# Contents

# Acronyms

**AI** Artificial Intelligence. 6–8, 10, 17, 26, 28

**DNN** Deep Neural Network. 10, 25

**DQN** deep Q-learning. 12, 13, 25

**JSS** Job Shop Scheduling. 18, 19, 22, 25, 26, 28

**KPI** Key Performance Indicator. 9

**ML** Machine Learning. 7–9, 17

**MRLCO** Meta Reinforcement Learning Computation Offloading Solution. 22–24, 27

**PPO** Proximal Policy Optimization. 10

**QL** Q-learning. 13

**RL** Reinforcement Learning. 6–12, 14, 16, 17, 19, 22, 25, 27, 28

# Chapter 1

# Introduction

The cloud is everywhere. All companies in the tech big 5, i.e. Facebook, Apple, Amazon, Netflix and Google, use the cloud extensively and offer many cloud services to their users. Movies are not bought anymore, but streamed. New MacBooks do not come with much storage, instead buyers get some storage in the Apple cloud. But how are the movies and pictures stored in the cloud? What even is the cloud? In this thesis it is shown that the cloud is a network of many computers. These computers need to be managed, which is done by schedulers. Efficient schedulers is very useful, because this means that using cloud services becomes more efficient. In this thesis we look at how methods for improving this efficiency can be selected.

In this chapter the many aspects of resource schedulers in cloud environments are discussed. Firstly, in Section 1.1.1 cloud environments are discussed. Secondly some theoretical background about the problem of job scheduling and its NP-hardness is provided in Section 1.1.2. Lastly, in the Section 1.1.3 the scope of the presented thesis is defined, the current situation is evaluated, the gap is identified and a research question is stated.

## 1.1 Background

### 1.1.1 The Cloud

The cloud is a popular term used for (a network of) data centers containing many computers that provide resources, for example storage and computing power. These computers communicate via standard network protocols locally. Data centers thus contain many computers that provide computing resources. The four major advantages of using data centers for large computing tasks are the following: 1) Distributed. Data centers are the ideal environment for running distributed

software, because the nature of data centers is distributed computing; 2) Robustness. When one computer breaks, the task can be sent to another computer and the broken computer can be replaced while the application is still running; 3) Configurable. The computer can be selected based on the resource-need of the task. Providing many different configurations can make sure that tasks fully utilize all the resources the computer provides; 4) Scalability. Data centers are modular and thus easier to scale. New computers can be added and connected to the network and directly be used.

An additional benefit of the cloud is how companies pay for the resources they need. This way, companies never pay for idle resources and can easily scale up to use more resources when needed. By building and sharing huge data centers, companies can effectively achieve the economies of scale principle for needed resources. This reduces the cost of the resources, but also the maintenance costs. The reason cost is reduced because resources are bought in bulk and maintenance costs are reduced because there are less maintainers needed.

### 1.1.2 Job Scheduling

Imagine cloud systems as described in Section 1.1.1. These systems have many tasks, or as in this thesis called jobs, that need to be executed for it to function. Jobs are scheduled by a job scheduler or simply called the scheduler. These jobs vary a lot in duration time and resource-need. To illustrate, if the cloud environment provides a back-end for a website, a job could be one of the following: Serving the right HTML page to a visiting user, compressing user uploaded images, spam filtering, detecting anomalies in user logins or many other types of jobs that need to be executed to provide a fully functioning website. The schedulers distributes jobs over many resources, provided by a cloud environment. Optimally distributing the jobs is important for cloud environments, because efficiently using resources means there are less resources needed and jobs will complete in a shorter time. Thus, well working schedulers are important for cloud environments, because efficiently using resources saves money and is easier to maintain.

What is efficient and how can schedulers be efficient? There are many metrics on which optimization can be done to make a scheduler more efficient. For example minimizing job slowdown time, minimizing average completion time, maximizing throughput (jobs per time unit) or minimizing total completion time (the makespan). Schedulers are created to be optimized on one or more of these aspects, varying the importance. Unfortunately, the optimization of schedulers is complex. It is a well-known problem in computer science, because of its NP-hardness. In the next paragraph is explained what it means to be a NP-hard problem and the reason that job scheduling is NP-hard.

#### NP-Hard

NP-hardness is a term used in the P versus NP problem. This problem is one of the seven Millennium Prize Problems (Cook, 2006) and still unsolved. The P versus NP problem is about computational complexity, an approach to categorizing problems based on 'how hard' they are. Computational problems have two complexity aspects: the complexity to solve the problem and the complexity to verify if the solution to the problem is correct. The question is if the solution to a given problem can be verified quickly (in polynomial time), is there an algorithm that can find the solution quickly? Problems that can be *solved* in polynomial time are in the P class. Problems that can be *verified* in polynomial time are in the NP (nondeterministic polynomial time) class. If these classes fully overlap, i.e. every problem in P is also in NP and every problem in NP is also in P, then P equals NP. Many computer scientists believe this is not the case for all problems (Rosenberger, 2012). Believed is that there are problems that can be verified quickly but not solved quickly, which are called NP-hard problems. A well-known NP-hard problem is the Sudoku puzzle, especially larger ones (Yato, 2003).

Job scheduling is a generalized version of the traveling salesperson problem. The problem is as follows: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" (Flood, 1956). In this problem, the cities are the resources and the salesman is the job.

Job scheduling is NP-hard because it can be derived from the Graph Coloring Problem, as done in Karp (1972). The graph coloring problem is a NP-hard problem itself.

### 1.1.3 Current Situation and Improvements

In this section current traditional (non-AI) schedulers are reviewed and improvements using Artificial Intelligence (AI) techniques like Reinforcement Learning (RL) are evaluated. The problem with current AI improvements of schedulers are identified and the aim of this resource is stated. The scope of this thesis is narrowed down to reducing retraining time of RL based schedulers in cloud environments. The choices made in defining the terms and scopes of this thesis are listed in the following paragraph.

#### Traditional Schedulers

Current cloud resource managers are developed specifically for the system it manages, based on simple heuristics and fine tuned by trial and error. Creating the resource managers is a hard and tedious task. A common aspect of the resource

managers based on simple heuristics is the straightforwardness. Current cloud schedulers are developed for ease of understanding. The schedulers generalize, i.e. they perform the same job regardless of whether the workload is heavy or light (Mao, Schwarzkopf, Venkatakrishnan, Meng, & Alizadeh, 2019). Three classic (non-AI ) algorithms are explained below to provide an idea of the simplicity of the non-AI schedulers.

