Online Immediate Commitment Job Scheduling: A Machine Learning Approach

Project Report

Submitted in partial fulfillment of the requirements for the award of

Master of Science Degree in

Automation and Robotics

of the Technische Universität Dortmund

Submitted by

PHILIP VARGHESE MODAYIL(229225)

Guided By
Prof. Dr.-Ing. Uwe Schwiegelshohn
Ganesh Kamath Nileshwar
Samin Jamalabadi

Institut für Roboterforschung

Technische Universität Dortmund

2022

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Background

System providers rent out infrastructure (usually but not exclusively computing power) to clients in the Infrastructure-as-a-Service business model of cloud computing. They are in charge of management of the system and resource allocation The business strategy may include numerous tiers of customer support. For instance, some routine work have a low level of urgency, whereas time-sensitive jobs have a high level of urgency and it necessitates a near-immediate completion.

The agreement is binding once a customer has rented infrastructure from the service provider. As a result, we expect the admission policy to commit to each decision it makes right away. Immediate Commitment, also known as commitment on arrival, is the most powerful and ideal type of commitment: we must determine whether or not to execute a job soon after it is submitted.

Our goal is to create an approach that commit right away, meaning that when a work is submitted, the algorithm either rejects or allocates it.

1.2 Problem Statement

Experiment with a Machine Learning approach for job scheduling on parallel machines with immediate commitment. This project aims at studying the possibilities for the machine to learn something from the job scheduling environment. The machine should then be able to accept or reject the incoming job with immediate commitment, based on its judgement of the current states of the machines and the properties of the incoming job.

1.3 Motivation

Many recent triumphs, such as Atari (1), AlphaGo (2), and AlphaStar (3), have piqued the curiosity of the AI community. The majority of methods in this discipline rely on well-known benchmarks from the OpenAI gym library or video games (Atari, Starcraft, Dota, and so on). Researchers quickly identified the potential benefits of DRL in practical domains such as Combinatorial Optimization Problems as a result of these successful implementations.

Since the Immediate commitment job scheduling problem is an extremely random process it is not feasible to identify any pattern from it. Hence, conventional machine learning approaches cannot be deployed. Since the job scheduling problem can be considered as a game between an adversary and online player, it makes sense for the machine to consider multiple scenarios of the game and learn a strategy to predict the next action based on the current states. Hence, reinforcement learning is the best fit for this approach.

Chapter 2

LITERATURE REVIEW

In (4) the researchers speaks about the issues faced by standard algorithms like greedy best fit in the sense of its lower bound, when faced with an adversary. The algorithm here rejects jobs based on the comparison of job's deadline with the deadline threshold at that time. The deadline threshold is calculated dynamically and will vary as per the current machine state. The deadline threshold is calculated based on the "f values" specific for the slack of the incoming job. Hence, the algorithm take the machine states and the job properties into account. This paper teaches us that the jobs need to be judged based on a rejection criterion even if it is legal to accept it

However, the algorithm needs to calculate the "f values" for different slack and number of machines used. These heavily mathematical calculations needs to be performed as the situation changes. Hence, a machine learning approach needs to be tried and tested to possibly map the current machine states and incoming job properties to a desirable action(accept or reject).

In (5) the researchers talks about tackling a job shop problem, where different jobs must be allocated resources, with Reinforcement Learning. This paper does give a little insight into how the environment can be set up for the learning part. This problem however, allows for preemption which is not allowed in an immediate commitment job scheduling technique. We have taken inspiration from this paper and from existing inverted pendulum environment in OpenAI GYM.

In (1) the researchers talk about tackling an Atari game with Deep Reinforcement Learning (DQN). It judges what the next action must be based on the current state of the game and the incoming bricks. This is very much similar to our problem with machine states and incoming job properties. Taking inspiration from this paper we can decide to move forward with Reinforcement Learning as our strategy to the problem at hand.

Chapter 3

METHODOLOGY

3.1 An Example

Consider an example with two parallel machines (`m_1` and `m_2`) and three incoming jobs with the properties release time(`r_j`), processing time(`p_j`), deadline(`d_j`). Let the incoming jobs be of the properties: `j_1`(`r_j`:0, `p_j`:10, `d_j`:11), `j_2`(`r_j`:0, `p_j`:10, `d_j`:11), `j_3`(`r_j`:0, `p_j`:30, `d_j`:33). While deploying the state-of-the-art algorithms like Greedy best fit, we end up with `m_1` taking `j_1` and `m_2` taking `j_2` and rejecting `j_3`. This approach loses `j_3` which was the higher processing time and hence higher preference job, due to illegal scheduling with deadline conflict. We then pose the question, what if the system rejects `j_2` and wait for `j_3` to schedule `j_3`. If the system manages to do that then we end up with `m_1` taking `j_1` and `m_2` taking `j_3`. This would result in a total accepted processing time of 40 rather than the earlier result of 20.

We can conclude that, some jobs must be rejected in anticipation of higher processing time jobs which may come later. Such a dynamic job scheduling approach might result in a better total accepted load.

