Contents

Acronyms

AI Artificial Intelligence. 5–7, 9

DNN Deep Neural Network. 9

DQN deep Q-learning. 11, 12

KPI Key Performance Indicator. 8

ML Machine Learning. 6–8

PPO Proximal Policy Optimization. 9

QL Q-learning. 12

RL Reinforcement Learning. 5–11, 13

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 cloud environments are discussed. Secondly some theoretical background about the problem of job scheduling and its NP-hardness is provided in Section 1.2. Lastly, in the Section 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 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.

1.2 Job Scheduling

Imagine cloud systems as described in Section 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.

1.2.1 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, a way of 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.

1.3 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.

1.3.1 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).

1.3.2 The Problem

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

Related Work

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 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 way 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 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.

2.2.1 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 way to train 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.

2.2.2 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 model from Zhang et al. (2020) is not hard bounded by the instance size (Zhang

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

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

2.3 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).

2.4 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. This is way 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.

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

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

2.5 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.

2.5.1 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.

2.5.2 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.

2.5.3 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.

2.6 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 2.5 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.

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 Approxima-

tion. In *Nips* (Vol. 99, pp. 1057–1063). Cambridge, MA, USA: MIT Press.

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).