Chapter 3

Method

In this chapter the methodology of this thesis 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 (§3.1). Secondly the data and its usage is described in Section 3.2.

To get the results needed for answering the research questions we first compare performance of the RL based schedulers in question (§??), then we integrate the approaches for reducing retraining time or robustness (§??) and at last we evaluate the robustness and duration of retraining time of these approaches on all RL algorithms (§??).

3.1 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

[1]python.org

[2]github.com/pyenv/pyenv

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 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].

3.2 Data

Four datasets were provided by the supervisors at the start of the thesis. 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 3.1.

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

[4]gym.openai.com

[4]Their respective home pages: numpy.org, scipy.org, keras.io, tensorflow.org and pytorch.org.

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	-

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 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%

The provided 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. 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], our provided sequential 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 our provided data was originally not JSS data, the converted data used parameters and other aspects from

[6]github.com/frs69wq/daggen

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 Fig. 3.1. It can be seen in Fig. 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 mentioned above.

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"]
}
```

Other models worked with the popular OpenAI Gym environment. The Gym

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

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.

In Tassel, Gebser, and Schekotihin (2021) a gym environment 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 Fig. 3.2.

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

Figure 3.2: 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 provided sequential 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 Fig. 3.3. 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 provided 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.

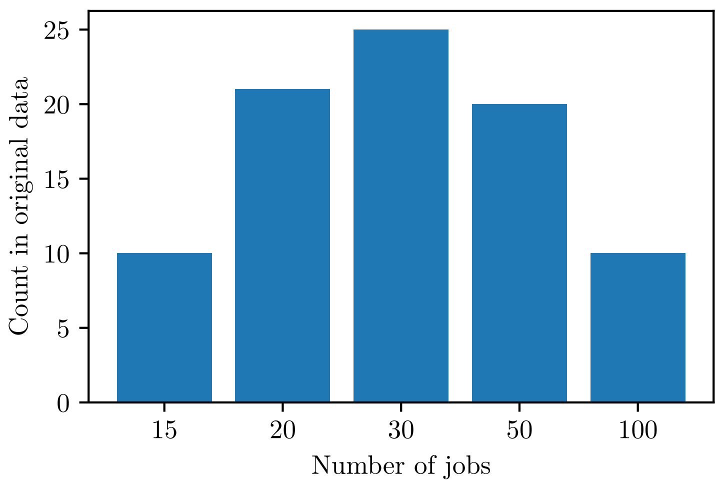

Figure 3.3: Distribution of number of jobs in data from Tassel et al. (2021).

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

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.*

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

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.

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`

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 *Advances 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
```