3.2 Rejection Criterion

Typical algorithms supply the inputs and rules to get the output, where as in machine learning we supply input and outputs and the machine learns the rules. As we are expecting the machine to decide on the action taken, there is no hard rejection rule given.

Since the machine learning approach used in this project is reinforcement learning, it is required to give rewards for the actions taken during the training process. Machine learns to take actions which would provide maximum reward.

The reward choice in this project was a dense reward approach where we give reward for the action taken at each step. This was done as we are trying to make the machine learn to take profitable actions. Hence, we provide a reward or penalty for each step. We are opting out of the sparse reward option where we provide a reward at the end of one cycle as we do not posses an optimal output to compare our results with.

3.3 Profitability Of Accepting Job

Since the rejection of jobs are required in anticipation of higher processing time jobs, it is required to determine a profitability factor for the incoming jobs. Here, we determine the profitability of the incoming job based on the amount of impact it can make on the current state of the machines.

First, all the machines' states are updated with the current remaining load on the machine since the arrival of job J. We take the position number of the mean of the loads on the system `pos_mean`. Then, we consider the situation where J is scheduled to the least loaded machine m'. After rearranging the system in descending order, we check for the new position of m' `leap_pos`. Since the state of the machine changes dynamically with each acceptance and rejection, the threshold for the leap of machine m' must change dynamically. We take threshold to be the position number we get from the difference between total number of machines and the `pos_mean` from the rear end. This allows for the threshold to be close to the front end after a series of acceptance and close to the rear end after a series of rejections.

We then say that a job J is profitable if its `leap_pos` is over the threshold. We

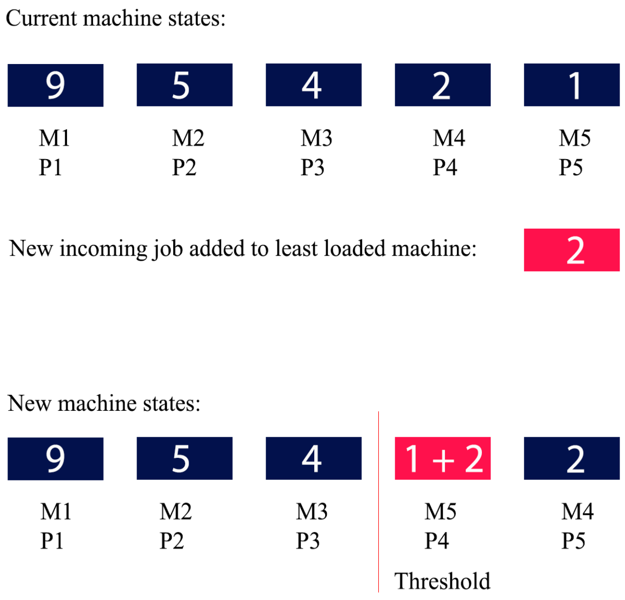


Figure 3.1: Profitability Example

shall use this profitability factor to give rewards for the actions taken by the reinforcement learning agent.

Figure 3.1 shows an example. Here the current `pos_mean` is P3 and hence threshold will be two positions from the rear end (P4). The `leap_pos` for M5 is P4 and hence this job does not make the leap beyond the threshold. This job is regarded as non profitable in our training approach.

Chapter 4

LEARNING ENVIRONMENT AND SET UP

4.1 Action Space

The action space used for training is a random discrete action space. Since the action that needs to be taken by the agent is either accept or reject the incoming job.

Num	Action
0	Reject the job
1	Accept the job

Table 4.1: Action Space

4.2 Observation Space

One of the challenges faced during the training part was to incorporate the different scenarios generated by the usage of different number or machines used within a fixed observation space. This was tackled with idea of using a five-point summary or better known as box plot of data. The remaining current load on the machines are described with the help of box plot summaries and hence we always obtain a fixed five data point summary.

The observation space includes the remaining load summary (box plot five-point summary) on the machines and the incoming job profile (release date, processing time, due date).

The observation is a ‘ndarray‘ with shape ‘(8,)‘ where the elements correspond to the following:

Num	Observation	Min	Max
0	max remaining load	-Inf	Inf
1	upper quartile	-Inf	Inf
2	Median	-Inf	Inf
3	lower quartile	-Inf	Inf
4	min remaining load	-Inf	Inf
5	processing time	-Inf	Inf
6	release time	-Inf	Inf
7	due time	-Inf	Inf

Table 4.2: Observation Space

4.3 Reward Scheme

Since we are expecting the agent to learn to take profitable actions, we provide each step with profitable action a +1 and non-profitable action a -1. This enables us to try and maximize the number of positive steps.

job assigned	Profitable	Reward
True	True	+1
True	False	-1
False	True	-1
False	False	+1

Table 4.3: Reward Scheme

Chapter 5

TRAINING

5.1 Implementation

We have used stable baselines 3 libraries for Reinforcement Learning implementation. We have taken inspiration from the training environment of the inverted pendulum model in stable baselines 3. The learning environment is implemented with OpenAI gym toolkit.

