valid arguments for both increasing and decreasing the number of actions for the scheduler, see Chapter 6.

Two systems with different configurations are used in order to evaluate the schedulers.

1.3 Related Work

Using reinforcement learning for dynamic scheduling is not unprecedented. For example, [Wang and Usher, 2005] where Q-learning is used on a simulated system to schedule jobs. Also, there is also an example in mobile communications where a reinforcement learning algorithm is used to schedule cellular network traffic [Chinchali et al., 2018]. Thus, there is promise in using reinforcement learning for dynamic scheduling.

1.4 Outline of the Report

A brief overview of the report structure and chapters is given below.

Introduction

The introduction gives a motivation for the necessity of a dynamic scheduler in a system with shared resources. It also gives account of the goal, objectives and delimitations of the report. Finally, it presents an outline of the report.

Background

The background contains descriptions of the system and its components. It also introduces technical details required to understand the challenges inherent in the system.

Model

This chapter provides an overview of how the simulated system is implemented and how it may differ from the real system.

Reinforcement learning

Gives an account of algorithms and concepts used in the reinforcement learning based schedulers.

Result

Presents and discusses the results of two reinforcement learning based schedulers applied on four systems. Also, provides an account of how the input and output of the reinforcement learning algorithms are derived.

Discussion

The discussion chapter discusses general matters regarding the scheduler design. Furthermore, possible improvements are presented here.

Conclusion

An overview of the conclusions that can be made from the results as well as the implementation in general.

2

Background

The following sections aims to explain the necessary technical details to give the reader a basic understanding of the system. Furthermore, certain parameters, state/action spaces and design variables correspond to concepts explained in this chapter.

2.1 Transmission

The EMCAs are used in a base station which receives and transmits data to a user. A transmission is called downlink, DL, when data is transmitted from the base station to a user. Likewise, when a transmission is sent from a user to the base station it is called Uplink, UL.

LTE, Long Term Evolution, is a set of technical specifications and standards for mobile communications first released in 2009. New Radio, NR, is a newer standard first available in 2017. NR was designed in order to make better use of new technologies but inherits a lot of structures from LTE [Erik Dahlman, 2018]. LTE is thus often more associated with 4G and NR is more often associated with 5G.

Orthogonal frequency-division multiplexing, OFDM, is a transport technology for communication systems based on using several overlapping subcarrier frequencies and sending data in parallel. These subcarriers each have a center frequency, of maximum signal power, and the spacing in frequency between subcarrier center frequencies is called subcarrier spacing [Ergen, 2009].

The largest unit in a 4G/LTE or 5G/NR transmission is called a radio frame, which is 10 ms. A radio frame consists of 10 subframes of 1 ms. In both 4G/LTE and 5G/NR OFDM symbols are used and each subframe is divided into slots, each slot containing 14 OFDM symbols, see Figure 2.1.

The numerology determines the subcarrier spacing in kHz, which in turn determines the time intervals between symbols in the system, see Figure 2.2. Thus for different numerologies there are different amount of slots per subframe [Erik Dahlman, 2018].

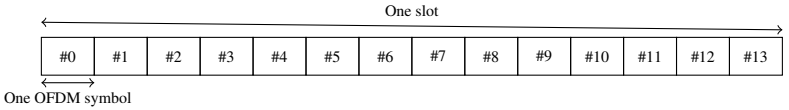

Figure 2.1 One slot is defined as 14 OFDM symbols.

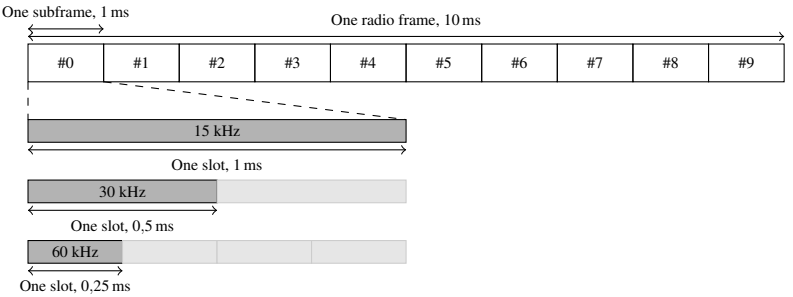

Figure 2.2 Radio Frame, subframes and slots for 15 kHz, 30 kHz and 60 kHz numerologies.

Numerology is usually referred to by a number, for example, numerology 0. Each number corresponds to a subcarrier spacing, see Table 2.1. LTE supports numerology 0 and NR supports numerology 0-3.

Number	Subcarrier spacing kHz
0	15
1	30
2	60
3	120

Table 2.1 Numerology number conversion to subcarrier spacing.

The duplex schemes determine how UL and DL transmissions are separated. For time division duplex, TDD, a single carrier frequency is used and transmissions are distinguished by time. Each OFDM symbol is either assigned as UL or DL. For frequency division duplex, FDD, the transmissions are carried out on different carrier frequencies. Thus each OFDM symbol is both UL and DL [Erik Dahlman, 2018]. Several data streams, called layers, can be sent simultaneously to one or several users. This is done in order to increase throughput where each layer can be beamformed [*Advanced antenna systems for 5G networks*]. However, increasing the number of layers increases the complexity of the beamforming process causing longer job times in the BAC.

Beamforming

Beamforming refers to the ability to direct radio energy toward a specific receiver through constructive interference of radio signals by manipulation of the phase and amplitude of the signals. This enables higher signal strength at the user and thus higher throughput. Likewise, beamforming when receiving refers to the ability to collect the signal energy from a transmitter [*Advanced antenna systems for 5G networks*]. However, beamforming requires the completion of large amount of matrix calculations which must be finished quickly to lower overall latency. Beamforming accelerator cores, BAC, can be used in order to accelerate these calculations. The BAC can in this case be viewed as resources and as previously mentioned, there are benefits to sharing resources.

2.2 Components

The EMCA is used in the lower layers of the base stations and is necessary for the beamforming process in the radio unit. There are several components in the system. However, the components relevant to the scheduler are the carriers and the resource pools in the form of queues and beamforming accelerator cores, BAC.

Carrier

The carriers generate the symbols in a transmission. Each carrier has a bandwidth, duplex scheme and numerology, which will affect the time of arrival and deadline of a symbol. The bandwidth, in kHz, is determined by the number of subcarriers in the carrier as well as the subcarrier spacing measured in kHz per subcarrier. The duplex scheme is either TDD or FDD described in Section 2.1.

Super Jobs

Super jobs, SJ, represent multiple underlying jobs in the system related to beamforming. However, from the scheduler's perspective these are the generated jobs and in the report the terms jobs and super jobs are used equivalently.

A physical resource block, PRB, consists of 12 sub-carriers in the frequency domain and one OFDM symbol in the time domain. The PRB associated with a super job corresponds to beamforming work load and higher PRB causes the super job to take longer to complete by a BAC.

Each symbol in the transmissions translates to the creation of jobs for the beamforming accelerator cores. The job size and number of PRBs is determined so that the whole bandwidth of the carrier the symbol belongs to is used. In TDD, UL and DL symbols correspond to different job types. DL transmissions create Beam-To-Antenna, BTA, jobs whereas UL creates two Antenna-to-beam job types, ATB1 and ATB2. The super job parameters are: the size of the super job in PRB, the size of the associated beamforming job in PRB and the job type: ATB1, ATB2 or BTA.