1. First in, first out (FIFO): This algorithm treats the awaiting jobs like a queue and lets later jobs wait until earlier jobs finish and resources are available for the next job.

2. Shortest Job First (SJF): This algorithm sorts awaiting jobs based on increasing order of completion time.

3. Tetris: ?

The above explained algorithms are highly intuitive resource managers and not fine tuned for different workloads. Due to the lack in flexibility of these non-AI algorithms , there are also situations in which they will perform much worse than other algorithms will do. For FIFO this leads to multiple jobs with a long duration time, blocking all other jobs till finish. The disadvantage of SJF is that it can cause starvation, meaning that short jobs are constantly added and will never be executed.

Currently, schedulers are not capable of handling differences in workload and have other shortcomings. There is a need for schedulers that are capable of handling increasingly complex large-scale systems, although the systems are currently already too complex for humans to fully understand and schedule. Due to the shortcomings of current non-AI implementations, previous research is done on implementing these resource managers using AI technology. Recent research, e.g. Mao et al. (2019); Mao, Alizadeh, Menache, and Kandula (2016); Zhang et al. (2020), has shown that using deep reinforcement learning for resource management improves average job completion time by at least 21% (Mao et al., 2019).

## 1.2 Motivation

In the previous section the problem with non-AI schedulers is identified. Thereafter, improvements using RL are shown, but these RL improvements also have their shortcomings. One of these shortcomings is flexibility. A known problem of Machine Learning (ML) algorithms is overfitting on the training environment and not being able to work with environmental changes. This is also the case with RL based schedulers. Schedulers can show undefined behavior when resources

are added or removed. Thus, when a change in resources is done, the RL based schedulers need to be retrained to work in the new environment. Retraining is a common problem with ML algorithms. Retraining takes time, in which a sub-optimally working scheduler is still scheduling. Retraining also costs computing resources itself. This costs money and contributes to global warming. Carbon emission of large ML models is a real issue which is currently researched on, e.g. Patterson et al. (2021). Lastly, having enough data available to cover all perturbations is not feasible. Finding ways to reduce retraining time of RL models in cloud environments is important, because many schedulers can benefit from using these techniques. This makes cloud environments more efficient which is better for the company in terms of costs, efficiency and maintenance and better for the environment. It also contributes to more robust RL schedulers. If RL schedulers are more robust, more cloud companies will switch to using RL based schedulers in stead of the current non-AI schedulers.

The goal of this research is to integrate robustness into current state-of-the-art ML based resource managers. This leads to the following research question: How to select a method for reducing retraining time of reinforcement learning based resource schedulers in cloud environments? This will be answered by the following sub-questions:

1. How can we effectively assess and compare RL based methods in job scheduling?

2. What are the indicators to assess robustness of RL based schedulers?

3. What are the state-of-the-art approaches for reducing retraining time?

The goal of this research is to provide a method for selecting methods that reduce retraining time. By reducing retraining time the models become more flexible and

Firstly the state-of-the-art RL schedulers and state-of-the-art methods for reducing retraining time for RL algorithms are reviewed and explained. In the following chapter, the methodology of selecting methods is described. Thereafter the results of this research are shown. Lastly, a conclusion of the results is formed and discussed.

# Chapter 2

# State of the Art

In this chapter related work is discussed. In Section 2.1 the scope of the thesis is defined and the methodology of finding related work is described. Thereafter related work on the topics of RL based schedulers, assessing robustness of RL methods and approaches for reducing retraining time is discussed. Lastly, the gap of in the related work is analyzed.

## 2.1 Scope and Methodology

The scope of this thesis is specifically RL based schedulers in cloud environments. In Section 2.2.1 is explained why this thesis only focusesses on RL based schedulers, not schedulers based on other forms of ML .

Comparing different methods and models is the main task in this thesis. To compare methods of models, a justified methodd of assessing methods and models is required. It is also important that the comparison is done on the indicators on which the performance depends. These are called the Key Performance Indicators (KPIs) . The KPIs of RL based schedulers are established from literature. Assessing the robustness of a model is also necessary for this research. By assessing robustness, a way to select a method for reducing retraining time can be selected.

## 2.2 Related Work

### 2.2.1 Reinforcement Learning

Most ML based schedulers are RL based, and therefore this thesis focuses on reducing retraining of RL models. The reason most ML based schedulers are RL based is because of how RL works and differs from the other ML paradigms. RL

and its differences with the other paradigms are very well explained in 'the most popular artificial intelligence textbook in the world'[1], Russell and Norvig (2010). Their explanation of RL is summarized in the following paragraph.

### Why Reinforcement Learning for Schedulers?

Reinforcement learning is one of the three basic paradigms in machine learning, along with supervised learning and unsupervised learning. RL is different from the other two paradigms. Supervised learning and unsupervised learning have a thing in common: the need for data. Supervised learning needs annotated data, various inputs and desired outputs are given and the algorithm learns a function to get as close to the wanted outputs as possible given the inputs. With unsupervised learning a model is forced to build an internal representation of the world by mimicking the data. The reason RL is different is that it does not depend on data, but rather learns from a feedback loop of rewards or reinforcements. It typically consists of one or more agents, a set of actions $A$ and a set of states $S$. This agent performs an action $a \in A$ to a perform state transition. This action leads to a reward. In many complex environments RL is the only feasible method for training models because there might be little data available or the environment is too complex to model (Russell & Norvig, 2010). For the reason that RL has no need for input data but solely needs an environment, actions and a reward function it is widely used in research about using AI in resource managers. The popularity of RL and the nature of how RL works is the reason this research is narrowed down to reducing retraining of RL schedulers.

### Reinforcement Learning Based Schedulers

Many reinforcement learning algorithms are used in state-of-the-art RL based schedulers. Three different state-of-the-art RL schedulers are described in this section.

The first state-of-the-art RL based scheduler is DeepRM, presented in Mao et al. (2016). This scheduler is based on deep reinforcement learning with policy representation via a Deep Neural Network (DNN) . The algorithm learns by performing gradient-descent on the policy parameters using the REINFORCE algorithm from Sutton, McAllester, Singh, Mansour, et al. (1999). For the state representation a 2-D image is used to capture the status of resources and jobs. The second state-of-the-art scheduler is proposed in Zhang et al. (2020) and also trains a policy network, but in this algorithm the network is trained using Proximal Policy Optimization (PPO) , an actor-critic algorithm. Unlike DeepRM, the

[1]According to this blog, but also shown on the homepage of the book (http://aima.cs.berkeley.edu/).

model from Zhang et al. (2020) is not hard bounded by the instance size (Zhang et al., 2020, p. 5). Because it is not hard bounded by the instance size it is more flexible and robust.