We have trained the environment in different scenarios supplied by an enormous data set that that contains ten days of worth data (Day 1 – 10: can be found in the github repository). Each day's data contains 990 text files with over 10378 job details in each of them. Hence a total of 9900 different scenarios were considered while training.

The reinforcement learning algorithm used here is the DQN algorithm. Here the environment is highly non deterministic since the next incoming job details and possibilities is unknown to the agent. The usage of DQN is mostly for deterministic environments where, if we had a function Q:StateXAction->R that tells us what the reward we shall get then it is possible to create a policy that maximizes our rewards. This is parallel with our problem and hence we have chosen to use DQN algorithm even though our environment is highly random.

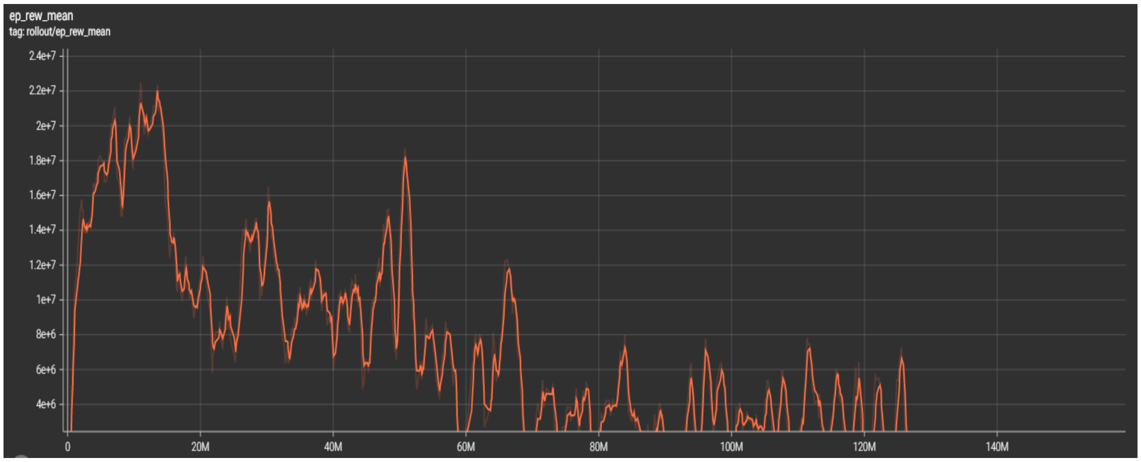


Figure 5.1: Reward Mean

5.2 Training Results

The environment was trained for 130 million timesteps (120 hours) and a model generated at each 1 millionth timestep. Figure 5.1 shows the tensorboard depiction of the training log data.

The peak reward was obtained at the 13.53 millionth timestep and hence we shall regard the model generated at 14 millionth timestep to be the optimal model obtained from the training.

Chapter 6

TESTING AND RESULTS

With the saved model in "14000000" zip folder we have run the testing on the files of Day 11 for number of machines (10,20,30,. . . . . ,200). The resulting values obtained from the testing is shown graphically in Figures 6.1 to 6.4. We should note in Figure 6.3 and 6.4 that higher the Performance Ratio worse the performance of our algorithm.

Parameters included in the test results are: Accepted load, Rejected load and Available resources, all averaged for the 10 files ranging from (S1 – S10) for the same slack and standard deviation.

Accepted Load: It is the total sum of job processing times accepted by the system of machines.

Rejected Load: It is the total sum of job processing times rejected by the system of machines.

Available Resources: min(total submitted load (accepted load + rejected load) , number-of-machines x latest-completion-time-of-a-job-in-any-of-the-machines)

Performance Ratio: Load processed in an optimal scenario / Load processed by our algorithm(Accepted load)

Load Processed in an optimal scenario: max(Available Resources Greedy, Available Resources ML)