Each carrier has a sub-carrier spacing in kHz determined by the numerology. Thus, the bandwidth of each carrier can be translated into PRB. For example, a carrier with 100 MHz bandwidth and sub-carrier spacing of 30 kHz, corresponds to 273 PRB. Since PRB consists of 12 sub-carriers in the frequency domain this corresponds to $273 \cdot 12 \cdot 30 \approx 98{,}28\,\text{MHz}$. The whole bandwidth cannot be utilized because guard bands in the bandwidth are needed to avoid interference [Erik Dahlman, 2018]. This means each OFDM symbol corresponds to 273 PRB beamforming work load.

Scheduler

The scheduler is responsible for creating Super Jobs from symbols and determining their properties. Also, it must determine the priority and the time to send them to the priority queue.

Resource Pools

The resource pools consist of a priority job queue and several BACs. The BAC continuously pops jobs with the highest priority from their assigned priority job queue. The BAC is responsible for performing the beamforming computations associated with the job picked from the priority queue.

Latency

In general, latency for jobs should be minimized in the system. However, the desired behaviour differs between UL and DL when using TDD and different latency definitions are used, see Figure 2.3. For UL it is optimal to reduce the wait time before sending the job to the job queue as much as possible. For DL the job should be completed as close to the deadline as possible, without missing it, in order to reduce the margin and, thus, the latency. There are several reasons, UL consists of receiving information and when beamforming is complete the information can be received immediately. For DL the transmission is sent to the user at a determined time that cannot be preempted. Thus, there is no benefit to completing DL jobs early and completing the job late reduces memory usage since the data can be stored for less time.

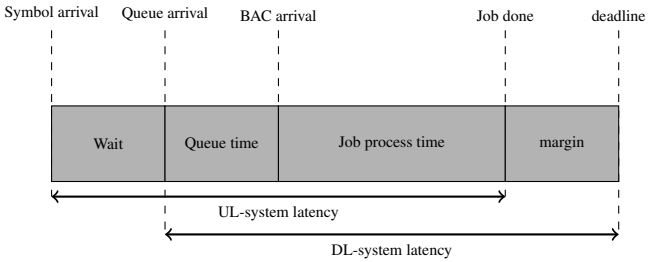

Figure 2.3 Latency definitions for UL and DL. Symbol arrival means the scheduled arrival of the symbol to the system. Queue arrival is the arrival of the job created from the symbol in the job queue. BAC arrival is the job arrival time to a BAC in the system.

2.3 System Overview

In the system, symbols arrive at periodic time intervals determined by the number of carriers, as well as their numerology and duplex scheme. Super job tokens, SJ tokens, are generated from symbols. These tokens are sent to a priority queue and then distributed among beamforming accelerator cores. When a core has received a SJ token it will complete the necessary beamforming computations associated with the token. The time until completion are determined by job and carrier parameters.

The amount of super jobs generated from a symbol, the priority, and when to send the super jobs to the queue, are all determined by the scheduler. The Beamforming Accelerator cores pop super jobs from the queue according to priority and complete them and the associated beamforming computations.

Static scheduler

A static scheduler, provided by Ericsson, can be used on the system which schedules the jobs based only the current symbol job type. Thus the scheduler is restricted to three static actions, one for each job type. However, for sufficient number of BACs it will ensure that all deadlines are met. This scheduler is used as the baseline when comparing other scheduling algorithms.

3

Model

In order to design the algorithm a Python-based simulation model of EMCA was developed. SimPy, a discrete event simulator, was used in order to create the model [Scherfke and Lünsdorf, 2020]. SimPy is able to simulate time and let processes add events to be triggered at a certain timestep or event. Thus, information of future events are hidden from the scheduler, mimicking the real process. The simulated time did not include the runtime of the scheduling algorithm and the job queue. This is a simplification of the real process. However, the time is small compared to the job process time and therefore have a small effect on the latency.

3.1 Simulation Components

Queues

The simulation model uses queues in order to simulate the system, see Figure 3.1. A list of symbols marked with timestamps are extracted from a simulated traffic file. Then, using a SimPy process, the symbols are sent as symbol tokens to a symbol FIFO queue at the time corresponding to the timestamp. The Agent block is responsible for generating and scheduling super job tokens from symbol tokens, as well as handling feedback from the beamforming accelerator cores. The SJ tokens are then sent to a Priority queue where beamforming accelerator cores can retrieve them when available. A result token is generated from a SJ token when the BAC is finished with the SJ token.

Beamforming Accelerator Cores

The beamforming accelerator cores are modelled as a process that continually pops jobs from the SJ queue and sleeps for the estimated time it takes to complete the job. The beamforming job time is estimated using the properties of the SJ token as well as the associated carrier.

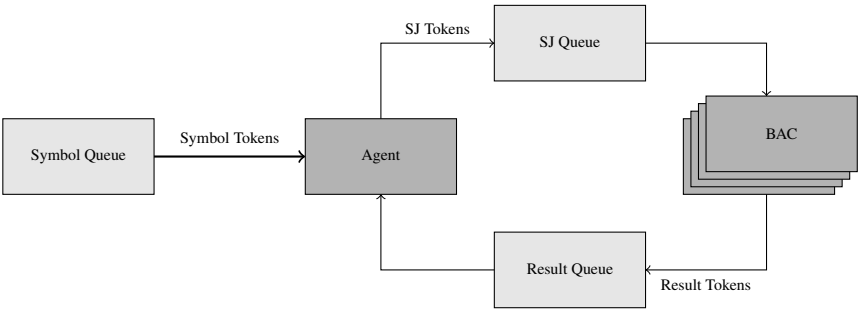

Figure 3.1 Queue structure in the simulation

Scheduling

The Agent in Figure 3.1 is responsible for scheduling the symbol as jobs on the BAC:s. There are three actions that must be determined: SJ PRB size, priority in the SJ token queue and the time to wait until sending the created SJ token to the SJ token queue. In this step the simulation can switch between the static scheduler and the RL scheduler depending on simulation setup.

Jobs and Job Types

All jobs generated from DL symbols in the system are of type BTA but UL symbols generates both ATB1 and ATB2 jobs. This is handled in the simulated traffic file by generating two symbol tokens for each UL symbol, one of type ATB1 and one of type ATB2. ATB1 symbols are handled in a more static way in the current implementation than ATB2 symbols. One ATB1 job is generated from each ATB1 symbol with a fixed 16 PRB size. However, the priority and waiting time before the job is sent to the job queue are determined by the scheduler.

4

Reinforcement Learning

A reinforcement learning, RL, system consists of an agent and an environment, see Figure 4.1. The agent receives the current state, s_t from the environment at time step t. The agent may then influence the environment with an action, a_t, which changes the state of the environment to s_{t+1}. This agent in turn, receives either a reward or a penalty, r_t, for the action-state pair, a_t and s_t. The agent may then choose an action for the next state, s_{t+1}, and the process continues. The goal of the agent is to select the best possible action, depending on the state, in order to maximize the total reward. This is achieved by the agent trying to improve its decisions over many iterations, referred to as the training process [Sewak, 2019]. Additional elements to reinforcement learning are the policy, the value function as well as the model. The policy is the mapping that the agent does between states in the environment and actions. The value function aims to capture long term benefits of certain states or actions. A model is something that reproduces the behaviour of the environment or something that allows inferences on how the environment will behave. Methods that use models are called model-based methods as opposed to model-free methods. Reinforcement learning is often formalized using incompletely known Markov decision processes, MDPs. Markov decisions processes are intended to include the aspects necessary to the agent, the concept of a state, action and goal [Sutton and Barto, 2018].

