Comparing Methods for Reducing Retraining Time in Job Scheduling

Yochem van Rosmalen

Layout: typeset by the author using LaTeX.
Cover illustration: C. Dustin on unsplash.com.

Comparing Methods for Reducing Retraining Time in Job Scheduling

Yochem van Rosmalen
12223247

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
H. Liu MSc. and Dr. Z. Zhao

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

July 2, 2021

Abstract

todo

Contents

Acronyms

AI Artificial Intelligence. 3, 4

DNN Deep Neural Network. 8, 24

DQN deep Q-learning. 10, 12, 24

JSS Job Shop Scheduling. 5, 13, 16, 17, 20, 24, 26, 31

KPI Key Performance Indicator. 7, 13

ML Machine Learning. 4, 7, 15

MRLCO Meta Reinforcement Learning Computation Offloading Solution. 12, 20, 22, 23, 25, 26, 28, 30

NN Neural Network. 15, 24

PPO Proximal Policy Optimization. 8

QL Q-learning. 11

RL Reinforcement Learning. 4, 5, 7–15, 18, 20, 24, 27, 28, 31

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 rather streamed. New MacBooks do not come with much storage. Instead, each purchase includes some storage in Apple's cloud. How are the movies and pictures stored in the cloud? What even is the cloud? The cloud is a network of many interconnected computers, the nodes in the network, providing resources. Oftentimes these nodes are abstracted as just resources. The utilisation of these resources need to be managed, which is done by schedulers. Efficient schedulers are essential, because they directly influence the cloud platform by having a high resource utilization rate and end-users by having a good quality of service. This thesis investigates methods for improving the robustness of schedulers.

1.1 Background

1.1.1 The Cloud

The cloud is a popular term used for (a network of) data centres containing many computers that provide resources, for example storage and computational power. These computers communicate via standard network protocols locally. Data centres thus contain many computers that provide computational resources. The four significant advantages of using data centres for large computational tasks are the following: 1) Distributed. Data centres are the ideal environment for running distributed software, because the structure of data centres offers much distributed scalability; 2) Robustness. When a component breaks, the scheduler could migrate the task to other available resources, and the broken component can be replaced while the application is still running; 3) Configurable. The resources can be se-

lected based on computation requirements of the task. Providing many different configurations improve the utilisation rate and efficiency of cloud platforms; 4) Scalability. Data centres offer virtualised resources and are thus easier to scale. Availability of resources can easily be scaled.

An additional benefit of the cloud is how companies pay for the resources, i.e. pay-as-you-go. Pay-as-you-go means that companies never pay for idle resources and can easily scale up to use more resources when needed. By building and sharing large data centres, companies can effectively achieve the economies of scale principle for needed resources. Economy of scales reduces the cost of the resources, but also the maintenance costs.

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 to scheduler. These jobs vary a lot in duration time and required resources. To illustrate, given that 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 functioning website. The scheduler allocates jobs over many resources. Optimally allocating the jobs is important for cloud environments because efficiently using resources could improve the resource utilisation rate and the quality of service.

What is scheduling efficiency and how can schedulers be efficient? There are many metrics on which optimization can be done to improve the efficiency of a scheduler. 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 optimised 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. The next paragraph explains what it means to be a NP-hard problem and why 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 entirely 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, called NP-hard problems. A well-known NP-hard problem is the Sudoku puzzle, especially larger ones (Yato, 2003).

Job scheduling is a generalised version of the travelling 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 colouring problem is an NP-hard problem itself.

1.1.3 Heuristics-based Scheduling Methods

Many cloud job schedulers that are still used today are explicitly developed for the system it manages, based on simple heuristics and fine-tuned by trial and error. Tuning job schedulers is a tedious task. Repeating work of adjusting parameters and testing is very time-consuming. 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-Artificial Intelligence (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. All available resources are filled, and when a resource is available it is assigned to the task first in queue.

2. Shortest Job First (SJF): This algorithm sorts awaiting jobs based on increasing order of an estimation of the completion time. It works the same as FIFO for workload: when a resource is free, it is assigned a new job from the queue. It can be described as a priority queue.

The above-explained algorithms are highly intuitive resource managers and not fine-tuned for different workloads. Due to the lack of flexibility of these non-AI

algorithms, there are also situations in which their scheduling efficiency and precision will have deviation. More specifically, FIFO possibly leads to multiple jobs with a long duration time, blocking all other jobs till they finish. The disadvantage of SJF is that it can cause starvation, meaning that short jobs are constantly added and will never be executed.