### 2.2.2 Evaluation of Reinforcement Learning Models

Currently there is not yet a standard evaluation method for RL models. Due to the absence of a standard evaluation model, reproducibility often becomes more difficult. Work is done in defining standard evaluation methods, but these methods are not yet the industry standard. Three metrics were found. The first evaluation is regret. Regret can only be used as a metric when an agent with optimal policy can be defined. The difference in actions taken is the regret per action. This is defined as the reward for the action of the optimal agent minus the reward for the taken action. Knowing the optimal policy is not always possible. The second evaluation metric is not used literature but rather in practice. Some non-scientific metrics that were used to evaluate RL models are popular, well-defined environments like the OpenAI gym (Brockman et al., 2016). A comparison of RL models is done on their leaderboard[2]. The used metric is the number of episodes it took to solve the problem. Lastly, proposed methods in literature are for example a framework for evaluating RL in Khetarpal, Ahmed, Cianflone, Islam, and Pineau (2018) and a novel evaluation method in Jordan, Chandak, Cohen, Zhang, and Thomas (2020).

### 2.2.3 Assessing Robustness of Reinforcement Learning

If a RL model is robust, it has the ability to cope with dynamic environments and works well in noisy environments. The goal of this thesis is to compare methods for reducing the retraining time of RL based schedulers. If a model is robust, it can handle the dynamics of the environment and will most likely not need much retraining time. Thus, making the RL scheduler more robust will most likely result in less retraining time. This is why related work on robust reinforcement learning algorithms is discussed. The comparison between robustness of reinforcement learning models is only as descriptive as the metric used for it. Selecting a well-formed method for assessing robustness is important for this research.

Many methods for assessing robustness of reinforcement learning use some kind of "disturbance" in the training environment (Morimoto & Doya, 2005). By changing dynamic aspects of the environment, the model learns about the dynamics and accounts for this.

In Al-Nima, Han, Al-Sumaidaee, Chen, and Woo (2021) the neuron coverage, i.e. the ratio of the activated neurons in a network, is used a measurement of

[2] github.com/openai/gym/wiki/Leaderboard

robustness. This is only applicable in RL methods with neural networks in it, for example deep Q-learning (DQN) .

### 2.2.4 Approaches to Reduce Retraining Time

Reducing retraining time of reinforcement learning models is an important task. By doing this, reinforcement learning models get more robust and can be used in more flexible environments. This allows using reinforcement learning as solution for problems which were too flexible for previous RL models. Reducing retraining time also has a cost efficient motivation; less retraining time means less resources-heavy training with high energy costs. In this section the current state-of-the-art approaches are discussed. The approaches are categorized in three categories: (1) approaches modelling the dynamic aspect of the environment, (2) approaches using adversarial agents and (3) approaches that reduce retraining by using earlier learned policies. The reviewed approaches to reduce retraining time are discussed on category basis in the following subsections.

#### Modelling the Dynamic Environment

Modelling the dynamics of the environment is a straightforward approach to taking account of the dynamics. For example, in job schedulers in cloud environments, this can be done by adding and removing resources during training. It would theoretically be possible for a model to learn that a change in resources is possible and take that in account. Numerous approaches include information about the dynamic aspect of the environment into the model. To successfully apply RL models in dynamic environments, Wiering (2001) proposed to include an a-priori model of the changing parts in the environment. In Nagayoshi, Murao, and Tamaki (2013), a method was proposed for detecting environmental changes on which the model can adapt. The aforementioned proposed methods require a model of the environment or a-priori knowledge of the changing parts of the environment. In many environments, this would not be feasible, but it could be possible in job scheduling since the only changing element of the environment is the number of resources. The disadvantage of modelling the dynamic environment beforehand is that it might not be possible due to the randomness of the changes in resources. The outage of a resource can probably be predicted when some information on the current state of the resource is provided, but predicting the outage of a resource is out of the scope of this thesis.

### Adversarial Agents

Another approach for learning how to handle disturbance is by adding in a disturbing agent (Morimoto & Doya, 2005). This agent learns to perform the most disturbing action as possible. The same approach is also proposed in Pinto, Davidson, Sukthankar, and Gupta (2017), therein called robust adversarial reinforcement learning. In Pinto et al. (2017) it is also stated that the gap between simulation and real world is too large for policy-learning approaches to work well. This might be achievable if the adversarial agent can control when specific resources inserted or deleted. This raises a number of questions. How would the amount of operations (insertion and deletions of resources) be selected? This has to be controlled, otherwise the adversarial agent would score best by deleting all resources. Will the agent work better than training with random resource operations? Are there critical moments where the deletion of resources is worse than other moments?

Implementing an adversarial agent can be done with Q-learning models. Q-learning (QL) is a model typically used in single agent environments and is of simple nature. Having two Q-learning agents in an environment, one learning the main task and the other one being the adversarial agent can improve robustness. This technique have been used in a simple two player game (Littman, 1994). There are also other approaches for reducing retraining time of QL models in specific. These approaches include repeated update Q-learning (Abdallah & Kaisers, 2016) and variants of DQN like robust DQN (Chen et al., 2018).

Another approach is to train robustness online, i.e. during deployment of the model (Fischer, Mirman, Stalder, & Vechev, 2019). This approach, called Robust Student-DQN (RS-DQN ) deploys adversarial agents online to trick the model into unwanted states. This approach maintains competitive performance.

### Reusing Knowledge

Meta reinforcement learning is a method for for learning to quickly adapt online to new tasks. This is done in the context of model-based reinforcement learning. Meta reinforcement learning models use meta-learning to train a dynamics model that can be rapidly adapted to local context (Nagabandi et al., 2019). Meta learning is described as "Learning to learn". A model trains to learn during meta-training. Here, the learner learns the new task and the meta-learner trains the learner. In (Schweighofer & Doya, 2003) it is proposed to learn meta-parameters with stochastic gradients. It is a robust algorithm that finds appropriate meta-parameters and controls the parameters in a dynamic, adaptive manner. Even when retraining is needed to adapt to the changes in environment, the parameters of the model will probably be nearly the same as before.

## 2.3 Gap Analysis

As shown in this chapter, much work is done in combining job scheduling and reinforcement learning. There is also research done in reducing the retraining time of reinforcement learning models and making these models more robust. In this section the gap is identified and analyzed.

