Deep reinforcement learning for solving job shop scheduling problems

B.J. Meijer
S2648164
b.j.meijer-1@student.utwente.nl

Exam Committee
prof.Dr.-Ing. Sebastian Thiede (chairman)
dr.ir. Sipke Hoekstra (supervisor)
dr. Marcus Pereira Pessoa (external member)

Master thesis
Mechanical Engineering
Department of Design, Production and Management

UNIVERSITY OF TWENTE.

University of Twente
The Netherlands
August 2, 2024

Summary

This thesis explores integrating machine learning, specifically deep reinforcement learning, into job shop scheduling to improve production efficiency. The study assesses DRL models in both static and dynamic settings, revealing that DRL can surpass traditional methods in well-defined scenarios. However, it faces challenges with computational demands, scalability, and generalization. DRL shows promise, especially with tailored environments and transfer learning, but its practical use is limited to specific cases. Recommendations include continuing with traditional methods for now and exploring advanced techniques that require more expertise to enhance performance and generalization, while also speeding up the training process.

Contents

List of symbols

Job Shop Scheduling	
M	Machine
m	Total number of machines
k	Specific identification number of machine
J	Job
n	Total number of jobs
i	Specific identification number of job
O	Operation (part of the sequence of machines for a specific job)
j	Specific identification number of operation
p	Processing time
C	Makespan
T	Tardiness
S	Slack of job
d	Due date of job
a	Arrival time
Deep Learning	
O	Number of outputs in output layer
L	Number of layers
z	Output of a neuron after applying the activation function
f	Activation function
w_i	Weights associated with each input
x_i	Inputs to the neuron
b	Bias term
ϑ	Parameters of the value network
ϑ'	Parameters of the target network
y_t	Target Q-value
Q_{target}	Q-value estimated by the target network for the next state and action
Reinforcement Learning	
S	State
A	Action
R	Total reward
r	Reward per step
π	Policy
V	State Value Function
Q	Action Value Function
Hyperparameters	
α	Learning rate
γ	Discount factor
ϵ	Exploration rate in the ϵ-greedy method
ϵ_{init}	Initial exploration rate
ϵ_{min}	Minimum value of the exploration decay rate
ϵ_{decay}	Exploration decay rate
D	Replay buffer size
N	Number of episodes
L	Number of layers in the network
Environment	
s	Status of a job or machine (0 for non-existing/finished, 1 for active)
m	Machine ID in the job state matrix
t	Processing time on the next machine
o	Number of operations left to complete the job
c	Remaining time to complete the job
f	Finished status of the job (0 for not finished, 1 for finished)
a	Arrival time of the job
d	Due date of the job

w	Priority weight of the job
p	Number of operations completed by the machine
u	Utilization rate of the machine

List of abbreviations

General	
JSSP	Job Shop Scheduling Problem
DJSP	Dynamic Job Shop Scheduling Problem
AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
DL	Deep Learning
DRL	Deep Reinforcement Learning
RNG	Random Number Generation
Deep and Reinforcement Learning	
DQN	Deep Q-Network
D3QPN	Double Dueling Q-Network with Prioritized Experience Replay
PPO	Proximal Policy Optimization
MARL	Multi-Agent Reinforcement Learning
GNN	Graph Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
SGD	Stochastic Gradient Descent
RMSprop	Root Mean Square Propagation
Adam	Adaptive Moment Estimation
MDP	Markov Decision Process
MSE	Mean Squared Error
SARSA	State-Action-Reward-State-Action
PBIG	Population-Based Iterated Greedy
GRL-AS	Graph Reinforcement Learning with Auxiliary Strategy
Dispatching Rules	
SPT	Shortest Processing Time
LPT	Longest Processing Time
SRPT	Shortest Remaining Processing Time
LRPT	Longest Remaining Processing Time
STPT	Shortest Total Processing Time
LTPT	Longest Total Processing Time
FIFO	First In, First Out
LIFO	Last In, First Out
LOR	Least Operations Remaining
MOR	Most Operations Remaining
Others	
CT	Computational Time
UL	Upper Limit
LL	Lower Limit
Var	Variance
StD	Standard Deviation
MDP	Markov Decision Process
Experiment Results and Statistics	
UL	Upper Limit
LL	Lower Limit
Var	Variance
StD	Standard Deviation
CT	Computational Time
RS	Reward Scaling Factor
E	Number of Episodes