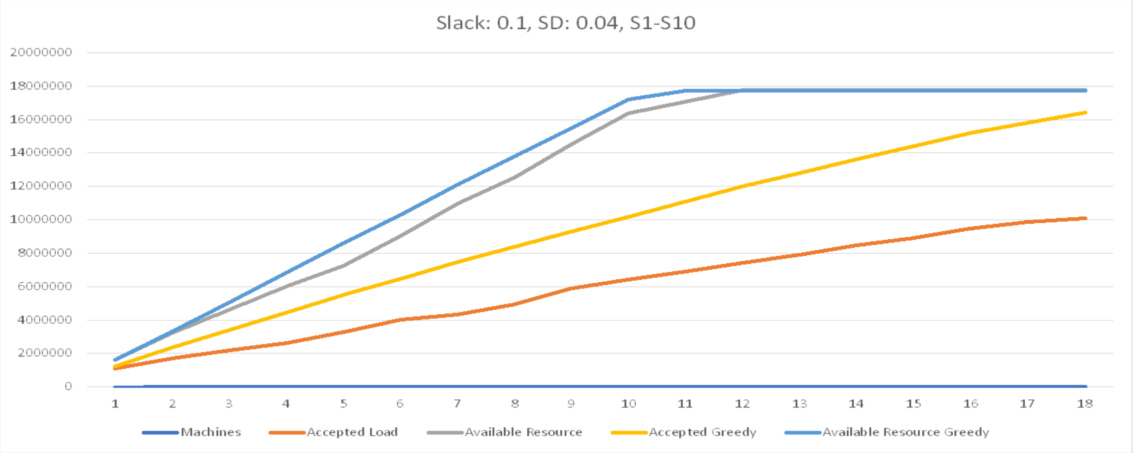


Figure 6.1: Results for Data:Day 11, Slack:0.1, SD:0.04

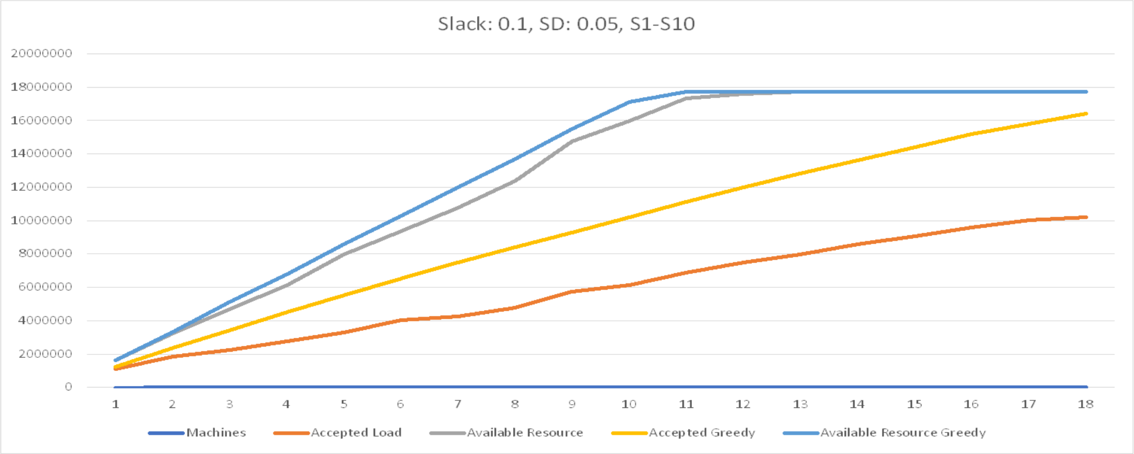


Figure 6.2: Results for Data:Day 11, Slack:0.1, SD:0.05

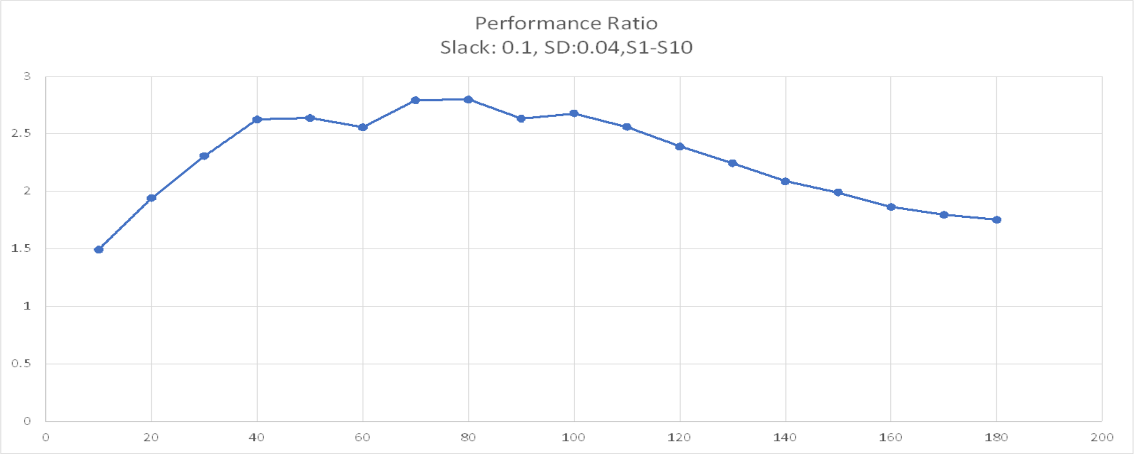


Figure 6.3: Performance Ratio of Results: Day 11, Slack:0.1, SD:0.04

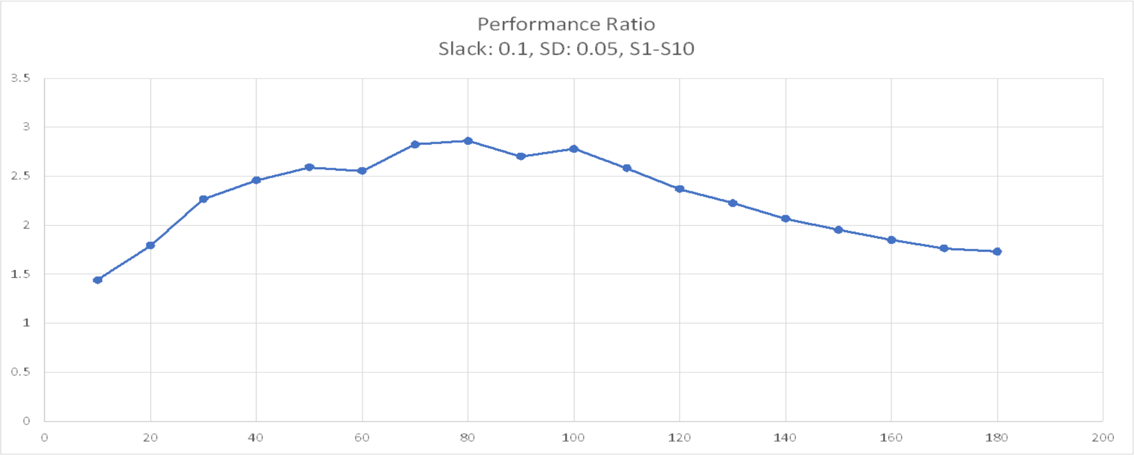


Figure 6.4: Performance Ratio of Results: Day 11, Slack:0.1, SD:0.05

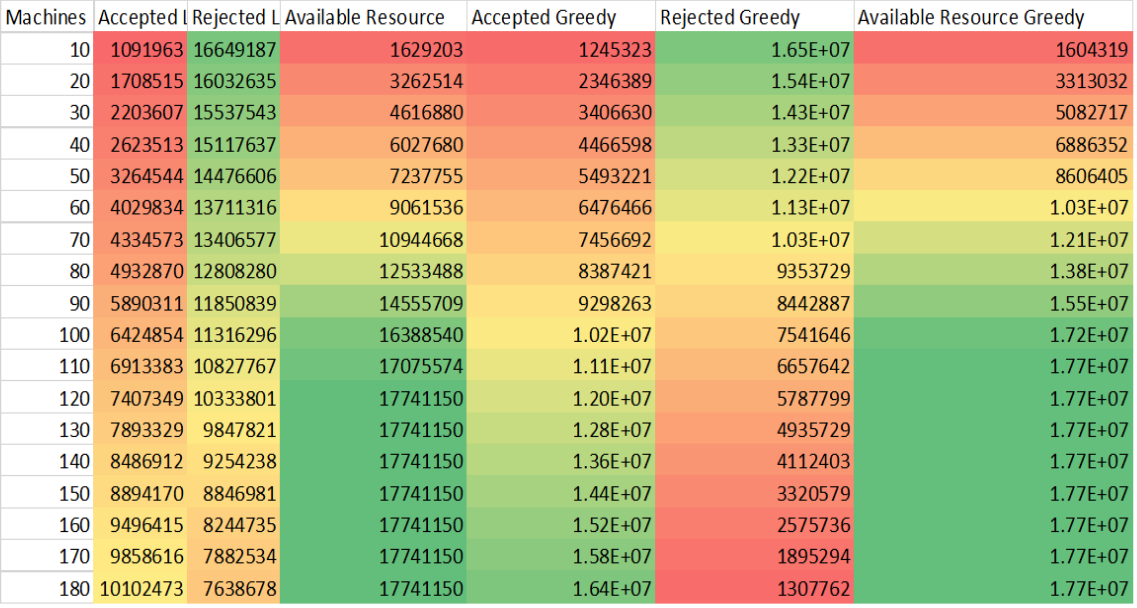
Chapter 7

OBSERVATIONS

An overview is provided by Figure 7.1 and Figure 7.2 (Red Colour: lowest, Green Colour: Highest). We can observe that, as the available resources increases, greedy best fit algorithm out performs our machine learning approach by large. However, for less resources the ML approach is comparable to the greedy best fit algorithm. A close scrutiny of the results can be seen in Table 6.1 and Table 6.2. We can make the observation that the performance ratio of the ML approach for number of machines 30-140 is quite bad. Whereas, the performance ratio of the ML approach is in the satisfactory range for number of machines 10,150,160,170,180. It is to be noted that for number of machines greater than 180, we will have idle machines, that is all the submitted load can be accepted into the system.