1.1.4 Machine Learning Based Scheduling Methods

Currently, schedulers are not capable of handling differences in workload and have other shortcomings. There is a need for schedulers capable of handling increasingly complex large-scale systems. Due to the shortcomings of current non-AI implementations, research is done in using Machine Learning (ML) techniques to learn efficient scheduling. This research is discussed in Section 2. Machine learning tends to overfit, and that it does not generalize. It might not be flexible. Overfitting is also the case with Reinforcement Learning (RL)-based schedulers. Schedulers can show undefined behaviour when resources are added or removed. Thus, when a change in environment occurs, the RL-based scheduler gets retrained to work in the new environment. If retraining has to be done for every environment change, it would spend more time retraining than scheduling.

1.2 Motivation

Retraining is not ideal and should be prevented as much as possible. The disadvantages of it are that retraining...

- ... takes time, in which a sub-optimally working scheduler is still scheduling;
- ... costs computational resources itself, which can not be used for performing other jobs;
- ... costs money and contributes to global warming. Carbon emission of large ML models is a real issue which is currently researched on (Patterson et al., 2021);
- ... means that the scheduler is not robust enough. Improving the robustness of schedulers can improve the quality of service for an end-user, reduce the cost for a company and impact the environment.

1.3 Research Questions

The goal of this research is to integrate robustness into current state-of-the-art ML based job schedulers. This leads to the following research question:

How to select a method for reducing the 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 the 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 indicators for selecting methods that reduce retraining time. To find the indicators, it is first needed to assess RL-based schedulers. More robust (also called flexible) RL-based schedulers can work with environmental changes and thus need less retraining time. Even though the training time does not change, requiring less retraining still reduces the overall retraining time. It is also required to know what state-of-the-art approaches are for reducing retraining time. Based on aspects of state-of-the-art approaches, criteria can be formed to select approaches for reducing retraining time.

There are two challenges of the research questions. The first challenge is how a method for reducing the retraining time can be selected if there is are no indicators to assess robustness of a RL-based scheduler. Comparing is not possible without indicators. The second challenge is to create a testbed that can test every arbitrary RL algorithm on a Job Shop Scheduling (JSS) problem, using the same data.

1.4 Contributions

This thesis seeks to contribute the following to the field of RL-based schedulers:

1. This thesis discusses the methods for assessing the performance of RL models, explain the difference and why some methods might be better. This contributes to the field by helping others select a method of assessing the performance of a RL model.
2. This thesis discusses approaches to assessing the robustness of RL models. This contributes to the field by introducing the importance of robustness and explaining how a method for adding in robustness can be selected.
3. This thesis will contribute the field by listing the state-of-the-art approaches for reducing retraining time. This contributes to the field by helping others to select a method for reducing retraining time.
4. This thesis will contribute to an approach for selecting reinforcement learning-based schedulers on the following indicators: the overall performance of the model, the robustness of the model and the training time.

In this thesis related work will first be discussed in Section 2. Thereafter the methodology and implementation of the experiments is discussed in Section 3. Thereafter the experiments and their results are described in Section 4. After this the results will be discussed in Section 5 and a conclusion will be drawn in Section 6. This thesis will end with suggestions for future work in Section 7.

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 selecting related work is described. After this related work on the topics of RL-based schedulers, assessing the robustness of RL methods and approaches for reducing retraining time are discussed. Lastly, the gap in the related work is analysed.

2.1 Scope and Methodology

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

Comparing different methods and models is the main task in this thesis. When comparing methods of models, a justified method of assessing methods and models is required. It is also important that the comparison is made 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 related literature.

The methodology of the research in related work is twofold. Most of the work of others is discovered via Google Scholar[1]. This search engine searches through known scientific databases, e.g. arXiv[2], Research Gate[3], and additionally universities' websites, e.g. from the University of Amsterdam[4]. The other method for finding related work is by reading literature referenced in earlier found related work.

[1] scholar.google.com
[2] arxiv.org
[3] researchgate.net
[4] uva.nl

2.2 Related Work

2.2.1 Reinforcement Learning

Reinforcement learning, as it does not rely on labeled data and its interactive learning process, gets widely used in scheduling. This difference is very well explained in 'the most popular artificial intelligence textbook in the world'[5], Russell and Norvig (2010). Their explanation of RL is summarised in the following paragraph.

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 shared property: 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. RL is different because it does not depend on data, but rather learns from a feedback loop of rewards or reinforcements. It typically consists of one agent, a set of actions and a set of states. The agent accumulate rewards by performing series of actions in the environment. See Figure 2.1 for a flowchart of reinforcement learning. In many complex environments RL can be the most 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 does not need large amounts of data but an environment to interact via actions and rewards, lots of work has been done with investigating RL-based scheduling approaches.

Many reinforcement learning algorithms are used in state-of-the-art RL-based schedulers. Three different state-of-the-art RL schedulers are described. The first scheduler is DeepRM, presented in Mao, Alizadeh, Menache, and Kandula (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 model from Zhang et al. (2020) is not hard bounded by the instance size (Zhang et al., 2020, p. 5), making it more flexible and robust.