1 Introduction

In today's fast-paced world, efficiency is often a necessity. Over the past few decades, the global manufacturing sector has been transformed by technological advancements, significantly enhancing production processes. As the world's population continues to grow, so does the demand for products, highlighting the urgent need for innovative and efficient manufacturing solutions.

This thesis explores recent advancements in production planning, focusing particularly on the potential integration of machine learning (ML) into the industry. The primary aim is to investigate how machine learning techniques are applied to solve job shop scheduling problems (JSSP) and how these techniques can be effectively integrated. This work examines current developments and potential applications of machine learning, implementing a method to address job shop scheduling challenges. By evaluating different approaches to deploying machine learning, this thesis seeks to assess both its capabilities and limitations in solving these scheduling issues, with the goal of determining the practical applicability of these advancements in real-world manufacturing environments. This research was conducted under the guidance of the University of Twente.

1.1 The job shop scheduling problem

Job Shop Scheduling (JSSP) can be visualized as managing production processes within a company's production hall. In this context, jobs represent the products to be manufactured, and machines are the resources or workstations where these jobs are processed. Each job must pass through a sequence of machines, with each step called an operation, and the specific order of operations can vary from job to job. Additionally, each machine can handle only one job at a time. The challenge is to schedule these jobs on the machines in a way that optimizes the production process, considering various constraints and objectives.

JSSP is a challenging problem in manufacturing, focused on the efficient allocation of resources such as machines and operators, which is essential for optimizing production schedules. Without additional constraints or variables, a job shop is considered static. However, when constraints such as arrival times, machine breakdowns, and due dates are introduced, the job shop becomes dynamic, more accurately reflecting real-world manufacturing environments. This dynamic nature introduces significant complexity, making manual planning impractical and highlighting the need for advanced algorithms and optimization techniques.

1.1.1 Static job shop scheduling

The static job shop scheduling problem is characterized by a fixed set of jobs, machines, and operation times. To further elaborate, JSSP can be described as follows: a job shop environment contains several machines $M = \{M1, M2, \ldots, Mm\}$, where m denotes the number of machines. It also contains a number of jobs $J = \{J1, J2, \ldots, Ji, \ldots, Jn\}$, where n denotes the number of jobs. Each job, represented by Ji, consists of a series of operations $Oi = \{Oi1, Oi2, \ldots, Oij\}$ processed in a predefined sequence, represented by Oi, j where j denotes the number of operations. Each operation corresponds to a machine in M, to be processed with a given processing time pi, j. The goal is to schedule these operations to minimize the total completion time, known as the makespan C. Static JSSP does not entail any dynamic variables, such as arrival times, machine breakdowns, and/or due dates, which makes it a static problem.

The complexity of JSSP increases when the number of jobs, machines, and operations grows, making manual planning impractical and inefficient. This has resulted in researchers and practitioners developing various algorithms and optimization techniques to tackle JSSP, ranging from heuristic methods and mathematical models to advanced scheduling algorithms based on artificial intelligence (AI) using ML. By addressing JSSP effectively, manufacturing facilities can improve their productivity, reduce lead times, and enhance overall efficiency. Hence the importance of understanding and solving JSSP in research and application in the field.

1.1.2 Dynamic manufacturing environments in job shop scheduling

Dynamic Job Shop Scheduling (DJSP) introduces additional complexities by incorporating variables such as demand fluctuations, machine breakdowns, and unexpected delays. Unlike static JSSP, DJSP

must adapt to changes and uncertainties in real-time. The challenge lies in developing scheduling algorithms and decision-making strategies that can effectively handle these dynamic changes while optimizing production objectives, such as resource utilization and minimizing delays.