In Section 1 multiple RL based schedulers were discussed. These schedulers lack robustness: flexible environments are not handled well. As shown in Section 2.2.4 methods for improving robustness of RL models exist, but are not yet integrated in RL based schedulers. Integrating these methods would improve robustness and reduce retraining time.

There also is not a standard collection of indicators to assess robustness of RL based schedulers. A standard collection of robustness indicators simplifies comparing RL based schedulers and method for reducing retraining time of of the schedulers.

# Chapter 3

# System Design

# Chapter 4

# Implementation

In this chapter the implementation of the system is described. In this thesis approaches to reduce retraining time of RL based schedulers are compared. The approaches are compared against multiple RL based schedulers. This is done to see how well the approaches work on different RL algorithms. Firstly the software and hardware for the experiments is described (§4.1). Secondly the data and its usage is described in Section 4.2.

## 4.1 Hardware and Software Specification

This section explains the hardware and software setup and configuration used for the research. First the hardware used is described and after this some important software and its usage is described. The experiments were run on workstation 8 provided by the Robotics Lab at the University of Amsterdam. Workstation 8 is an Alienware Aurora R9 1.0.4. The CPU is an Intel i9-9900 (16 threads) running at 5.000 GHz and the GPU is a NVidia 2080 Ti (11 GB) with CUDA 10 and driver 415. The operating system is Ubuntu 18.04.4. All algorithms were implemented in the Python programming language[1]. To prevent versioning issues between dependencies, a virtual Python environment was created for every algorithm. This also allowed us to run a different Python version per algorithm. The tool used for creating and managing virtual Python environments is called `pyenv`[2]. An example of setting up a virtual environment:

```
$ pyenv install 3.7.10
$ pyenv virtualenv 3.7.10 meta-rl
$ pyenv activate
```

[1] python.org
[2] github.com/pyenv/pyenv

The above commands install Python 3.7.10, create a virtual environment called 'meta-rl' that uses Python version 3.7.10 and activates this virtual environment. From then on, packages could be installed using the Python package installer `pip` and can only be used in the 'meta-rl' environment. Standard practice is to include a `requirements.txt` file in the root directory of your project. However, only some of the projects used in this thesis included it. Installing the needed packages at the right version is as simple as running the command `pip install -r requirements.txt`.

Some packages are almost always used when writing AI software in Python. These packages are NumPy, Scipy, Keras, Tensorflow or PyTorch[3]. NumPy and SciPy provide an API for scientific computing and most importantly a solid multidimensional array object. The other three provide a back-end for creating well optimized ML algorithms, especially for neural networks. Another software project used extensively with this thesis is the OpenAI Gym (Brockman et al., 2016). Gym is a for developing and comparing reinforcement learning algorithms. It mostly provides access to many simple environments to develop your RL algorithm on, or as their slogan says "We provide the environment; you provide the algorithm."[4].

## 4.2 Data and Preprocessing

Four datasets were provided by the supervisors at the start of the thesis. The data is log data from the Euro-Argo project[5]. The datasets described transfered messages in intervals of one or ten minutes. The four datasets provided 43,200, 182,283, 4,320 and 28,312 rows of data, which sums up to 258,118 rows of data in total. The datasets all had five columns: the start time, the transfer time, request number, byte count and completion status. An overview of the datasets is provided in Table 4.1.

**Table 4.1:** Description of the dataset `mts_june_10m_rh.csv`. The description is somewhat applicable to all four datasets.

| | Start Time | Transfer Time | Request ID | Bytes | Completed |
|---|---|---|---|---|---|
| Date Type | date | integer | integer | integer | boolean |
| Min | - | 145 | - | 8164772 | - |
| Max | - | 299061 | - | 165483000000 | - |
| Mean | - | 2451 | - | 986461130 | - |

[4] gym.openai.com
[4] Their respective home pages: numpy.org, scipy.org, keras.io, tensorflow.org and pytorch.org.
[5] euro-argo.eu/

The data was complete, i.e. it had zero missing values. Not having missing values does not mean the data is clean. The dataset contained 3 rows where the transfer time, request id and byte count had the value of 0. The 3 rows were dropped because it lead to problems during the conversion of the data into another format. Another quirk to take note of is that even though the column name 'request ID' would suggest its values will be IDs and thus unique, this is not the case. For example, there are 1,576 unique IDs in the 4,320 rows of of the `mts_june_10m.csv` file. All counts of unique IDs and rows are shown in Table 4.2. The commands for getting this data can be found in Appendix A.

**Table 4.2:** Overview of the four dataset files. The columns 'Rows' and 'Unique IDs' contain the row and ID count, the 'Completed' column lists the percentage of completed requests (where the completed column equals 1).

| File | Rows | Unique IDs | Completed |
|---|---|---|---|
| `mts_june_10m.csv` | 4320 | 1576 | 44% |
| `mts_june_10m_rh.csv` | 28311 | 1016 | 87% |
| `mts_june_1m.csv` | 43200 | 655 | 79% |
| `mts_june_1m_rh.csv` | 182283 | 620 | 94% |
| Total | 258118 | 3867 | 90% |

### 4.2.1 Converting to other data formats

The Euro-Argo dataset was not compatible with most models. This is first of all because it is not a Job Shop Scheduling (JSS) dataset. The data are incoming requests from many Internet of Things-like devices. The data is sequential and are not different jobs that need to be scheduled. It is possible to convert the original data to formats that are more used in JSS tasks. In the following two sections, two conversions are discussed.

#### Directed Acyclic Graph

The first format in which the data was converted was a JSS . Following the example of the provided data by the model that uses JSS data[6], the Euro-Argo data was converted to a JSS using DAGGEN[7]. DAGGEN is a command line tool that generates a JSS usable by scheduling algorithms. It has a number of parameters; `n` ($n$) is the number of nodes, `mindata` ($D_{min}$) is the minimum size of processed