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

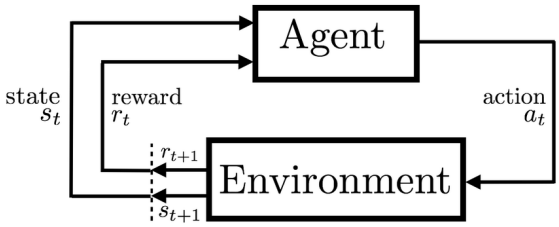

Figure 2.1: During a trial t the agent receives the state s_t and performs an action a_t in this state. After the action, it receives the new state s_{t+1} and a reward for the action r_{t+1} from the environment (Rafati & Noelle, 2019).

2.2.2 Evaluation of Reinforcement Learning Models

Finding approaches to effectively assess and compare RL-based schedulers is a sub-question of this thesis and, by that, a key factor to answer the research question. RL-based schedulers must be as effective after reducing retraining time as before the reduction. In the reviewed related work, there was no standard found for evaluating RL models. Work is done in defining standard evaluation methods, but these methods are not yet the industry standard. There are three representative metrics adopted by many works. 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 received reward for the actions taken by the optimal agent and the learning agent is the regret per action. However, knowing the optimal policy is not always possible. The second evaluation metric is not used much among literature but rather in practice, namely the OpenAI Gym. Gym provides well-defined environments that are suitable for RL to learn. A comparison between RL models is done on their leaderboards[6]. The metrics are the number of episodes before 'solve'. Some environments define being solved as receiving an average reward threshold over 100 consecutive trials (when a model is converged), and other environments have a clear goal which can be achieved. 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). Both of these are not used often in practice. It can be concluded that even though no standard exists, many

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

approaches for evaluating RL models exist and are used.

2.2.3 Assessing Robustness of Reinforcement Learning

To select a method for reducing retraining time, it is vital to assess the robustness of a model. Robustness is the ability to cope with dynamic environments and working 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 needs less time of adapting models to the new environment, which results in less retraining time. Because less retraining time can be achieved with more robustness, 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 essential for this research.

Many methods for assessing the robustness of reinforcement learning use some "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. Both of these methods require knowledge of the dynamics in the environment and the ability to simulate the dynamics.

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

Another approach to evaluating robustness is training on data and testing on completely other data. By evaluating the difference, the resulting model can be compared on how well it performs on the same values versus how it performs on other values. This is done in Pinto, Davidson, Sukthankar, and Gupta (2017), and other papers.

2.2.4 Approaches to Reduce Retraining Time

To answer the third sub-question, the current state-of-the-art approaches are discussed. The approaches are categorised into 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 categorical basis in the following sections.

Modelling the Dynamic Environment

Modelling the dynamics of the environment is a straightforward approach to taking the dynamics into account. For example, for job schedulers in cloud environments, this can be done by dynamically adding and removing resources during training. It theoretically helps the agent to learn the knowledge of dynamics. Numerous approaches include information about the dynamic aspect of the environment in 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 there are not many dynamics that can change. The disadvantage of modelling the dynamic environment is that lots of data is needed for modelling dynamics. Another approach to modelling dynamics is letting the dynamics randomly happen. It is unsure if this would actually work. Resource failure can be predicted using state information about resource state. This 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). The disturbing agent learns to perform the most disturbing action as possible. The same approach is also proposed in Pinto et al. (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. Good working policy-learning approaches is achievable only if the adversarial agent can control the time when specific resources are inserted or deleted. This raises a number of questions. How would the number 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 performed with Q-learning models. Q-learning (QL) is a model typically used in single-agent environments and is simple. Having two Q-learning agents in an environment, one learning the main task, and the other being the adversarial agent, can improve robustness. This technique has 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 robustly 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.

Meta-Learning

Meta reinforcement learning is a method for learning to quickly adapt online to new tasks. Adapting 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 is trained 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 the environment, the models parameters will probably be nearly the same as before. Another meta-learner is the Meta Reinforcement Learning Computation Offloading Solution (MRLCO) model from J. Wang, Hu, Min, Zomaya, and Georgalas (2020). In the article, a task offloading method based on meta reinforcement learning is proposed. This model uses a sequence-to-sequence neural network that is trained by order approximation and clipped surrogate objectives. The model reduces latency by up to 25% and can adapt fast to new environments. After the first time learning, it requires much less iterations to receive the same reward as previous training sessions.