The DJSP tends to be a more realistic representation of real-world scheduling problems for scheduling jobs. The focus on JSSP is mostly on minimizing makespan, with little to no variables taken into account other than the machines, jobs, and their operations. DJSP focuses more on adapting the production schedules in a time-based environment, while optimizing the objective, for example resource utilization, delays, or customer demands.

The challenge of DJSP is in the development of scheduling algorithms and decision-making strategies that can effectively handle the dynamic changes while maintaining efficiency and meeting production objectives. Researchers have approached DJSP with AI, optimization algorithms, and heuristic methods, like JSSP, to try and solve the complex problems. By addressing these dynamic natures of the manufacturing environments, the solutions to DJSP can help to improve responsiveness and overall performance in real-world job shops. Hence, understanding and solving DJSP remains a crucial area of research and development in the field of production and planning scheduling.

1.1.3 General scheduling method and representation

In general, job scheduling involves assigning operations to machines, creating a schedule, and evaluating it for bottlenecks and planning gaps. Traditional scheduling methods may include heuristic approaches and algorithms to generate and refine solutions. As job shop sizes increase, human-like scheduling methods become less feasible, underscoring the need for advanced algorithms. Tools such as Gantt charts are commonly used to visually represent job sequences, machine assignments, and timelines, facilitating communication and coordination.

An outline of the flow of scheduling and improving upon the schedule is given in Figure 1. The inputs are the operation order of the jobs, in both the machine sequence and corresponding processing times. When starting scheduling, the operations are assigned to machines, starting at time 0 and planning each operation per job, creating a schedule. After the schedule is created, it is evaluated to determine bottlenecks and planning gaps. If these are found, the schedule is adjusted. After reiterating, if needed, and no more bottlenecks are found, the process is stopped, resulting in a best-found schedule. As can be imagined, when the size of the job shop increases, such a method will not suffice as humans might not be able to indicate the gaps or be able to reschedule over 1000 operations in an efficient manner. Hence the need for improved methods such as algorithms.

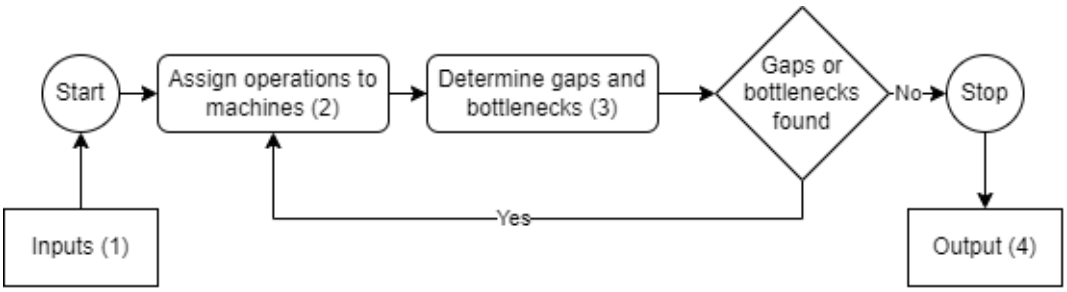


Figure 1: Flowchart of general scheduling in job shops

After scheduling, the Gantt chart is a common tool to create a visual representation that shows the sequence and timing of jobs on the different machines in the job shop, as shown in Figure 2. The Gantt chart has several benefits. It shows a clear visualization of the job sequences, machine assignments, and timeline, which is easy to understand. Also, the progress of each job is easily identified, showing delays and understanding the machine utilization. Finally, the clear schedule overview facilitates clear communication and coordination, for example, for team members.