The initial development of reinforcement learning resulted from the combination of three threads of research in the late 1980s. Learning by trial and error, optimal control using value functions and dynamic programming, as well as temporal-difference learning. The term "optimal control" was minted in the 1950s and refers to the problem of designing a controller that maximizes or minimizes a measure of a dynamical system over time. Dynamic programming is a class of methods that can be used to solve optimal control problems, assuming the system is fully known. Furthermore, Richard Bellman introduced the discrete version of the optimal control process, something that has been previously mentioned, the Markov Decision processes, MDP[Sutton and Barto, 2018].

There are many modern advances in the field of reinforcement learning but a famous example is the computer program known as AlphaGo, which was used to

defeat a world champion in the complex game of Go in March 2016. Reinforcement learning was an important part of the algorithm which allowed it to learn a model of its environment [*AlphaGo*]. This illustrates the potential of the reinforcement learning framework.

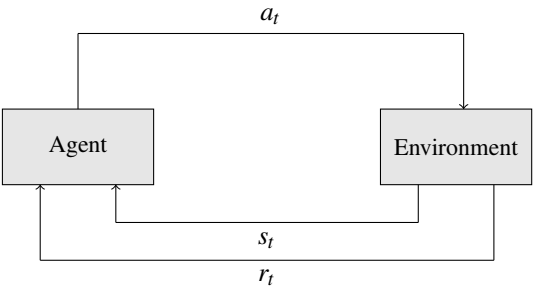

Figure 4.1 The general reinforcement learning system.

4.1 Temporal-difference Learning

The value function, $V(s_t)$, can be defined recursively as an expectation, see Equation 4.1. The temporal-difference algorithm is able create an estimate of this function [Sutton and Barto, 2018].

$$V(s_t) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})] \tag{4.1}$$

where γ is the constant discounting factor.

Temporal-difference, TD, learning is a method that is both able to learn from experience without a model of the MDP and uses bootstrap. Bootstrap refers to being able to update estimates based on other learned estimates without waiting for a final outcome [Sutton and Barto, 2018]. Thus using temporal-difference, value functions can be updated for each successive action and reward instead of waiting until the simulation has finished. There are many variants of Temporal-difference learning, but the one relevant to the purpose of this thesis is the TD(0) variant. For each step TD(0) updates the estimate of the value function, $V(s_t)$, according to the update in Equation 4.2 [Sutton and Barto, 2018].

$$V(s_t) \longleftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{4.2}$$

where α is a constant learning factor, γ is the constant discounting factor for the value function estimate of the new state.

The target for the TD update is the reward and discounted value of the next state, $r_{t+1} + \gamma V(s_{t+1})$. α corresponds to the degree that the TD target updates the

new estimate. If α is 1, then the target replaces the estimate, $V(s_t)$. If α is between 0 and 1, then the new estimate is a combination of the TD target and the old estimate.

4.2 Q-learning

The Q-learning algorithm is a model-free and off-policy temporal difference control RL algorithm [Sutton and Barto, 2018]. Model-free in this context means that the algorithm does not require a prior understanding of the model of the underlying MDP of the system. Off-policy means that the policy itself is not used when exploring the MDP, see Section 4.3.

The algorithm uses the state-action value function, the Q-function, which aims to capture the current and future reward when taking an action, a_t, from a state, s_t. By estimating and maximizing the Q-function, the algorithm may determine the best next action in each state in the MDP. The estimation of the Q-function can be achieved through temporal difference learning, TD(0). Thus to update the Q-function a tuple containing the state, action, (s_t, a_t), reward, r_t, and the next state, s_{t+1}, is required, see Equation 4.3 [Sewak, 2019].

$$Q_{(s_t,a_t)} = (1-\alpha)Q_{(s_t,a_t)} + \alpha(r_t + \gamma \max_{a_{t+1}} Q_{(s_{t+1},a_{t+1})}) \tag{4.3}$$

where α is the learning factor, γ is the discounting factor, and a_{t+1} is the next action that maximizes the current Q-function [Sewak, 2019].

In the regular Q-learning algorithm, the Q-function is usually expressed as a table with a size corresponding to the number of states as rows and number of actions as columns. This Q-table can be zero initialized. Thus $Q_{(s_t,a_t)}$ corresponds to an element in the table and can be updated according to Equation 4.3. Since the agent estimates the Q-function it requires many iterations of the process in order to converge. The iterations required for the agent to sufficiently estimate the Q-function is refereed to as the training process [Sewak, 2019].

4.3 Exploration versus Exploitation

When considering the mechanism of choosing an action, i.e., the policy, we must consider the explore versus exploit dilemma. Exploit means using the knowledge that the agent has accumulated to choose the best action. In the case of Q-learning this corresponds to choosing an action through $a = \max_a Q(s,a)$. Explore means collecting new information for the agent thus making it possible for the agent to improve. For Q-learning this is done off-policy which means that the policy for exploration can be disconnected and separate to the RL algorithm as opposed to the SARSA algorithm, State-Action-Reward-State-Action, which is another RL algorithm. For Q-learning, the explore versus exploit problem can be handled by an algorithm such as the epsilon-greedy algorithm [Sewak, 2019].

The epsilon greedy algorithm handles the explorations phase by using a constant probability, ε, of choosing a random action in each step. The chance of choosing greedily, by exploiting, is then $1-\varepsilon$. The choice of ε is meant to be based on how deterministic the underlying MDP is, generally more stochastic MDPs require larger ε [Sewak, 2019].

4.4 Deep Q-learning

The deep Q-learning network algorithm, DQN, is a version of Q-learning where the Q-function is approximated with a neural network, a Q-network. Usually this is applied when the action and state spaces are large. The most common applications involve images as inputs, and thus the definition of DQN uses convolutional neural networks. However, since the state space in this system does not consist of images this is replaced with a fully connected feed-forward network.

Updates in the DQN are handled in a similar way as in Q-learning but use back-propagation. Two Q-networks are used, one for estimating the current $Q(s_t, a_t)$ value, the online network, and one for estimating the future target $\max_{a_t} Q(s_t, a_t)$, the target network, see Equation 4.3. The target network is not updated at every step, but instead copies the online network every, c steps. This is done because of issues with convergence in case of too frequent correlated data. Furthermore, using the same function to update itself might cause it to become unstable [Sewak, 2019].

Neural Networks

A neural net consist of layers which takes an input and produce an output. Each layer have a number of neurons or units, which is interconnected with the next layers neurons with weights. One neural net may have an input layer, then several hidden layers and finally an output layer. Each layer also have an activation function which determines the output of the neurons. A loss function quantifies how good the neural net is performing and is used when updating the network weights. When training the neural net, it is this loss function that is minimized [Gad and Jarmouni, 2021].

Experience Replay Buffer

If concurrent updates of the agent have high correlation, it might slow down or hinder convergence. Therefore, a replay experience buffer is used in order to reduce the correlation between updates. Instead of updating the Q-values instantly for a tuple (s_t, a_t, r_t, s_{t+1}), the tuple is stored in an experience replay buffer implemented as a ring buffer. Then a batch of random tuples from the ring buffer is sampled and used to update the neural net similar to Equation 4.3.

Double Q-learning

In instances where the state space and state-size are very large, it may take a long time for the agent to gain sufficient information of the system. This may cause the

Q-values to be over-estimated leading to sub-optimal training. In order to solve these issues in each update, the action can be determined from the online network but the values are used from the target network.

However, due to lack of time this modification was not implemented. For eventual future investigations into DQN it is highly recommended to use Double Q-learning.