2.3 Gap Analysis

As shown in this chapter, much work is done in optimising job scheduling with reinforcement learning. The popularity and convenience of RL algorithm is the reason it gets attention in the field of scheduling. The reason of our work choice to investigate reducing retraining time is because conventional RL is not robust enough to deal with dynamic scheduling scenarios. Many state-of-the-art schedulers using RL outperform heuristic-based methods with great distance. The downside of the RL-based schedulers is that they lack the ability to cope with the dynamics in the environment. Once dynamics occur in the environment, the RL model has to be retrained, which is time consuming. Most of them are trained for a specific

instance and did not train for dynamics. Including these aspects in training, along with methods for reducing retraining time or adding in robustness, can help reduce retraining time. Although there exist methods for reducing the retraining time, these methods have not been compared in JSS environments, trained on the same data. A comparison is needed to help with selecting a well-suited method. KPIs have to be found on which these reduce methods can be selected. There seems like there is not yet a standard for assessing the robustness of models, but combining multiple can have the desired result. To summarise, the gaps in related work are the following:

- Conventional RL is not robust enough to deal with dynamic scheduling scenarios;
- RL-based schedulers lacking robustness require more retraining time, which takes time, costs resources itself, costs money and contributes to global warming;
- There are no standards for assessing robustness of a model, but approaches from related literature can also be used here to assess the robustness.
- There has not been a comparative study of methods for adding in robustness of reducing retraining time in the same conditions;

This thesis will fill in the gap by providing a standardised method for selecting approach to reduce retraining time. It does this by having a solid approach for assessing robustness and overall performance of RL-based schedulers.

Chapter 3

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 to evaluate how well the approaches work on reducing retraining time.

Firstly, the workflow of the experiments is described. Secondly, the software and hardware for the experiments is described (§3.2). Lastly the data and its usage are described in Section 3.3.

3.1 Workflow

Two approaches are taken to address the research questions. The three sub-questions required review of literature to answer the sub-questions. The methodology to discover literature is described in Section 2.1. After literature review, the state-of-the-art meta-learning model and other state-of-the-art RL models were selected to assess robustness. The data had to be converted into the desired data format to run the models. The methodology of converting this data is described in Section 3.3. After data was converted, the algorithms were run in their specific environments. Setting up the environments is described in Section 3.2.

3.2 Hardware and Software Specification

This section explains the hardware and software setup and configuration used for the research. Firstly the hardware used is described, and after the hardware some vital 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 machine, with an Intel i9-9900

(16 threads) CPU running at 5.000 GHz, and a NVidia 2080 Ti (11 GB) GPU with CUDA 10 and driver 415. The operating system is Ubuntu 18.04.4. All algorithms were implemented in the Python programming language[1]. A virtual Python environment was created for every experiment to prevent versioning issues between dependencies. 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
```

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 the 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`.

Frequently used Python packages in scientific computing are the following: NumPy, Scipy, Keras, Tensorflow and PyTorch[3]. NumPy and SciPy provide an API for scientific computing and a solid multidimensional array object. The other three provide a back-end for creating well-optimised ML algorithms, especially for Neural Network (NN) . Another software project used extensively with this thesis is the OpenAI Gym (Brockman et al., 2016). Gym is a package for developing and comparing reinforcement learning algorithms. It provides access to many environments to develop a RL algorithm on.

3.3 Workload Data Set

The supervisors provided four datasets at the start of the thesis. The data is log data from the Euro-Argo project[4]. The datasets described received 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,

[1] python.org
[2] github.com/pyenv/pyenv
[3] Their respective home pages: numpy.org, scipy.org, keras.io, tensorflow.org and pytorch.org.
[4] euro-argo.eu/

byte count and completion status. An overview of the datasets is provided in Table 3.1.

Table 3.1: 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	-

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 leads to problems during converting the data into another format. Another quirk to note is that even though the column name 'request ID' suggests 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 the `mts_june_10m.csv` file. All counts of unique IDs and rows are shown in Table 3.2. The commands for getting this data can be found in Appendix A.

Table 3.2: 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%

3.3.1 Converting to other data formats

The Euro-Argo dataset was not compatible with most models. It is not compatible with most models because it is not a 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[5], the Euro-Argo data was converted to a JSS using DAGGEN[6]. 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 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 the 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 a uniform distribution $[4, 10]$, for ccr it was a uniform distribution in $[3, 6]$ and for fat it was a 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 3.1. It can be seen in Figure 3.1 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 described above.

[6] github.com/frs69wq/daggen

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

Figure 3.1: 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"]
}
```

Taillard Specification

Other models worked with the popular OpenAI Gym environment. The Gym environment allows people to build custom environments to make it possible to test RL algorithms in other kinds of environments. Gym itself 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.

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 3.2.

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 analysed the existing instances. In the

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

Figure 3.2: A computing instance following the Taillard specification.

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 3.3. The Euro-Argo 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 Taillards' specification we only used the transfer time. The transfer time 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.