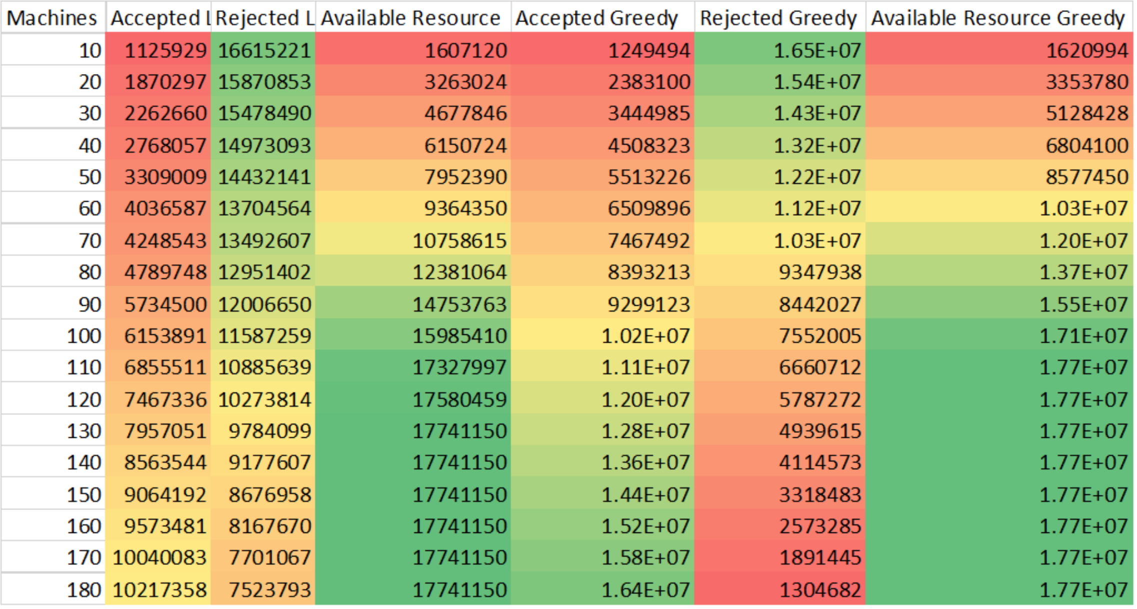
A similar observation can be made from Figures 7.3 to 7.6, it represents the difference in the performance ratio of greedy best fit algorithm to that of our ML approach. Just like our previous observation we can see that the difference steeps down for average amount of resources. From these heat maps we can understand that, greedy best fit algorithm has better performance ratio than our ML approach.

We can look at Figure 7.7 and observe that the exploration rate while training was constant. We have trained for more than 130 million timesteps and from Figure 5.1 we understand that the reward mean converges to an almost constant range. From these observations we can argue that more training time will not effectively bring better results.



Machines	Accepted l	Rejected L	Available Resource	Accepted Greedy	Rejected Greedy	Available Resource Greedy
10	1091963	16649187	1629203	1245323	1.65E+07	1604319
20	1708515	16032635	3262514	2346389	1.54E+07	3313032
30	2203607	15537543	4616880	3406630	1.43E+07	5082717
40	2623513	15117637	6027680	4466598	1.33E+07	6886352
50	3264544	14476606	7237755	5493221	1.22E+07	8606405
60	4029834	13711316	9061536	6476466	1.13E+07	1.03E+07
70	4334573	13406577	10944668	7456692	1.03E+07	1.21E+07
80	4932870	12808280	12533488	8387421	9353729	1.38E+07
90	5890311	11850839	14555709	9298263	8442887	1.55E+07
100	6424854	11316296	16388540	1.02E+07	7541646	1.72E+07
110	6913383	10827767	17075574	1.11E+07	6657642	1.77E+07
120	7407349	10333801	17741150	1.20E+07	5787799	1.77E+07
130	7893329	9847821	17741150	1.28E+07	4935729	1.77E+07
140	8486912	9254238	17741150	1.36E+07	4112403	1.77E+07
150	8894170	8846981	17741150	1.44E+07	3320579	1.77E+07
160	9496415	8244735	17741150	1.52E+07	2575736	1.77E+07
170	9858616	7882534	17741150	1.58E+07	1895294	1.77E+07
180	10102473	7638678	17741150	1.64E+07	1307762	1.77E+07

Figure 7.1: Heat Map: Day 11, Slack:0.1, SD:0.04



Machines	Accepted l	Rejected L	Available Resource	Accepted Greedy	Rejected Greedy	Available Resource Greedy
10	1125929	16615221	1607120	1249494	1.65E+07	1620994
20	1870297	15870853	3263024	2383100	1.54E+07	3353780
30	2262660	15478490	4677846	3444985	1.43E+07	5128428
40	2768057	14973093	6150724	4508323	1.32E+07	6804100
50	3309009	14432141	7952390	5513226	1.22E+07	8577450
60	4036587	13704564	9364350	6509896	1.12E+07	1.03E+07
70	4248543	13492607	10758615	7467492	1.03E+07	1.20E+07
80	4789748	12951402	12381064	8393213	9347938	1.37E+07
90	5734500	12006650	14753763	9299123	8442027	1.55E+07
100	6153891	11587259	15985410	1.02E+07	7552005	1.71E+07
110	6855511	10885639	17327997	1.11E+07	6660712	1.77E+07
120	7467336	10273814	17580459	1.20E+07	5787272	1.77E+07
130	7957051	9784099	17741150	1.28E+07	4939615	1.77E+07
140	8563544	9177607	17741150	1.36E+07	4114573	1.77E+07
150	9064192	8676958	17741150	1.44E+07	3318483	1.77E+07
160	9573481	8167670	17741150	1.52E+07	2573285	1.77E+07
170	10040083	7701067	17741150	1.58E+07	1891445	1.77E+07
180	10217358	7523793	17741150	1.64E+07	1304682	1.77E+07