5

Results

Different systems are used in order to evaluate the scheduling algorithms, see Table 5.1. For simplicity, a pattern of 10 UL symbols followed by 10 DL symbols are generated from each carrier in the systems. In all tested systems 40 BACs are used and the carrier bandwidth is 100 MHz. Four schedulers are applied on each system, the baseline, Q-learning based scheduler, DQN based scheduler and a random scheduler. The random scheduler picks random actions in the action space of the reinforcement learning based schedulers. The purpose of the random scheduler is to both illustrate the difficulty scheduling the system as well as act as a point of reference for the other schedulers. Furthermore, a constant $\varepsilon = 0.1$ has been used for exploration in the RL schedulers during the training process. Then during the simulations used to compare the schedulers, it is set to $\varepsilon = 0$, i.e. no exploration takes place.

Table 5.1 Carrier setup for the systems.

Name	# Carriers	# Antennas	# Layers	DD	numerology
System 1	1	64	16	TDD	1
System 2	5	64	16	TDD	0

The RL schedulers have been trained on the system before being evaluated. The Q-learning based schedulers have been trained on 200000 simulations. Since the DQN based schedulers take longer to train per simulation, they have only been trained on 20000 simulations. The DQN experience replay buffers have been initialized with values from the baseline scheduler in order to try to improve the scheduler.

5.1 I/O Design

In order to use RL algorithms the system must be converted in a way that it can be seen as a Markov decision process, MDP. Thus, a state and action space must be defined for the scheduler.

State Space

The state space for the algorithm is part of the overall design and has been chosen to try to capture the system properties. This choice is non-trivial, since an excess of states causes the training process to take too long and too few states might not be enough for the algorithm to make an optimal choice. In the current algorithm, the state consists of: job type, symbol index and the total number of SJ tokens in the resource pool, i.e. jobs in the queue or worked on by the BACs. Since the static scheduler determines an action per symbol input, this should be reflected in the state space. The reasoning behind the job type state property is that it determines the latency definitions and should thus be known to the scheduler. The symbol index aims to confer a sense of time to the algorithm. The total number of jobs in the resource pool gives information on how busy the resource pool is when the jobs are scheduled.

Action Space

The action space corresponds to the decisions the scheduler makes. The current action space consists of the SJ size in PRB, the priority in the queue, and the wait-time before sending the job to the resource pool. Similarly to the state space, this is also a design problem as there are many options for how much freedom the scheduler has in determining an action. For example, the job queue supports more priorities than three, however, as there are only three different types of jobs, the choice has been made to restrict it to three.

The priority will not matter as much when there is not much traffic since the BAC will pick and complete all the jobs in the queues before there is any competition of the BAC. The priority will be important when there are both DL and UL jobs in the job queues. This is because the latency definitions encourage that DL jobs should be completed late and UL jobs should be completed early. This means that if UL jobs have higher priority when DL jobs are sent to the queue there is a larger risk that the DL jobs will miss their deadlines. Perhaps it would be beneficial to restrict the scheduler to assign priority only in the shift between UL and DL, or perhaps to assign priority in a static way instead. This might make it easier for the scheduler to find an optimal policy.

Reward Function

The goal of the reward function is to reward desired results. The desired result in the system is to decrease the latency as well as to ensure that deadline requirements are met. Equation 5.1 has been designed in order to reflect this on a symbol basis.

$$r(x) = \begin{cases} e^{-k_{DL} \cdot x} & \text{for DL} \\ e^{-k_{UL} \cdot x} & \text{for UL} \\ -1 & \text{if deadline is missed} \end{cases} \tag{5.1}$$

where x is the maximal latency of SJ from the same symbol and k_{DL} and k_{UL} are positive constants.

Low latency is rewarded with higher rewards, whereas deadline misses are penalized with a -1 penalty. Also, two different exponents, k_{DL} and k_{UL}, were used because the latency is defined differently for UL and DL, see Figure 2.3. The reward function also restricts the reward between 1 and -1. This is important because if the total reward can grow too large from the reward from one action, it might be optimal to miss deadlines to maximize the reward, which is not a desired behaviour.

In the simulation the reward is determined by collecting all the result tokens corresponding to a symbol token and then calculating the latency for the last result token. Then the reward function is applied to get the reward corresponding to this symbol token.

Simulation Feedback

Note that the scheduler does not receive the reward and the next state instantaneously when determining an action. It must wait until the corresponding jobs are finished until a reward can be determined and for the next symbol to arrive in order to get the next state. Thus the scheduler stores the state until it can pair it with a reward and the next state and update the estimate of the state-action value function, i.e., the Q-function.

5.2 RL Implementation Details

Q-Learning

The Q-learning algorithm is implemented with a matrix spanning the action and the state space. The parameter values used in the two systems are: a learning factor of $\alpha = 0.5$ and a discount factor of $\gamma = 0.99$.

DQN

The DQN is implemented through the Keras application programming interface and the machine learning platform, Tensorflow, see [Chollet et al., 2015] and [Martín Abadi et al., 2015]. The neural net is implemented with an input layer of 64 units, with the states as inputs, 4 hidden layers of 256 units each and an output layer with units corresponding the the total number of actions. All layers except the output use the rectified linear unit activation function and the output layer use a linear activation function. The states used as inputs are translated into numbers and restricted between 0 and 1. The Adam optimizer is used to compile the Keras model and the the Huber loss is set as the loss function. Further information regarding the activation function, optimizer and the loss function can be found in the Keras documentation [Chollet et al., 2015].

The parameters used in the simulations were $\gamma = 0.99$ and the learning rate were set to a low number to try to increase stability, $\alpha = 0.0001$. The experience replay

buffer used a batch size of 64, i.e. each update used 64 experience tuples from the experience replay buffer. Finally, the target network is updated every $c = 10000$ updates.

5.3 System 1

The number of missed deadlines and latency for each scheduler and job type can be seen in Table 5.2. It is evident that the Q-learning based scheduler achieved the lowest overall latency, with better latency for BTA but worse for ATB1 and ATB2 compared to the baseline. The DQN based scheduler suffers from deadline misses and poor latency. It even performs worse than the random scheduler for BTA jobs.

Table 5.2 Number of deadline misses, total latency and latency per job type using different schedulers in system 1.

Scheduler	Deadline misses	Total latency	ATB1	ATB2	BTA
Baseline	0	1001421	121050	251801	628570
Q-learning	0	997167	198825	263491	534851
DQN	26	1580366	341384	572875	666107
Random	217	1957882	553700	872269	531913

Q-learning

It can be difficult to determine if the reinforcement learning algorithm improves during the training process. The average rewards during the training process are too erratic to analyze because of exploration and random actions that could lead to poor rewards. However, the average estimated Q-values of the state-action pairs per simulation seems more stable and an increase in Q-values corresponds to the scheduler choosing better actions over time. The average Q-values for each simulation are plotted in Figure 5.1. The Q-values seems to increase first rapidly, and then slower as it becomes harder to get better rewards to increase the estimate of the state-action values. The plot still varies heavily between high and low average Q-values but a positive trend can be observed. The variation is most likely because of the exploration, that the scheduler does not always pick the highest Q-value when determining an action during the training process. Since there is a positive trend it is possible that running the algorithm further would increase the average Q-values and likely find a better policy. However, as the training process takes a long time, there was not enough time to properly attempt this and analyze those results.