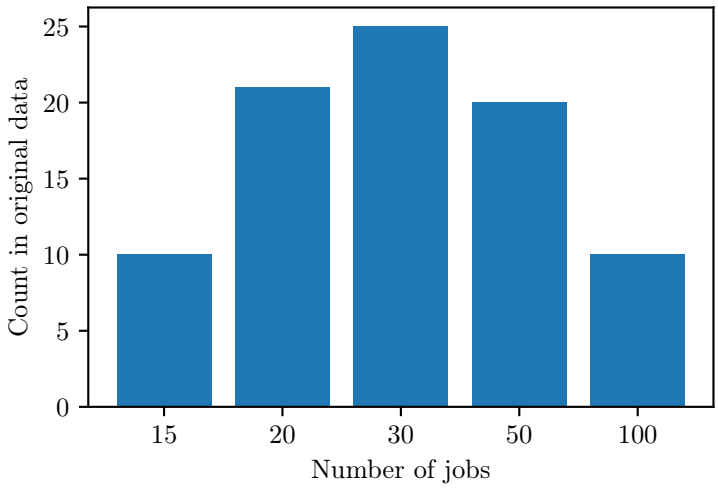

Figure 3.3: Distribution of number of jobs in data from Tassel et al. (2021).

Chapter 4

Experiments

In this section, the performed experiments are presented, and results of the experiments are shown. The two experiments were performed on the workstation described in Section 3.2. Both experiments were programmed in Python, and open-source software projects for the models and environment were used extensively.

4.1 Characterizing Meta Reinforcement Learning

The objective of the first experiment was to evaluate the adaptiveness and reduce in retraining time with meta RL . The evaluation is done using the meta RL offloading model, called MRLCO , proposed in J. Wang et al. (2020). The data was converted into JSS data (see 3.3). Running the experiments with the converted data required modifications in the source code of MRLCO . The modifications were made in the `meta_trainer.py` file. There was a list of directories containing 100 graph data files, which was changed into the directories with the Euro-Argo data. More modifications have been done to fix bugs, e.g. adding in logic for taking the maximum value of an array when it is empty.

After the bug fixing, running the model was possible. 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 comma separated value (csv) 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

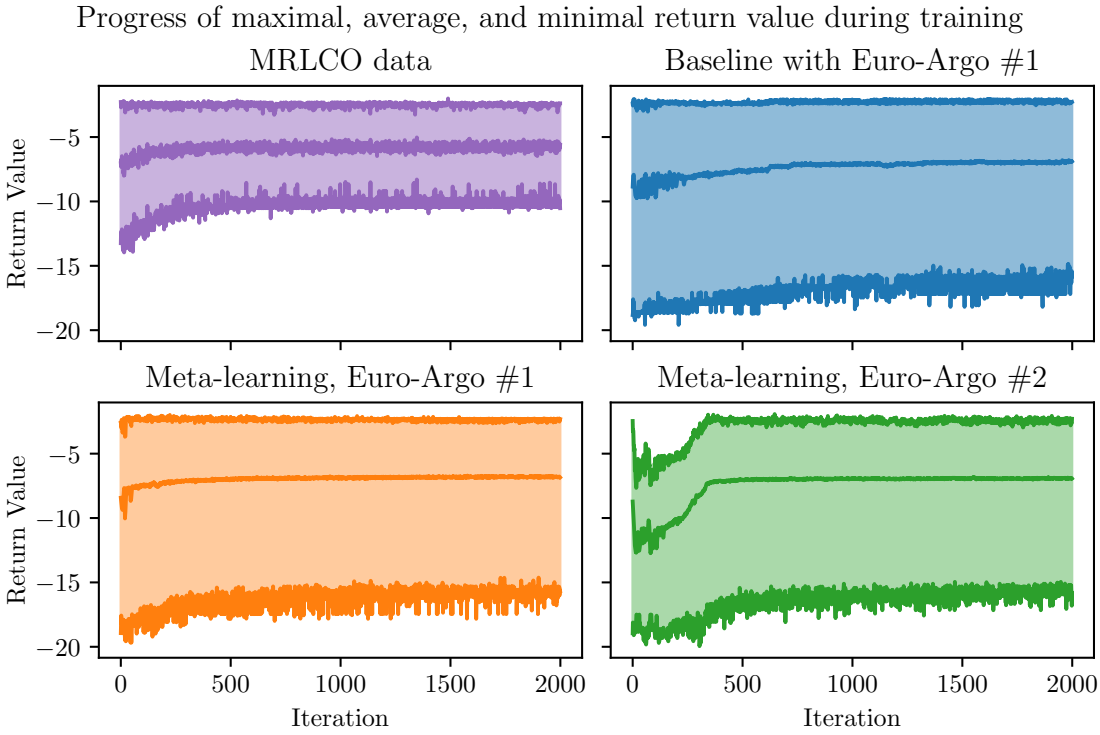

Figure 4.1: The progress of the return value during training. The upper line plots the maximal return value per iteration, the middle line the average return value and the lower line plots the minimal return value per iteration. The #1 and #2 are respectively the first and second batch of the Euro-Argo data.