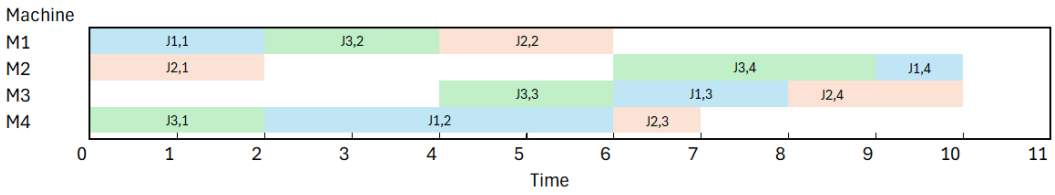


Figure 2: Example of Gantt chart. $J_{i,j}$ depicts the job i with operation j

1.2 Machine learning for job shop scheduling

At the current time, ML has been applied multiple times to solving both static and dynamic job shop scheduling problems with success. Research has shown that machine learning can indeed be applied to these problems in similar ways that other methods can be applied. However, the focus is on training the algorithm on a single problem at a time, thus creating a solver per problem. It is assumed that this solver is not able to solve other problems with comparable performance to traditional and heuristic methods. Hence it can already be concluded that in dynamic real-world environments these methods are not applicable yet.

1.3 Research scope

This research explores the integration of reinforcement learning (RL) and deep learning (DL) techniques in job shop scheduling, focusing on both static and dynamic environments. The objective is to evaluate how these techniques can potentially be implemented in real-world settings. Unlike previous studies that have addressed singular scheduling problems, this research aims to study how these methods can improve general performance over solving a single job shop scheduling problem, to assess the applicability in real-world settings.

1.4 Research goals

Solving these scheduling challenges can significantly enhance production efficiency, reduce costs through minimized downtime, and improve overall productivity by reducing the need for manual adjustments. This research seeks to answer the question: "How can machine learning be applied in real-world dynamic job shops?" The sub-questions guiding this research are:

1. Which machine learning methods are applicable to solve dynamic job shop scheduling?
2. How do machine learning methods compare to traditional methods?
3. How do machine learning methods scale with increasing job shop sizes?
4. How well do machine learning models generalize to new, unseen job shop scheduling scenarios?
5. In what way can these methods be applied to real-world problems?
6. What are potential drawbacks and benefits of applying machine learning to real-world problems?

To address these questions, this study proposes a deep reinforcement learning (DRL) model using a deep Q-learning neural network (DQN) implemented in MATLAB, which is a combination of the RL method Q-learning and DL's neural networks. The model will be trained on both static and dynamic job shop scheduling problems. The performance of the trained agents will be evaluated based on metrics such as machine utilization, makespan, tardiness, and slack. Although the research will not deploy these models in actual manufacturing environments, it will assess their feasibility through extensive testing and evaluation.

1.5 Thesis outline

The document is organized as follows: Chapter 2 defines ML techniques, specifically explaining RL and DL. Chapter 3 reviews existing research and identifies gaps. Chapter 4 outlines the methodology, including the chosen ML method, problem generation, data collection, and analysis. Chapter 5 describes the chosen machine learning technique. Chapter 6 explains the environment setup. Chapter 7 sets out the experimental setup, covering benchmarks, hyperparameter tuning, and testing. Chapter 8 presents experimental findings, Chapter 9 analyzes results and discusses implications and limitations, and Chapter 10 concludes the research with a summary and recommendations.

2 Analysis of Machine Learning Algorithms

Machine learning is a rapidly evolving field within AI, with big advancements in recent years. One of the most well-known advancements is by Google's company DeepMind, which created an AI that could play the complex game of Go, eventually able to defeat the world champion [1]. Such an achievement shows promise for other complex problems to be solved with machine learning. In this chapter, the different subsets of machine learning are given, to determine the applicable subsets to job shop scheduling. These applicable subsets are then elaborated on, explaining their general workings.