The final simulation with the Q-learning based scheduler is presented in Figure 5.2 and Figure 5.3. Figure 5.2 shows the accumulative rewards of the scheduler in the final simulation for each action of the scheduler. Interestingly, the scheduler's total reward does not exceed that of the baseline even though the overall latency is lower, see Table 5.2. Thus the scheduler did not beat the baseline in terms of the

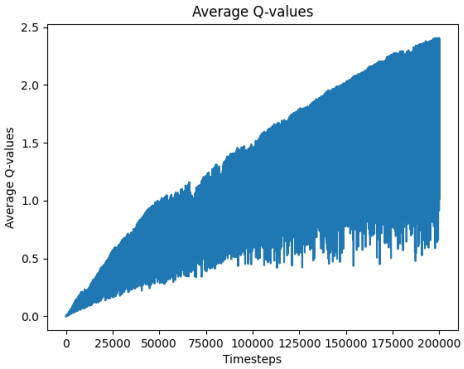

Figure 5.1 Average Q-values over the training process. Each time step corresponds to a simulation.

goal of a reinforcement learning algorithm i.e. maximizing the total reward. This is most likely because the reward function, see Equation 5.1 is not linear in terms of latency. One improvement to the system could be to design a linear reward function in order to better align the reinforcement learning goal to the general goal of the scheduler.

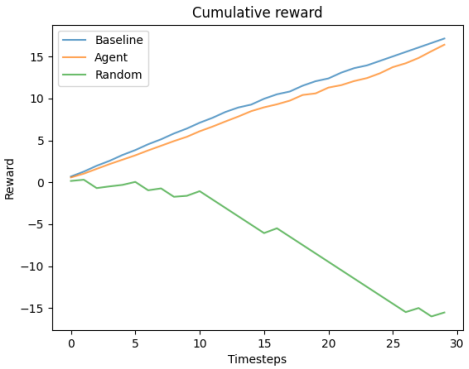

Figure 5.2 The cumulative rewards for the Q-learning based scheduler compared to baseline and the random scheduler.

The distributions of jobs on the BACs on the y-axis, with time on the x-axis, can be seen in Figure 5.3. Each box represent one job being completed by a BAC and the vertical lines represent the arrival of symbols to the system. The optimal choice for the scheduler when there are only uplink jobs is to reduce the latency by

sending the jobs to the BAC as soon as possible. This is not the case when viewing Figure 5.3 and Figure 5.6 as there seems to be a wait time for the ATB1 jobs in the first part of the simulation. This seems to indicate that the agent has not found the overall optimal solution. However, BTA jobs are completed later than the baseline in the end of the simulation. According to the latency definitions for uplink, see section 2.2, this corresponds to lower latency. This is reflected in Table 5.2 where the latency for the the Q-learning agent is higher for ATB1 jobs than the baseline, but lower for BTA jobs.

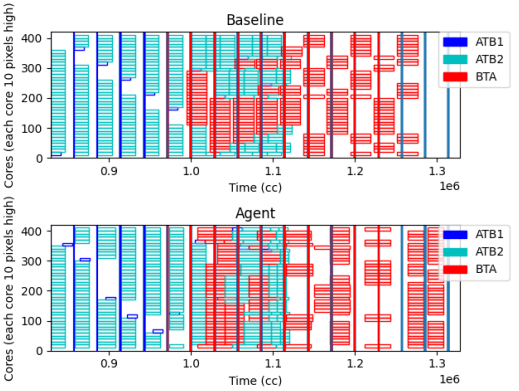

Figure 5.3 An overview of the jobs distributed over the BAC. The y-axis represents the BAC cores, one per 10 units. The vertical lines represent when symbols arrive to the system, colored red for DL and blue for UL.

The plots in Figure 5.4, Figure 5.5 and Figure 5.6 are scatter plots of the choices that the scheduler makes. The PRB size plot is somewhat misleading as the ATB1 job type is static and set to 16. Thus it seems that the PRB size for the other job types are scattered in both larger and smaller sizes. The priorities do not seem to exhibit a discernible pattern. The wait times are lower for UL and higher for DL, this is the desired behaviour as the latency definitions encourage DL jobs to be completed as close to their deadline as possible. However, around the transition between DL and UL, it seems that DL have generally lower wait times, perhaps in order to reduce the chance of missing a deadline because of a busy BAC.

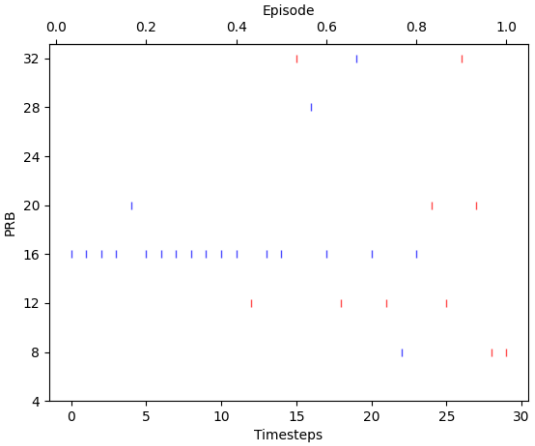

Figure 5.4 A scatter plot for the Q-learning scheduling agent choices in PRB sizes for each step. Blue corresponds to downlink and red corresponds to uplink.

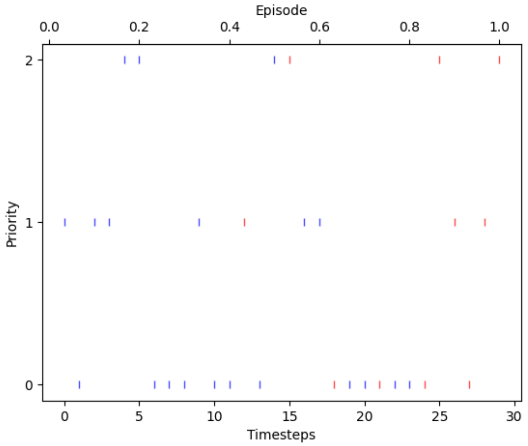

Figure 5.5 A scatter plot for the Q-learning scheduling agent choices in priority for each step. Blue corresponds to downlink and red corresponds to uplink. 0 corresponds to the highest priority and 2 corresponds to the lowest.

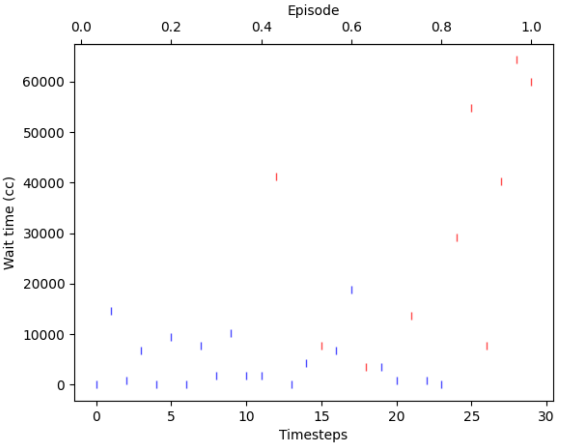

Figure 5.6 A scatter plot for the Q-learning scheduling agent choices in wait time for each step. Blue corresponds to downlink and red corresponds to uplink.

DQN

The average Q-values for the DQN agent, see Figure 5.7, seem to diverge. It seems that the scheduler overestimates the Q-values since the average grows far beyond the maximum total reward of 30. Since the state-action values represent an expected future reward of taking an action in a state, this is not reasonable. Overall, it seems the agent does not converge. This is supported by the loss function of the neural net, see Figure 5.8, that seem to grow instead of converge.