Table 4.1: The average policy and environment execution time per run. The values are in seconds.

ID	Batch	Policy	Environment	Total
1	1	4955	1907	6862
2	1	4971	1907	6878
3	2	5013	1822	6835

4.1.1 Recreating MRLCO Results

The first experiment was recreating the results from the paper. This was done by running the MRLCO model without changes on the provided data. The two runs are plotted in Figure TODO. It is visible that the second run converges quicker due to meta-learning. The reward in the last iteration of training was -5.42. The average return value at the last iteration was -5.60.

4.1.2 MRLCO with Euro-Argo Data

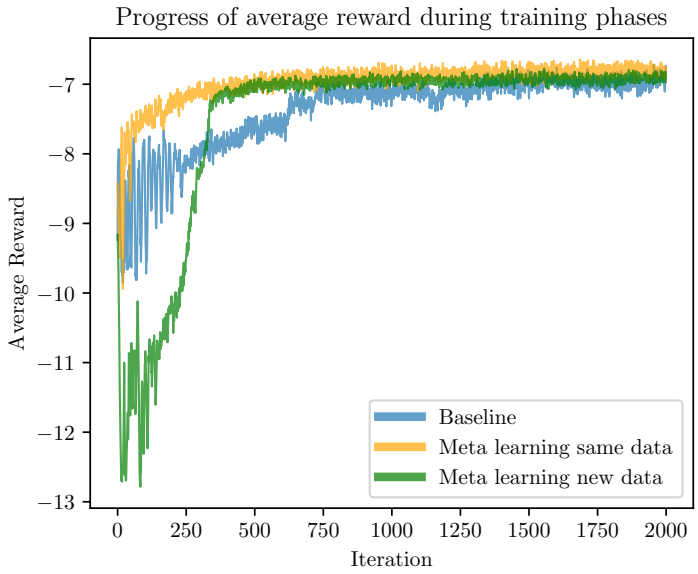

Figure 4.2: The progress of average reward during training Euro-Argo log data. The data was split in 2 batches of the same size. The blue line is the first time learning, so no meta-learning involved. After this, a model using the same batch data (orange) and a model using the other batch of data (green) were trained.

The second set of experiments was done with the converted Euro-Argo data. Three training phases were done. The first training phase was a baseline run, without any meta-learning up front. It was trained on the first half of the data: 1500 graph files. This half will from now on be called batch 1, the other half batch 2. After 1955 iterations the baseline model reached its optimal reward of -6.7. Now the baseline model is trained, it can be used for meta-learning. Two meta-learning runs were done: one with the exact same data, batch 1 and another one with different data from the same distribution, batch 2. These runs reached their optimal reward after respectively 521 and 920 iterations. The real time of these iterations are listed in 4.2. In the table it is shown that the training time speedup on the same data is 105 minutes, from almost three hours to a bit more than one hour. The training time speedup on new data is 54 minutes. The time difference between the full training sessions is not significant. In Table 4.1, the total execution times for policy and environment are shown. The difference in execution time between the different runs are only a couple seconds. The maximal reward on the second data batch is -6.8, a 0.1 difference with the maximal reward from the base line model. The progress of average reward for all three training phases are plotted in Figure 4.2. In this figure it is clear that the meta-learned runs achieve their optimal policy in an earlier iteration. Even when the reward at the start is much lower, it still learns very quick using meta-learning.

The return value is plotted in Figure 4.1. The difference in return values between the runs using Euro-Argo is not significant. There is however a difference with the MRLCO data, which has a higher minimal return value. No reason is found for this difference. The maximal and average return value are around the same value.

Table 4.2: The time it takes to converge to the maximal average reward (-6.8). The time is expressed as the number of iterations and the real (sometimes called wallclock) time. An experiment of 2000 iterations took 255 minutes and 10 seconds (15310 seconds). The time to converge thus is calculated using the following formula: $it/2000 \cdot 15310$.

ID	Batch	Iterations	Time (min)
1	1	1345	171
2	1	521	66
3	2	920	117

4.2 Comparing Reinforcement Learning Model in OpenAI Gym

The second group of experiments were done to evaluate multiple RL algorithms as schedulers. A side objective of this experiment was to have a good testbed for other experiments in this research project. OpenAI's Gym was chosen for this because of its popularity and simplicity. As described in Section 2.2.2 and Section 3.2 OpenAI is a popular platform for creating, testing and evaluating reinforcement learning models. The environment used in this experiment is described in Section 3.3.1. The JSS environment (`JSSEnv`) in combination with the `keras-rl` package[1], would allow 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. As start make the Gym environment, after this define a NN (shown in Figure 4.3), then define a RL agent providing a policy and the NN and at last fit the agent on the environment.