2.1 Subsets of machine learning

Machine learning encompasses different techniques and methodologies, each suited for different types of problems and data. The main subsets of machine learning are [2] [3]:

- **Supervised learning:** Supervised learning uses labeled data, where each input is associated with a corresponding target output. The goal is to learn a mapping from inputs to outputs based on example input-output pairs, which allows the algorithm to make predictions on new, unseen data. This is for example used to classify animal behavioural states from environmental features [4].
- **Unsupervised learning:** Unsupervised learning uses structures from unlabeled data and learning patterns. The algorithm explores data to discover hidden patterns, clusters or relationships without explicit guidance and supervision. This is for example used to solve jigsaw puzzles [5].
- **Deep learning (DL):** Deep Learning is a subset of machine learning which focuses on using neural networks with multiple layers, also called deep architectures. These deep neural networks are capable of learning complex patterns and representations directly from raw data, which enables breakthroughs in areas such as image recognition, natural language processing and speech recognition. DL is for example used in medical image analysis [6].
- **Reinforcement learning (RL):** Reinforcement Learning is a type of machine learning where an agent is deployed, which learns to make decisions based on interacting with an environment and receiving feedback in rewards and/or penalties. RL learns optimal policies that maximize the cumulative reward over time, making it well-suited for sequential decision-making tasks. This is for example used to compete with top-level players in different games, like StarCraft II [7].

This research will mainly focus on RL and DL, as these two subsets have demonstrated promises in addressing the challenges of JSSP and DJSP. RL and DL methodologies are well-suited for scheduling due to the ability to learn and adapt, without requiring explicit knowledge of optimal solutions. By receiving inputs the correct inputs, RL and DL algorithms can effectively learn to navigate and optimize scheduling decisions, which can result into a flexible and adaptive approach to solving JSSP and DJSP.

2.2 Training Neural Networks: Deep learning

The following information is based on the book "Deep Learning" by Ian Goodfellow [3]. Deep learning involves training neural networks to learn hierarchical representations of data through multiple layers. Each layer in a deep neural network learns increasingly abstract features from the input data, making deep learning particularly powerful for complex tasks.

Deep neural networks are termed "deep" because they consist of multiple layers, depicted in Figure 3, that process and transform data. These layers work together to automatically learn hierarchical representations of the data, enabling the network to identify intricate patterns and relationships.

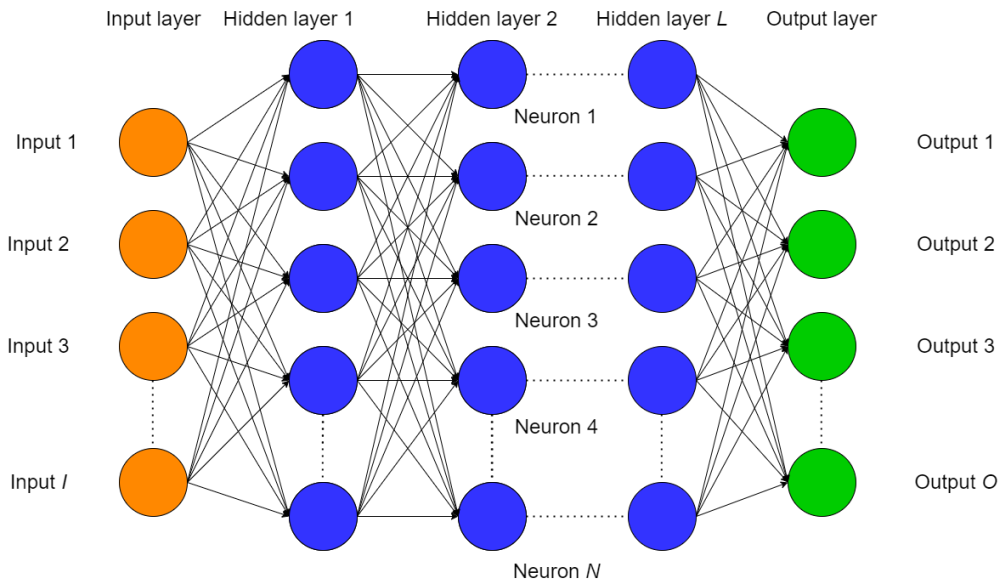


Figure 3: A general overview of a neural network

The key components of deep learning models are outlined in Table 3 and will be elaborated upon in the following subsections.