[7] github.com/frs69wq/daggen

data, `maxdata` ($D_{max}$) is the maximum size of processed data, `density` ($d$) determines the numbers of dependencies between tasks, `ccr` ($ccr$) the communication to computation ratio and `fat` ($fat$) the width of the JSS , i.e. the maximum number of tasks that can be executed concurrently. Because the Euro-Argo data was originally not JSS data but log data of requests, the converted data used parameters and other aspects from the example data. In the example data, all $n$ values were in the range of 1 to 50. The $n$ values were determined by our byte count value, which was scaled to be in $[1, 50]$. Ten outliers were removed to get the mean $n$ closer to the example mean $n$. The $D_{min}$ and $D_{max}$ were generated from the transfer time values. The other three parameters, $d$, $ccr$, and $fat$, were randomly chosen. The distribution of these random values followed the distribution of the parameter values found in the example data. For $d$ this was an uniform distribution $[4, 10]$, for $ccr$ it was an uniform distribution in $[3, 6]$ and for $fat$ it was an uniform distribution in $[4, 9]$. There was one other parameter, the regularity, which was always 0.5 in the example graphs. An example of a generated DAGGEN command is the following: `daggen --dot -n 4 --mindata 2656 --maxdata 2658 --density 0.4 --ccr 0.5 --fat 0.6 --regular 0.5 -o graphs/4-2657.gv`. The generated graph file from this command is shown in Figure 4.1. It can be seen in Figure 4.1 that the tasks on the lines without an arrow (`->`) contain an attribute `expect_size`. This can not be added with the DAGGEN tool and is probably added by the authors of the model that uses this data. We noticed that the expected size is 2.00, 2.50, or 3.33 times smaller than the size attribute of that task. The distribution was almost uniform: 16500 times a factor of 2, 16820 times a factor of 2.50, and 16680 times a factor of 3.33. The expected size was thereafter integrated using the factors as mentioned above.

### Taillard Specification

Other models worked with the popular OpenAI Gym environment. The Gym environment allows people to build their own environment to make it possible to test RL algorithms in other kinds of environments. Gym itself mostly provides game-like environments. The interface for Gym is very minimal, as typical RL algorithms have simple interaction with the environment. Most importantly, it exposes the following methods and attributes: `env.reset()` which resets the environment and returns the start state, `env.observation_space` which resembles the current state, `env.action_space` which are all possible actions in the current state, and `env.step(action)` which performs an action and returns the new state, a reward for the action and if the model is done.

[7] The model called metaRL-offloading uses DAG data. The example data can be found here: github.com/linkpark/metarl-offloading.

**Figure 4.1:** A relatively small graph generated with DAGGEN. The graphs varied from 1 line to 381 lines. Each line represents a task. The task have the size, alpha and expect_size attributes.

```
// DAG automatically generated by daggen at Thu Jun 10 17:00 2021
// daggen --dot -n 4 --mindata 2656 --maxdata 2658 --density 0.4
//        --ccr 0.5 --fat 0.6 --regular 0.5 -o graphs/4-2657.gv
digraph G {
  1 [size="8589934592", alpha="0.12", expect_size="3435973836"]
  1 -> 3 [size ="33554432"]
  1 -> 4 [size ="33554432"]
  2 [size="42466422292", alpha="0.15", expect_size="16986568916"]
  3 [size="31452222268", alpha="0.15", expect_size="9530976444"]
  4 [size="8589934592", alpha="0.10", expect_size="4294967296"]
}
```

In Tassel, Gebser, and Schekotihin (2021) a gym environment called `JSSEnv` for simulating the job shop scheduling problem is proposed. This environment provides the bridge between instances specified in the Taillard specification and the Gym interface. The Taillard specification, proposed in Taillard (1993), is a specification for computing instances. It specifies the number of machines $\mathcal{M}$ and jobs $\mathcal{J}$ and processing time per job for all machines. An example for an instance with one job and three machines. The machines are numbered from 0 to 2 and the processing time of the job is respectively 5, 8 and 9. The visiting order is 1, 2, 0. The representation of this instance is shown in Figure 4.2.

```
1 3
1 8 2 9 0 5
```

**Figure 4.2:** A computing instance following the Taillard specification.

As can be seen, first the number of jobs and the number of machines and then one job per line, with the visiting order and their processing time. To apply the Euro-Argo data to this format, we first analyzed the existing instances. In the instances provided by Tassel et al. (2021) there were either 40 or 45 machines, evenly distributed. The distribution of the number of jobs is shown in Figure 4.3. Our data was sequential, as is every line in the Taillard specification. One line describes the path one job travels between machines. To convert the Euro-Argo data into we only used the transfer time. This was converted in the processing time in the Taillard specification. Then the data was divided in groups of size $\mathcal{M}$,

then popped $\mathcal{J}$ items from these groups, shuffled the machine IDs ($id \in [0, \mathcal{M})$) and zipped the shuffled machine IDs and the transfer times together into one line for the Taillard instance.

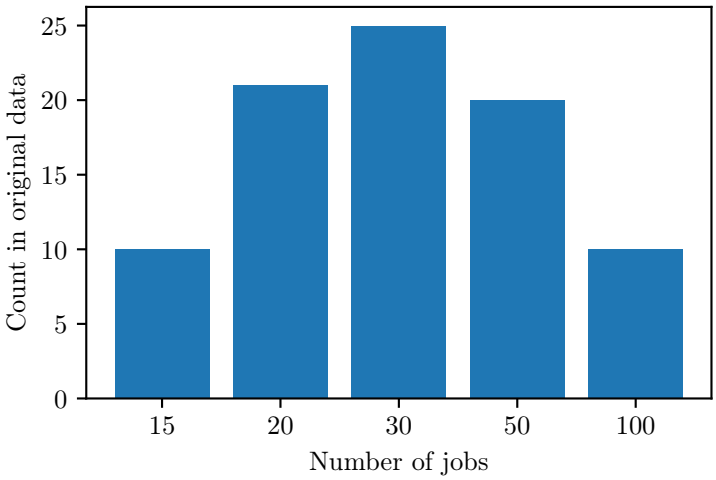


**Figure 4.3:** Distribution of number of jobs in data from Tassel et al. (2021).

# Chapter 5

# Experiments

In this section the performed experiments are described and results of the experiments are shown. The experiments were performed on the workstation described in Section 4.1. All experiments were programmed in Python and open-source software projects for the models and environment were used extensively.

## 5.1 Meta Reinforcement Learning

The objective of the first experiment was to evaluate the adaptiveness and reduce in retraining time with meta RL . This is done using the meta RL offloading model, called Meta Reinforcement Learning Computation Offloading Solution (MRLCO) , proposed in J. Wang, Hu, Min, Zomaya, and Georgalas (2020). This model required the data to be a JSS in DOT format. For this, the Euro-Argo log data was converted into these JSS files. This process is described in Section 4.2. Running the experiments with the newly converted data required modifications in the source code of MRLCO . In the `meta_trainer.py` file there was a list of directories containing 100 graph data files. This was changed into the directories with our data. It was also possible to change hyperparameters of the model in this source file, but this was not done in this experiment.