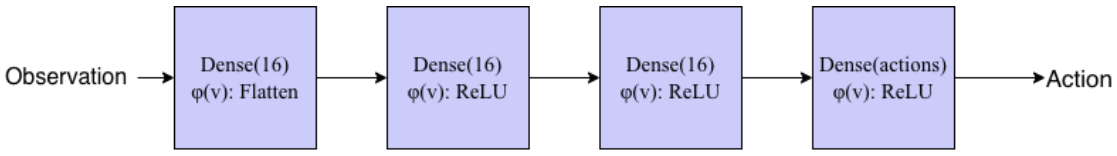

Figure 4.3: The DNNs used in the RL models from `keras-rl`.

Starting the experiment, a FIFO scheduler in the environment was run. 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 5.

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

Chapter 5

Discussion

This thesis, like every other thesis, has its achievements and shortcomings. These aspects of the thesis are discussed in this section. First, the achievements that are gained are summed up, and after this, the shortcomings are discussed. The obtained results are discussed and it is reflected on the methodology.

5.1 Achievements

The first experiment has successfully shown that meta-learning improves learning speed and reduces retraining time. In Table 4.1, it is shown that the execution time of policy and environment is the same for the first and second run, meaning that running 2000 iterations for both models has the same duration. However, running all 2000 iterations is not necessary. The first run baseline model achieved its maximum reward of -6.7 in iteration 1345 (Table 4.2). The second run, which was meta-learning, achieved the same reward in only 521 iterations. It can be stopped after this iteration. If the model is stopped after achieving the maximal reward of the baseline model, the retraining time can be reduced by 105 minutes, from 171 minutes to 66 minutes duration, a reduction of 61.4%. The progress of the average reward during training is plotted in Figure 4.2. The first run on Euro-Argo data does not have a tipping point, but the second run, which has meta-learned has a tipping point after around 250 iterations, and after only 280 iterations the reward is the same as the reward of the first run after 1250 iterations. This shows that meta-learning decreases the number of iterations needed for MRLCO to converge.

Meta-learning also works on new data. The model learned the second batch of the Euro-Argo data in fewer iterations than without meta-learning. It even learns fast despite the reward starting much lower than the reward of the first batch. Learning quicker with different data shows robustness. The model can work with different data and thus learns quicker to work in dynamic environments.

Although the reward on the Euro-Argo data fluctuates less than the MRLCO data, the return value of the Euro-Argo has a larger spread, i.e. the difference between the maximal return value and the minimal return value, see Figure 4.1. A reason to explain this difference between the MRLCO data and the Euro-Argo data is not found.

The first achievement was the successful converint of the Euro-Argo log data to the Taillard specification and the JSS data. Even though the JSS data provided by the MRLCO model had some quirks and was not as described how it should be, it was managed to convert the Euro-Argo data so that it works with the MRLCO model. The conversion provided insights into the many aspects of an instance for job scheduling. Many parameters were available to change the JSS data and change the job flow accordingly.

5.2 Shortcomings

During the experiments it was not managed to finish all experiments that were planned. 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 did 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.

Another shortcoming of the converting of data is that converting data into other formats can also create unfairness in the evaluation. It might be that one data format provides more information to the algorithm than other data formats. Receiving more information means learning can possibly go faster. Difference in data formats would make the comparison that was intended in this research invalid.

Chapter 6

Conclusion

The cloud is a network of nodes providing resources for computation. Jobs require a specific amount of resources to complete and needs to be scheduled on the nodes. As a result of the growth of cloud environments, the scheduling task is becoming more and more complex. Cloud systems are currently too complex for humans to create a scheduler with heuristics. Now that humans are not able to design efficient schedulers, reinforcement-learning-based schedulers learn to schedule jobs efficiently. Although these RL-based schedulers can achieve high efficiency, they lack robustness. After a change in the environment, i.e. the addition or removal of nodes, the scheduler has to retrain. In this thesis, a method for selecting approaches to reduce this retraining time is researched. This led to the following research question:

> How to select a method for reducing the retraining time of reinforcement learning based resource schedulers in cloud environments?

Three sub-questions were formed to answer the research question. The three sub-questions will now be answered by drawing a conclusion from the literature and results.