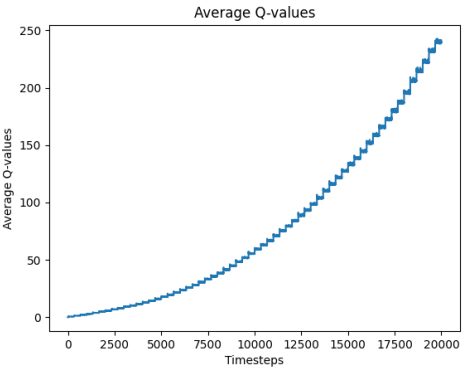

Figure 5.7 Average Q-values over the training process. Each timestep corresponds to a simulation.

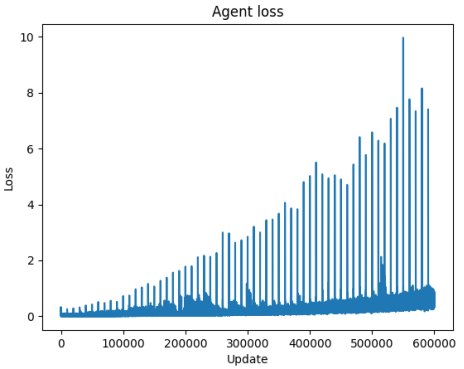

Figure 5.8 Loss in the neural net during the training process.

The cumulative rewards can be seen in Figure 5.9 and while the agent performs better than the random scheduler, it is worse than the baseline in terms of latency

and total reward, see Table 5.2.

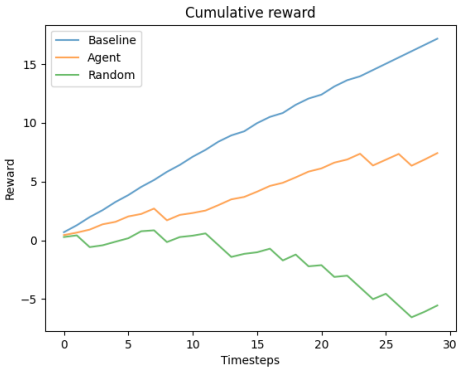

Figure 5.9 The cumulative rewards for the DQN based scheduler compared to baseline and the random scheduler.

From the job type and deadline plots in Figure 5.10 and Figure 5.11 it can be observed that the algorithm fails in areas of only uplink jobs or only downlink jobs. Furthermore, it schedules the uplink jobs much later compared to the baseline, thus increasing the latency. This is not surprising since the algorithm does not converge.

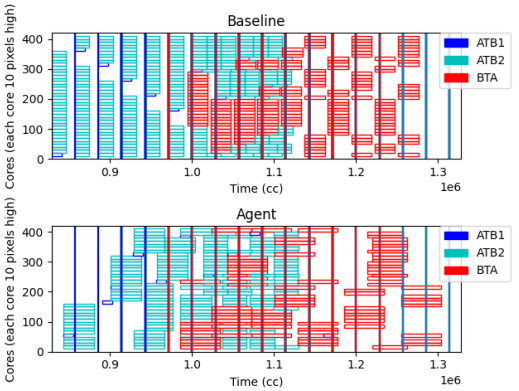

Figure 5.10 An overview of the jobs distributed over the BAC. The y-axis represent the BAC cores, one per 10 units. The vertical lines represent when symbols arrive to the system, colored red for DL and blue for UL.

The plots in Figure 5.12, Figure 5.13 and Figure 5.14 are scatter plots of the choices that the DQN scheduler makes. The scheduler seems to prefer higher PRB

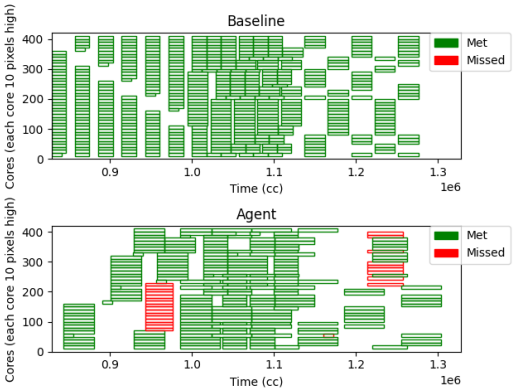

Figure 5.11 An overview of the jobs distributed over the BAC and if they met or missed their deadlines.

sizes, in general low wait times and assigns the same priority in many decisions.

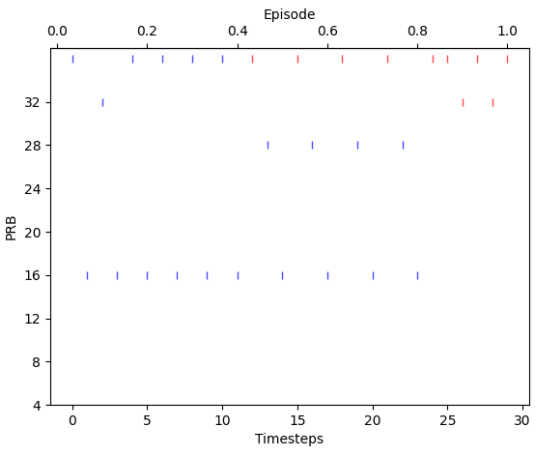

Figure 5.12 A scatter plot for the DQN scheduling agent choices in PRB sizes for each step. Blue corresponds to downlink and red corresponds to uplink.

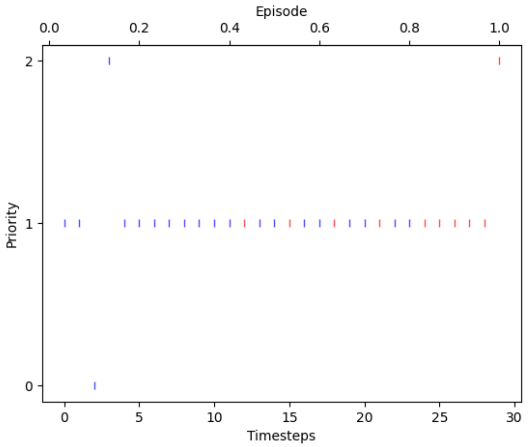

Figure 5.13 A scatter plot for the DQN scheduling agent choices in priority for each step. Blue corresponds to downlink and red corresponds to uplink. 0 corresponds to the highest priority and 2 corresponds to the lowest.

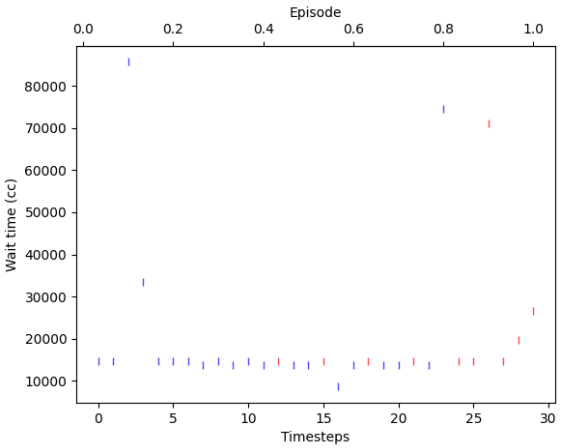

Figure 5.14 A scatter plot for the DQN scheduling agent choices in wait time for each step. Blue corresponds to downlink and red corresponds to uplink.

5.4 System 2

The number of missed deadlines and latency for each scheduler and job type can be seen in Table 5.3.

Table 5.3 Number of deadline misses, total latency and latency per job type using different schedulers in system 2.