**Table 5.1:** The average policy and environment execution time per run. The values are in seconds.

| | Policy | Environment |
|---|---|---|
| MRLCO Data | 2.477595 | 0.953273 |
| Provided Data #1 | 2.485432 | 0.953415 |
| Provided Data #2 | 2.506485 | 0.910931 |

The model was ran three times: first with their own data, then with the first

part of the converted Euro-Argo data and thereafter with the second part of the converted Euro-Argo data. Each run consisted of 2,000 iterations in batch sizes of 100. After each batch a TensorFlow checkpoint was saved on disk. Every iteration it appended a row of data to a csv (comma separated value) file. The following aspects of each iteration were saved:

- Iteration
- Number of trajectories
- Average reward
- Execution time:
  - Policy execution time
  - Environment execution time
- Return value:
  - Average return value
  - Minimal return value
  - Maximal return value
  - Standard deviation of return value
  - Average discounted return
- Latency:
  - Average latency
  - Average greedy latency

The first run used 19 directories of 100 graph files. The reward in the last iteration of training was $-5.42$. The average return value at the last iteration was $-5.60$. For the second run the model was trained on 15 directories of 100 graph files, thus 1,500 graphs in total. The average reward and average return in the last iteration where respectively $-6.74$ and $-6.89$. The last run, on the other half of the Euro-Argo data, was also performed on 1,500 graphs. The average reward and average return were $-6.97$ and $-6.93$ respectively. The maximal, minimal and average return value per iteration are shown in Figure 5.1 and the progress of the average reward during training is shown in Figure 5.2. The model also logged the execution time of the policy and environment. The average execution times per run is listed in Table 5.1. There is little difference in the execution times. The largest difference in Policy execution time between MRLCO data and the second batch of the Euro-Argo data is 0.02889 and the largest difference in environment execution time is 0.042484.

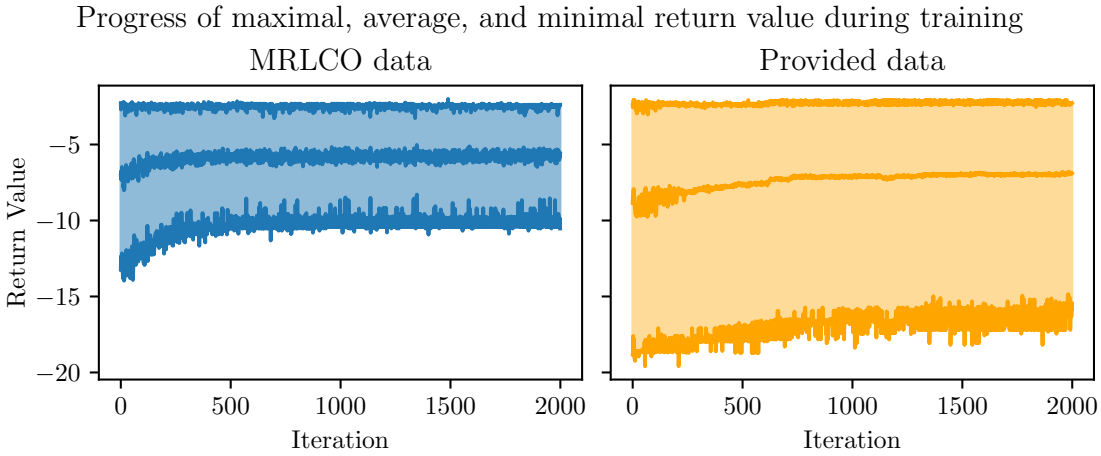


**Figure 5.1:** The progress of the return value during training. The upper line plots the maximal return value per iteration, the middle line the average return value and the lower line plots the minimal return value per iteration.

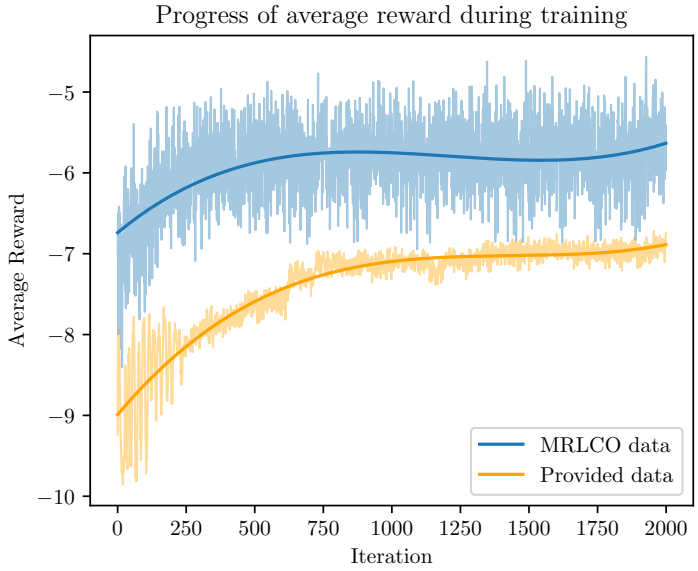


**Figure 5.2:** The progress of average reward during training on MRLCO and Euro-Argo log data. The average reward on the MRLCO data seems to variate much more than on our data. The MRLCO data delivers a higher reward.

## 5.2 OpenAI Gym

The second experiments were done to evaluate multiple RL algorithms as schedulers. A side objective of this experiment was to have a good testbed for future experiments. OpenAI's Gym was chosen for this because of its popularity and simplicity. As described in Section 2.2.2 and Section 4.1 OpenAI is a popular platform for creating, testing and evaluating reinforcement learning models. The environment used in this experiment is described in Section 4.2.1. The JSS environment (`JSSEnv`) in combination with the `keras-rl` package[1] would allow us to experiment with many different RL algorithms and evaluate their performance as scheduler. The `keras-rl` package currently supports deep Q-learning (DQN) (Mnih et al., 2013), Double DQN (Van Hasselt, Guez, & Silver, 2016), Deep Deterministic Policy Gradient (Lillicrap et al., 2015), Continuous DQN (Gu, Lillicrap, Sutskever, & Levine, 2016), Cross-Entropy Method (Szita & Lörincz, 2006), Dueling network DQN (Dueling DQN ) (Z. Wang et al., 2016), and Deep SARSA. Simulating the agent in the JSS environment required very little code. First make the Gym environment, than define a neural network (shown in Figure 5.3), define a RL agent, providing a policy and the neural network for it, and lastly fit it on the environment.