The first research question, "How can we effectively assess and compare RL-based methods in job scheduling?" is answered using literature. Although the OpenAI Gym is the most popular method for evaluating reinforcement learning algorithms, there is not yet a standard evaluation method. The metrics are the number of episodes before solve. Some environments define solving by receiving an average reward threshold over 100 consecutive trials, while other environments have a clear goal that can be achieved. It can be concluded that openAIs Gym is an effective method for assessing and comparing RL-based methods in job scheduling.

The second research question, "What are the indicators to assess the robustness of RL-based schedulers?", is also answered by literature. Indicators to assess the robustness of RL-based schedulers are their performance in noisy environments with

disturbance. Another measurement for assessing this performance is by changing the dynamic aspects of the environment. For methods using deep learning, the neural coverage is a metric for robustness. Since deep learning is used in combination with RL by many schedulers, this metric can also effectively indicate the robustness of RL-based schedulers. Thus, to assess the robustness of RL-based schedulers, the performance in noisy or dynamic environments can be used as an indicator. When the scheduler uses deep learning, the neural coverage is also an available indicator for robustness.

The last research question, "What are the state-of-the-art approaches for reducing retraining time?", is answered by three types of approaches. The first approach is modelling the dynamics in the environment. The state-of-the-art method in this approach is Nagayoshi et al. (2013). The second approach is the use of adversarial agents. These agents learn to disturb the actual agent as much as possible. The state-of-the-art research on this is Chen et al. (2018). The last approach is meta-learning. Meta-learning learns how to learn, which can improve learning efficiency. The MRLCO model from J. Wang et al. (2020) is the state-of-the-art meta-learn, and also used in the experiments.

With an answer found for the three sub-questions, the research question will be answered. Again, the research question of this thesis is "How to select a method for reducing retraining time of reinforcement learning based resource schedulers in cloud environments?". A method for reducing the retraining time of RL-based schedulers can be selected on three criteria: the overall performance of the model, the robustness of the model and the training time. The first criterion is needed to prevent the method from deteriorating the actual model. The overall performance should not be worse than before. The second criterion, robustness, can reduce the frequency of retraining. If a model is more robust it might only have to retrain on large environmental changes, which means fewer retraining, reducing training time. The last criterion is reducing the retraining time. A method might require as many training sessions as first, but if it greatly improves the time it takes to train, it would also be an improvement. As discussed in Section 5, in Figure 4.2 it can be seen that meta-learning requires fewer iterations to achieve the same reward as the previous training session.

To conclude, selecting a method for reducing retraining time of RL-based schedulers can be done on three criteria: the overall performance of the model, the robustness of the model and the training time of the model.

- The overall performance of the model, assessed by the performance in a Gym environment or comparing the received reward to other models.
- The robustness of the model, assessed by including disturbances in the environment or an adversarial agent, or by controlling the dynamics of the environment.

- The training time of the model, which can be reduced by modelling the environment, adding in robustness through adversarial agents and meta-learning.

Chapter 7

Future Work

In this section approaches for improving this reseach are suggested. The current shortcomings of the field and how they can be prevented are discussed.

7.1 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 papers themself do not document how code works, and additional documentation is not written either. Working with the codebase requires reading the written code, which usually lacks comments and good design patterns, and figuring out how to adjust it to work with other 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 attribute is not something that can be done with `DAGGEN` and must be later on added by the authors of the research paper, which should definitely be documented. 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. Good practice in future research will be to better document code and make it available for others to experiment with the code.

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

7.2 Standards in Reinforcement Learning

In future research, more research is needed to design standards for research in reinforcement learning. Three standards would significantly improve comparative research.

The first improvement would be a standard in data formats used in RL-based schedulers. The data formats differ too much to compare RL-based schedulers rightfully using different data formats. The second improvement would be the exposed API for environments. OpenAIs Gym is a step in the right direction. The API is clear and straightforward. The exposed functions and attributes as described in Section 3.3.1 should be the standard for every JSS environment. The third improvement would be a standard in the API exposed by the model. It should be intuitive to provide a policy, environment and hyperparameters to a RL-based scheduling model.

In addition to improving standards to improve comparative studies on RL-based schedulers, future work should be done on reducing retraining time. Comparing methods for reducing retraining time is an important subject and should be better executed in the future. If new standards are adopted in the field, it should cost less time to compare methods for reducing retraining time. New methods could then be actively compared with other methods in a controlled, unbiased environment. Cloud providers and future researchers can then select a method to reduce retraining time in their cloud environments. Reducing retraining time reduces the energy usage and costs required for training machine learning algorithms. Reducing the retraining time is better for cloud providers, end-users and the environment.

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).

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.
Rafati, J., & Noelle, D. C. (2019). Learning Representations in Model-Free Hi-

erarchical Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, pp. 10009–10010). doi: https://doi.org/10.1609/aaai.v33i01.330110009

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 3.2.

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
```