Scheduler	Deadline misses	Total latency	ATB1	ATB2	BTA
Baseline	0	5426884	584520	1985264	2857100
Q-learning	0	10333449	2015532	2328240	5989677
DQN	185	17678841	6982557	2859370	7836914
Random	152	14573144	4857540	5357121	4358483

This system is more difficult for the reinforcement learning algorithms. This is because the five carriers each generate 20 symbols. These 20 symbols in turn is treated as 30 symbols in the simulation, because each of the 10 UL symbol is split into two symbols corresponding to the UL job types. Thus the scheduler must schedule 150 symbols in the simulated environment instead of 30 as the previous simulation. In the current implementation this increases the state space of the scheduler since the symbol index is used as a state. Thus it is not surprising that the DQN scheduler performs even worse on this system as seen in Table 5.3. Because of this only the performance of the regular Q-learning algorithm is shown in depth below.

Q-learning

The average Q-values for each simulation has been plotted in Figure 5.15.

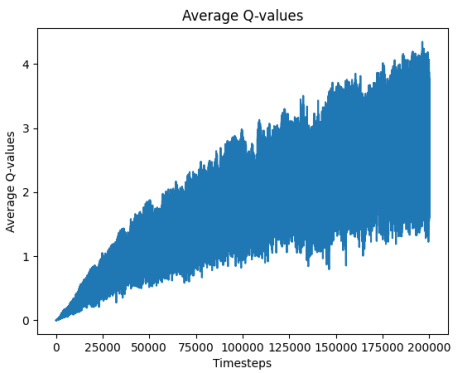

Figure 5.15 Average Q-values over the training process. Each timestep corresponds to a simulation.

The cumulative reward for the baseline, random and Q-learning based scheduler can be seen in Figure 5.16.

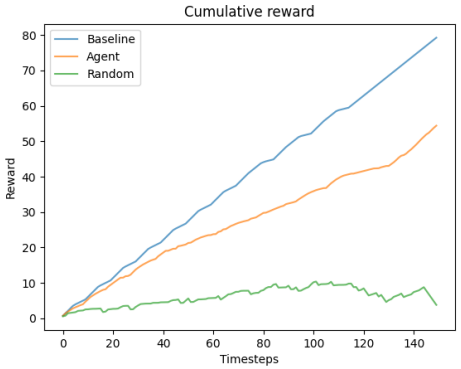

Figure 5.16 The cumulative rewards for the Q-learning based scheduler compared to baseline and the random scheduler.

The job distribution over the BAC can be viewed in Figure 5.17. It can be observed that the BACs are busier when the RL agent is used than the baseline.

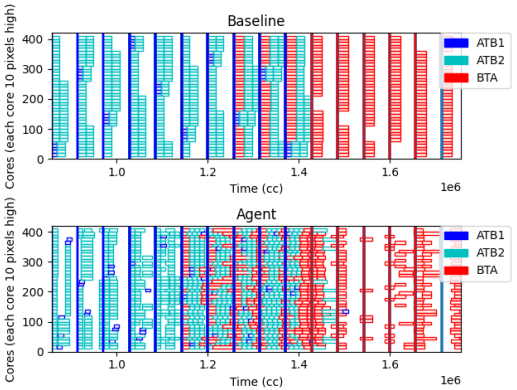

Figure 5.17 An overview of the jobs distributed over the BAC. The y-axis represent the BAC cores, one per 10 units. The vertical lines represent when symbols arrive to the system, colored red for DL and blue for UL.

The PRB size, priority and wait time actions of the scheduler can be viewed in Figure 5.18, Figure 5.19 and Figure 5.20. Interestingly, it seems that a low PRB size of 8 is a common choice which makes the BAC busy, as can be seen in Figure 5.17.

The priority seems to mostly be 0, which corresponds to the highest priority in the simulation. However, it seems that most jobs are set to the same priority in the area where downlink and uplink symbols meet. This seems to indicate that perhaps the scheduler does not use the priority to achieve lower latency. Perhaps it does not contribute enough to the reward or perhaps the scheduler has not explored enough of the action space. For wait times, it seems that the final part of the simulation with only downlink symbols are scheduled with higher wait times. The rest of the symbols are mostly scheduled with low wait times.

Making the BACs busier while not missing deadlines has the potential to decrease latency. This divides the work on more BACs and increases throughput. However, it seems that the Q-learning scheduler suffers from not completing ATB1 jobs immediately as can be seen in Table 5.3. Furthermore, it seems that BTA jobs also achieve poor latency. This can probably be explained by the wait time actions in Figure 5.6 which show that many DL symbols have been assigned with low wait times, which for DL means higher latency.

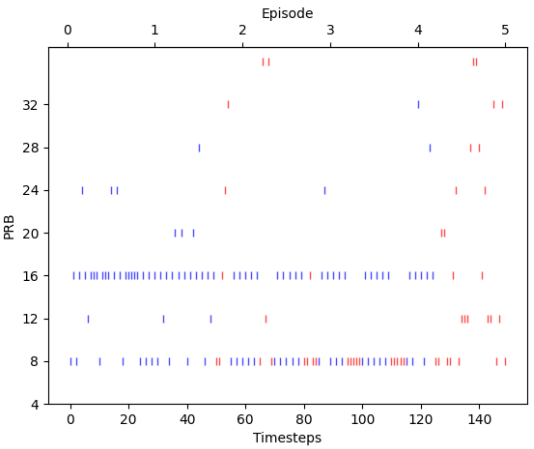

Figure 5.18 A scatter plot for the Q-learning scheduling agent choices in PRB sizes for each step. Blue corresponds to downlink and red corresponds to uplink.

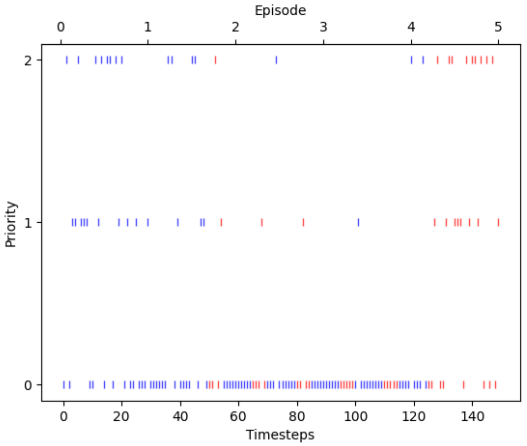

Figure 5.19 A scatter plot for the Q-learning scheduling agent choices in priority for each step. Blue corresponds to downlink and red corresponds to uplink. 0 corresponds to the highest priority and 2 corresponds to the lowest.

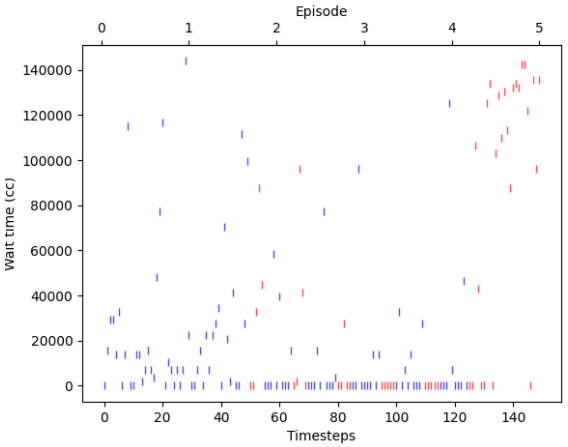

Figure 5.20 A scatter plot for the Q-learning scheduling agent choices in wait time for each step. Blue corresponds to downlink and red corresponds to uplink.

6

Discussion

6.1 Simulation Design

Some parts of the simulation could be extended. While FDD has been implemented in the simulation, the choice was made to exclude the configurations that use it from the results. This is because configurations that use TDD were tested and examined to a higher degree than FDD. Thus the simulation results could not be trusted when using FDD. Future tests with FDD when the simulation has been tested further could provide interesting results.