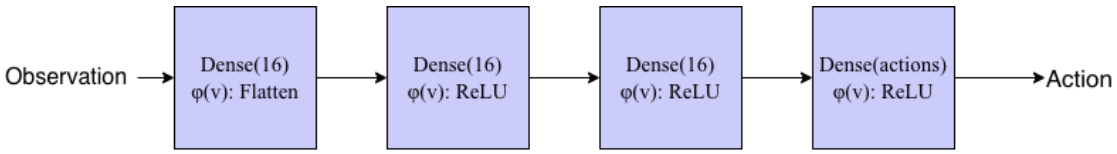


**Figure 5.3:** The DNNs used in the RL models from `keras-rl`.

Starting the experiment, we first ran a FIFO scheduler (First-In-First-Out, basically a queue) in the environment. The makespan after this run was 5724 and the final reward was -5.80 and the average reward was 0.40. After this general Q-learning and the deep RL algorithms mentioned above were tried. There were no results due to various errors, which will be discussed in detail in Section 6.

[1] github.com/keras-rl/keras-rl

# Chapter 6

# Discussion

## 6.1 Achievements

During this research project there were many larger and smaller achievements. We discuss three achievements in this section.

Firstly, right at the start of the project, the two first chapters (§1 and §2) were written. Because the BSc. AI does not require much academic writing, writing was not one of the best skills. However, during writing, this developed. Continuing with the other chapters after the experiments was easier and costed a lot less time. One of my achievements therefor is the improvement in English academic writing.

Secondly, during the experiments, a lot if projects with very specific Python dependencies had to be run on the same machine. Virtual environments were not necessary for us to be used before this project. After the experiments a better understanding of Python's virtual environments and the knowledge of the advantages of containerizing these environments has developed.

Lastly, during the whole project, much is learned about the fields of cloud computing and JSS . Before the project there was almost no knowledge about this fields. The only aspect of this thesis with some foreknowledge are the workings of reinforcement learning and python programming. Due to the lack of knowledge in cloud computing and scheduling much information had to be read to get somewhat of an understanding of it. It is a complex field working on very complex technology. It was very interesting to get some insights in these fields.

## 6.2 Shortcomings

In this section the shortcoming of this research are discussed. During the experiments it was not managed to fulfil all experiments to properly answer the research

question. The reasons for this are discussed in the following subsections.

### 6.2.1 Papers with Code

Nowadays, many papers include the corresponding code[1]. Although this is generally a good thing that should be encouraged, it is not always usable yet. All code repositories that accompanied related work, lacked documentation. The paper itself does not document how code works and additional documentation is not written either. Working with the codebase required reading the written code, that usually also lacks comments and good design patters, and figuring out how to adjust it to work with your own data, code, or environment. To give an example, the MRLCO code[2] only specifies the entry point of the code and which packages should be installed. The list of packages is incomplete and it is not specified that an old version of Python should be installed. Another example from the same code repository is that the command for generating a specific graph is commented in the first line of the file. But when running these commands, the jobs lack an `expect_size` attribute. This is not something that can be done with `DAGGEN` and must be later on added by the authors of the research paper. This should definitely be documented. Academic code is also error-prone with unclear error messages as result. The unclear error messages require lots of time to debug, which is not doable in a time-critic project like the thesis. To conclude, the shortcoming was the expectation that code provided with papers would be industry-grade code with clear documentation. This was not the case with most projects, which leads into time-consuming debug sessions. Because of this, I had less time for the actual experiments. Although papers that include code are a good step in the right direction, it is not very useful yet because of the low quality code and documentation. Due to lack of documentation it is necessary to read the code to be able to use it, which can cost much time.

### 6.2.2 Standards in Reinforcement Learning

At the start of the experiments it was clear that the papers on RL and scheduling algorithms lack standards in data format and API. We found no standard for this during literature review. This made comparing and assessing them much harder. It was not possible to loop through the models, fit them on the same data and get the same metrics from them. It also was not possible to reuse code written for one specific algorithm. Data had to be converted into data formats that differ very much from each other. From graph structure to a specification dating from

[1] paperswithcode.com
[2] github.com/linkpark/metarl-offloading

1993. This did not match the expectations. Many times in the BSc. AI , data would be presented in a *Pandas DataFrame* and algorithms would work on this tabular data. Looking back, the assumption that JSS data would also generally be in tabular form should not have been made. There should be a standard in the field of RL and JSS for data and code APIs to easily swap RL algorithms and train many different RL algorithms on the same data or environment. Good steps have been made by OpenAI with their Gym environment, but the JSS environment is not good yet.

### 6.2.3 Generating Data

A popular method for training AI schedulers is to generate instances. There are many advantages: generating random environments is easy and can still be tweaked by parameters. The data does not have to be saved on disk and there is no I/O overhead. Random environments are also a good exercise: there are likely no patterns in incoming jobs. Because of this, the AI will not just recognize patterns and change approach accordingly. But generating data has its disadvantages, e.g. not reproducible due to randomness, unclear how parameters might affect the generated data, not the right probability distribution in randomness, and it might differ to much from real world data. It is also hard to detect possible overfitting on certain data aspects and can introduce biases (Hall & Posner, 2001).

Another important problem with generated data is that it is hard to compare performance between two algorithms using different data generators. Even if the parameters are all equal, it still might have different random distributions, other logic or some magic numbers in it. The problem with different data input formats as discussed in Section 6.2.2 also exists when the data is generated.

### 6.2.4 Concluding Shortcomings

The results are not the expected results from the start of the research project. Most of the shortcomings that led to not achieving the intended results are due to the models that do not meet the expectations on standardization and clear documentation. Converting the data for every model that would be evaluated was not feasible in the amount of time for this project.

# Chapter 7

# Conclusion

# Chapter 8

# Future Work

# References


Abdallah, S., & Kaisers, M. (2016). Addressing Environment Non-Stationarity by Repeating Q-learning Updates. *Journal of Machine Learning Research*, *17*(46), 1–31. Retrieved from http://jmlr.org/papers/v17/14-037.html

Al-Nima, R. R. O., Han, T., Al-Sumaidaee, S. A. M., Chen, T., & Woo, W. L. (2021). Robustness and performance of Deep Reinforcement Learning. *Applied Soft Computing*, *105*, 107295. doi: https://doi.org/10.1016/j.asoc.2021.107295

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym.* Retrieved 2021-05-06, from https://arxiv.org/abs/1606.01540

Chen, S.-Y., Yu, Y., Da, Q., Tan, J., Huang, H.-K., & Tang, H.-H. (2018). Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *Proceedings of the 24th acm sigkdd international conference on knowledge discovery & data mining* (p. 1187–1196). New York, NY, USA: Association for Computing Machinery.