Figure 7.2: Heat Map: Day 11, Slack:0.1, SD:0.05

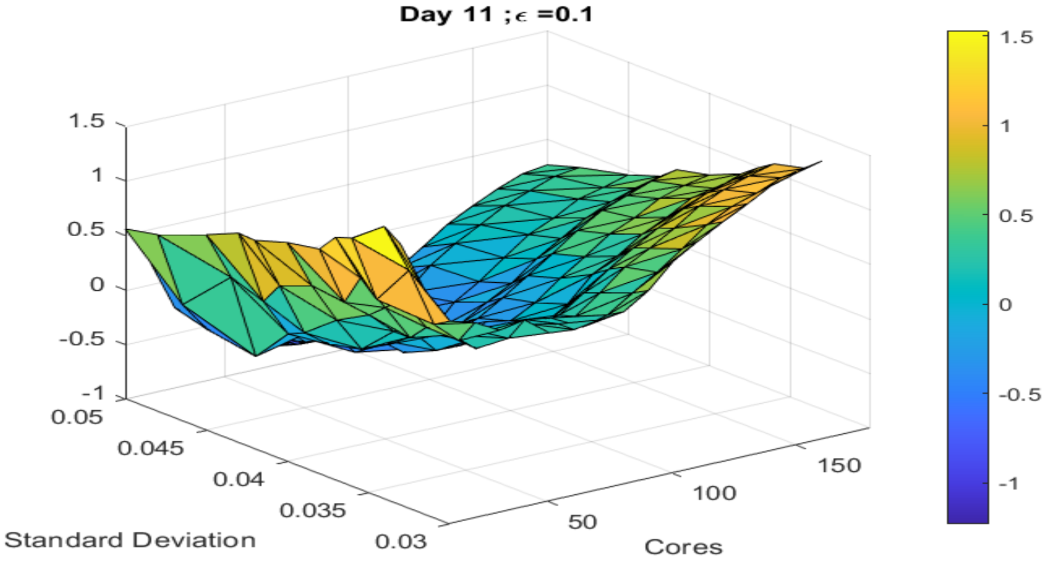


Figure 7.3: Heat Map Difference In Performance Ratio: Day 11, Slack:0.1

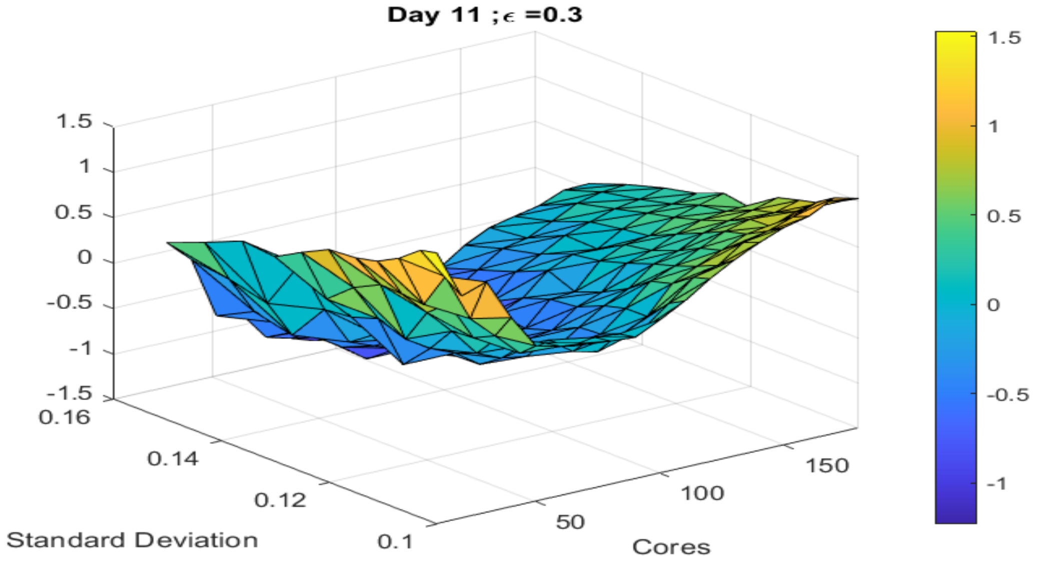


Figure 7.4: Heat Map Difference In Performance Ratio: Day 11, Slack:0.3

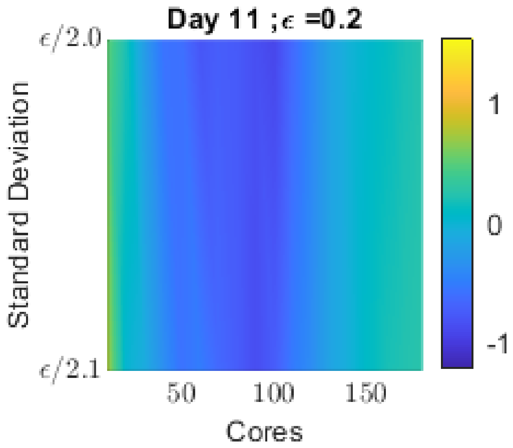


Figure 7.5: Heat Map Difference In Performance Ratio: Day 11, Slack:0.2

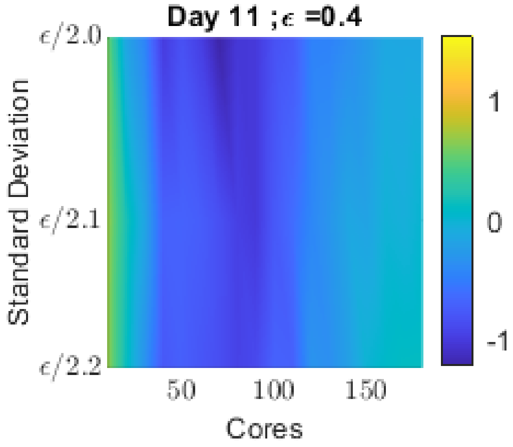


Figure 7.6: Heat Map Difference In Performance Ratio: Day 11, Slack:0.4

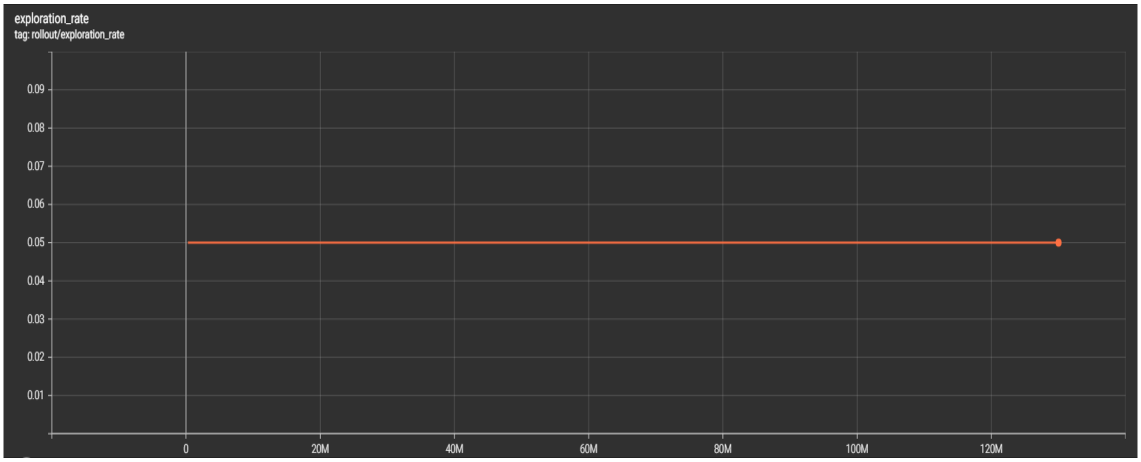


Figure 7.7: Training Exploration Rate

Chapter 8

CONCLUSION AND FUTURE SCOPE

The results show that our ML approach is not effectively competitive against an offline optimal algorithm in the average case. The results of the lower number of machines (resources) gives a slight hope that the algorithm may work better in worst case scenarios. The ML approach was designed to reject jobs based on their profitability to perform well in worse case scenarios. Hence, the algorithm rejects incoming jobs based on their profitability as it is always in anticipation of new jobs. This approach is at a disadvantage in an average case where ample amount of resources are available.

From the observations made we can conclude that the algorithm is not effective in average case scenarios and further testing needs to be done in tailor made worse cases for testing the algorithm's effectiveness. We could try training the learning environment in specific worst case scenarios too.

The scenario in which we find ourselves is quite unpredictable. Whenever a new job is provided, the scheduling decision will be heavily influenced by the current state of the schedule/machine as well as the nature of the new job (which may not follow a historical pattern). The proposed method employs a dense reward strategy, it attempts to map the current machine state and job locally, make a decision, and issue a reward at each step, without considering the larger picture. A sparse reward strategy, in which the full timeline is examined as a whole and incentives awarded proportionally at the end, is an alternative for the future.

Bibliography

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] Y. Chen, A. Huang, Z. Wang, I. Antonoglou, J. Schrittwieser, D. Silver, and N. de Freitas, "Bayesian optimization in alphago," *arXiv preprint arXiv:1812.06855*, 2018.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[4] S. Jamalabadi, C. Schwiegelshohn, and U. Schwiegelshohn, "Commitment and slack for online load maximization," in *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, 2020, pp. 339–348.

[5] P. Tassel, M. Gebser, and K. Schekotihin, "A reinforcement learning environment for job-shop scheduling," *arXiv preprint arXiv:2104.03760*, 2021.