The simulation generates one ATB1 symbol token and one ATB2 symbol token from one uplink symbol. One change to the general design that would simplify the problem would be to not schedule ATB1 and ATB2 separately. The job types take different amount of time to complete, however, they share the latency definitions. As can be seen in Table 5.2 ATB1 contributes to the latency increase. Furthermore, in Figure 5.3 and Figure 5.6 it can be observed that the scheduler actually schedules ATB1 jobs with a wait time larger than zero even when BACs are available. Thus, in order to make the system easier to control it might be beneficial to abstract the job types to UL versus DL and let the scheduling between job types in UL be handled in a static way.

6.2 Scheduler Design

The design of the state space requires some tweaks in order to be applied to the real system. The real system is a continuous task where the number of symbols increases indefinitely. Thus the state property of symbol index is not suitable as it would increase infinitely. One possible way to do this would be to replace the symbol index with a property that captures progression in time but does not increase indefinitely, perhaps using subframe and slot number in a radioframe as part of the state space, see Section 2.1.

As discussed previously the action space is part of the design and care will have to be taken in how much freedom the schedulers are allowed. For example, it might

not be reasonable to include priority as part of what the scheduler is allowed to decide. As discussed before in Section 5.1, it mostly matters when there are both DL and UL jobs in the job queue. Perhaps the priority could be assigned so that DL jobs are prioritized higher than UL jobs in order to avoid DL jobs missing their deadlines. Similar arguments could be made to further restrict the action space of the scheduler. For example, some possible wait times will never be optimal as they will always result in missed deadlines. On the other hand, these actions also make it easier to check if the scheduler actually converges since it is guaranteed that some actions will result in poor performance. Furthermore, the scheduler is meant to be able to handle different configurations and thus it can be difficult to determine that some combination of action should be allowed or not.

It is clear from the results that the DQN-based scheduler performed poorly. However, it should be noted that this does not necessarily mean that it is unreasonable to investigate this algorithm further for the purpose of scheduling. The regular Q-learning scheduler has been easier to implement and there is a possibility that the DQN scheduler performs poorly because of human error. If this is the case, it it probable that there is a problem with the design of the neural net that approximates the Q-function or the parameters used for the simulations. Furthermore, the DQN has been able to handle complex systems, for example being able to play Atari games [Volodymyr Mnih and Silver, 2015]. One important thing to consider here is that DQN is usually used in a system where the states consist of pictures and the Q-function is estimated through underlying convolutional neural nets. Thus the DQN has potential in the fact that it is better suited to handle large state spaces which might become a problem for the regular Q-learning scheduler when developed further.

This thesis has used Q-learning based schedulers in trying to schedule in an optimal way. However, other reinforcement learning algorithms might be better suited to this task. For example the current Q-learning schedulers are model-free and do not require any information about the underlying Markov decision process. But it is known when symbols will arrive to the scheduler and in some configurations it is also known beforehand if these symbols are uplink or downlink. Perhaps if this information of the underlying Markov decision process could be incorporated in a model-based scheduler it could beat the Q-learning based schedulers.

7

Conclusion

A simulation of the scheduling environment has been implemented and several scheduling algorithms have been implemented. While there was not enough time to examine FDD, most of the important system dynamics were simulated. Two reinforcement learning based schedulers were implemented and tested on the system, and one of them provided promising results.

The results shows that the Q-learning based scheduler were able to beat the baseline in terms of total latency for system 1. Furthermore, even if the scheduler was unable to beat the baseline for system 2 it still passed all deadlines. The DQN based scheduler, however, faced issues regarding convergence in system 1 and 2 which led to missed deadlines and poor latency. As discussed previously, there are some additions and revisions that could be made to the scheduler. The primary of these is the ability to handle the continuous task of the real system. This requires some modification of the state space of the scheduler. Another interesting prospect is to investigate to what degree the scheduler should be limited in the action space in order to ensure better performance. Furthermore, some modifications to how the symbols are handled by the overall simulation is suggested.

In conclusion, it is shown that the Q-learning based scheduler is able to handle scheduling on the EMCA for two different configurations in terms of passing deadlines on the simulated system. For one of these, the simulated total latency was also reduced. While these results provide validation for reinforcement learning-based schedulers, there are also additional steps required until it could be applied on the real system.

Bibliography

Chinchali, S., P. Hu, T. Chu, M. Sharma, Bansal, M., Misra, R., M. Pavone, and S. Katti (2018). "Cellular network traffic scheduling with deep reinforcement learning". DOI: https://doi.org/10.1016/j.engappai.2004.08.018..

Chollet, F. et al. (2015). *Keras*. https://keras.io.

DeepMind. *Alphago*. URL: https://deepmind.com/research/case-studies/alphago-the-story-so-far.

Ergen, M. (2009). *Mobile Broadband*. Springer US. ISBN: 9780387681924.

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

Gad, A. F. and F. E. Jarmouni (2021). *Introduction to Deep Learning and Neural Networks with Python*. Academic Press. ISBN: 9780323909334.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, et al. (2015). *TensorFlow: large-scale machine learning on heterogeneous systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Scherfke, S. and O. Lünsdorf (2020). *Simpy, discrete event simulation for python*. URL: https://simpy.readthedocs.io/en/latest/index.html (visited on 2021-04-08).

Sewak, M. (2019). *Deep reinforcement learning*. Springer.

Sutton, R. and A. Barto (2018). *Reinforcement Learning: An Introduction*. Second edition. MIT Press. ISBN: 9780262352703.

Volodymyr Mnih, K. K. and D. Silver (2015). "Human-level control through deep reinforcement learning". *Nature*. DOI: https://doi.org/10.1038/nature14236.

von Butovitsch, P., D. Astely, A. Furuskär, B. Göransson, B. Hogan, J. Karlsson, and E. Larsson. *Advanced antenna systems for 5g networks*. URL: https://www.ericsson.com/en/reports-and-papers/white-papers/advanced-antenna-systems-for-5g-networks.

Wang, Y.-C. and J. M. Usher (2005). “Application of reinforcement learning for agent-based production scheduling”. DOI: https://doi.org/10.1016/j.engappai.2004.08.018..

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

Document name
MASTER'S THESIS

Date of issue
June 2021

Document Number
TFRT-6143

Author(s)
Patrik Trulsson

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

Title and subtitle
Dynamic Scheduling of Shared Resources using Reinforcement Learning

Abstract

The goal of the thesis is to simulate the Ericsson Many-Core Architecture, EMCA, and implement a dynamic scheduler for the system using reinforcement learning methods. The system contains shared resources that receive and complete jobs. Also, the deadlines and latency definitions can change depending on the job type. The scheduler should aim to avoid missing deadlines as well as aim to reduce the overall latency in the system. A python simulation has been implemented of the Ericsson Many-Core Architecture and two reinforcement learning based schedulers have then been developed and used for different configurations. They are evaluated by comparing their performance to a random and a static scheduler. The first scheduler uses Q-learning and the second uses a version of Q-learning with a neural net that approximates the Q-function. The results showed that the second version experienced issues with convergence which caused deadline misses and poor latency. The regular version of Q-learning showed promising results, avoiding deadline misses and was able to reduce the latency below that of the static scheduler for one of the systems. There are still some issues that could be addressed as well as avenues to explore regarding the scheduler. Furthermore, in order to apply the scheduler to the real system, some modifications are necessary. However, the simulations show that the reinforcement learning can successfully be used as a scheduler on the EMCA for different configurations.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-46

Recipient's notes

Security classification

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