Cook, S. (2006). P versus NP Problem. In J. Carlson, A. Jaffe, & A. Wiles (Eds.), *The millennium prize problems* (pp. 87–106). Providence, RI: American Mathematical Society.

Fischer, M., Mirman, M., Stalder, S., & Vechev, M. T. (2019). Online Robustness Training for Deep Reinforcement Learning. *CoRR*, *abs/1911.00887*. Retrieved 2021-06-02, from http://arxiv.org/abs/1911.00887

Flood, M. M. (1956). The Traveling-Salesman Problem. *Operations research*, *4*(1), 61–75. Retrieved from https://www.jstor.org/stable/pdf/167517.pdf

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016). Continuous deep q-learning with model-based acceleration. In *International conference on machine learning* (pp. 2829–2838).

Hall, N. G., & Posner, M. E. (2001, November). Generating Experimental Data for Computational Testing with Machine Scheduling Applications. *Operations Research*, *49*(6), 854–865. Retrieved 2021-06-28, from https://www.jstor.org/stable/pdf/3088599.pdf

Jordan, S., Chandak, Y., Cohen, D., Zhang, M., & Thomas, P. (2020, July).

Evaluating the Performance of Reinforcement Learning Algorithms. In *International Conference on Machine Learning* (Vol. 119, pp. 4962–4973).

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85–103). Springer. doi: 10.1007/978-1-4684-2001-2_9

Khetarpal, K., Ahmed, Z., Cianflone, A., Islam, R., & Pineau, J. (2018). Re-evaluate: Reproducibility in evaluating reinforcement learning algorithms. In *International Conference on Machine Learning*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994* (pp. 157–163). Elsevier.

Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016, November). Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (pp. 50–56). Atlanta, GA: ACM. doi: 10.1145/3005745.3005750

Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., & Alizadeh, M. (2019, August). Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication* (pp. 270–288). Beijing, China: ACM. doi: 10.1145/3341302.3342080

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Morimoto, J., & Doya, K. (2005, 02). Robust Reinforcement Learning. *Neural Computation*, *17*(2), 335-359. Retrieved from https://doi.org/10.1162/0899766053011528 doi: 10.1162/0899766053011528

Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., & Finn, C. (2019, February). Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning. *arXiv:1803.11347 [cs, stat]*.

Nagayoshi, M., Murao, H., & Tamaki, H. (2013). Reinforcement learning for dynamic environment: a classification of dynamic environments and a detection method of environmental changes. *Artificial Life and Robotics*, *18*(1-2), 104–108.

Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., ... Dean, J. (2021). Carbon Emissions and Large Neural Network Training. *arXiv:2104.10350 [cs]*. Retrieved 2021-05-04, from https://arxiv.org/abs/2104.10350

Pinto, L., Davidson, J., Sukthankar, R., & Gupta, A. (2017, August). Robust

adversarial reinforcement learning. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (Vol. 70, pp. 2817–2826). PMLR.

Rosenberger, J. (2012, May). P vs. NP poll results. *Communications of the ACM*, *55*(5), 10. Retrieved 2021-05-04, from https://mags.acm.org/communications/201205?pg=12#pg12

Russell, S. J., & Norvig, P. (2010). *Artificial intelligence: a modern approach* (Third ed.). Upper Saddle River: Prentice Hall.

Schweighofer, N., & Doya, K. (2003). Meta-learning in reinforcement learning. *Neural Networks*, *16*(1), 5-9. doi: https://doi.org/10.1016/S0893-6080(02)00228-9

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Nips* (Vol. 99, pp. 1057–1063). Cambridge, MA, USA: MIT Press.

Szita, I., & Lörincz, A. (2006). Learning tetris using the noisy cross-entropy method. *Neural computation*, *18*(12), 2936–2941.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278–285. (Project Management and Scheduling) doi: https://doi.org/10.1016/0377-2217(93)90182-M

Tassel, P., Gebser, M., & Schekotihin, K. (2021). A reinforcement learning environment for job-shop scheduling. *arXiv:2104.03760 [cs.LG]*. Retrieved 2021-06-22, from https://github.com/prosysscience/JSSEnv

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 30).

Wang, J., Hu, J., Min, G., Zomaya, A. Y., & Georgalas, N. (2020). Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning. *IEEE Transactions on Parallel and Distributed Systems*, *32*(1), 242–253.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning* (pp. 1995–2003).

Wiering, M. A. (2001). Reinforcement learning in dynamic environments using instantiated information. In *Machine learning: Proceedings of the eighteenth international conference (icml2001)* (pp. 585–592).

Yato, T. (2003). *Complexity and completeness of finding another solution and its application to puzzles* (Master's thesis, University of Tokyo, Tokyo, Japan). Retrieved 2021-05-04, from http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/MasterThesis.pdf

Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P. S., & Chi, X. (2020). Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. In *Ad-*

*vances in Neural Information Processing Systems* (Vol. 33, pp. 1621–1632).

# Appendix A

# Dataset Analysis

These are the bash commands used for the data analysis in Table 4.2.

Number of rows:

```
$ sed 1d mts_june_10m.csv | wc -l
$ sed 1d mts_june_10m_rh.csv | wc -l
$ sed 1d mts_june_1m.csv | wc -l
$ sed 1d mts_june_1m_rh.csv | wc -l
```

Number of unique request IDs:

```
$ sed 1d mts_june_10m.csv | cut -d',' -f 3 | sort -h | uniq | wc -l
$ sed 1d mts_june_10m_rh.csv | cut -d',' -f 4 | sort -h | uniq | wc -l
$ sed 1d mts_june_1m.csv | cut -d',' -f 3 | sort -h | uniq | wc -l
$ sed 1d mts_june_1m_rh.csv | cut -d',' -f 4 | sort -h | uniq | wc -l
```

Percentage completed (and after this $(N_1/(N_1 + N_0) \cdot 10)$:

```
$ sed 1d mts_june_10m.csv | cut -d',' -f 5 | sort | uniq -c
$ sed 1d mts_june_10m_rh.csv | cut -d',' -f 7 | sort | uniq -c
$ sed 1d mts_june_1m.csv | cut -d',' -f 5 | sort | uniq -c
$ sed 1d mts_june_1m_rh.csv | cut -d',' -f 7 | sort | uniq -c
```