Component	Explanation
Neurons	Basic units that process input and produce output through activation functions.
Layers	Groups of neurons organized into input, hidden, and output layers.
Weights	Parameters that adjust the strength of connections between neurons.
Biases	Parameters added to neuron inputs to allow the model to better fit the data.
Activation Functions	Functions that introduce non-linearity into the model.
Loss Functions	Metrics that evaluate the model's performance by comparing predictions to actual outcomes.
Backpropagation	Algorithm for updating weights based on gradients to minimize the loss function.
Optimization Algorithms	Techniques used to adjust weights and minimize loss.
Regularization Techniques	Methods used to prevent overfitting and improve generalization.

Table 3: Components of deep learning with brief explanation

2.2.1 Neurons, layers, weights and biases

In a neural network, the learning process begins with the input layer, where data enters the network. Each subsequent layer in the network performs transformations on this data, and two fundamental components facilitate these transformations: weights and biases. The weights are parameters that modulate the strength of connections between neurons. They determine how the input data is transformed as it moves through the network. Each connection between neurons has an associated weight that scales the input received by the neuron. Biases are additional parameters added to the weighted sum of inputs to help the model fit the data more effectively. They allow the neuron to shift, which enhances the network's capacity to model complex relationships. In Eq. 1 the total weighted sum, or output of the neuron, z is given, where f the activation function applied to the sum of the weights and bias, w_i are the weights associated with each input x, x_i are the inputs to the neuron, n is the number of inputs to the neuron and b is the bias term.

$$z = f(\sum_{i=1}^{n} w_i x_i + b) \tag{1}$$

As the data moves through the network, each layer between the input layer and output layer, called hidden layers, process the inputs given. The initial hidden layer processes the raw inputs, to identify low-level features, such as edges or colors. Subsequent hidden layers build on these features to detect more complex patterns like shapes or textures. Finally, the output layer synthesizes these learned features to produce predictions or classifications based on the high-level patterns identified.

Each layer's learning process involves adjusting the weights and biases through backpropagation. This iterative process allows the network to refine its feature representations and improve its performance over time.

Overall, the interplay between weights and biases at each layer enables neural networks to learn and represent intricate patterns in data, leading to accurate and effective predictions.

2.2.2 Activation Functions in Neural Networks

Activation functions are essential for deep learning models, as they are responsible for introducing non-linearity into the neural network. While linear models can only represent linear relationships between input and output, neural networks need to model non-linear relationships to understand complex patterns. Activation functions allow neural networks to approximate and learn these non-linear mappings.

In addition to enabling non-linearity, activation functions play a crucial role in the backpropagation process, which is used to train the network. During training, the network needs to update its weights to reduce prediction errors. This update process relies on gradients, which are numerical values that indicate how much and in which direction to adjust the weights. Differentiable activation functions produce smooth gradients, ensuring that these adjustments are effective and stable, which helps in optimizing the network efficiently.

Common activation functions, such as Sigmoid, Hyperbolic Tangent, Rectified Linear Unit (ReLU), Leaky ReLU and Softmax, are explained in Appendix B.1.

2.2.3 Measuring performance: loss functions

Loss functions, also known as cost functions, measure how well the network's predictions match the actual target values. They quantify the discrepancy, or error, between predictions and targets. This error guides the model's learning process.

During training, the neural network makes predictions based on its current weights and biases. The loss function evaluates these predictions against the true target values, producing an error value that reflects the difference between the predictions and the actual outcomes.

As the model iteratively minimizes this error, it progressively improves its predictions. The process continues until the loss function reaches an acceptable level, indicating that the network has effectively learned to approximate the target values.

Two common loss functions, Mean Squared Error (MSE) and the Binary Cross-Entropy Loss, are given in Appendix B.2.

2.2.4 Improving model accuracy: understanding backpropagation

Backpropagation is a fundamental algorithm in deep learning designed to optimize the training of neural networks. Its primary purpose is to update the network's weights to minimize the difference between predicted and actual outputs, as indicated by the loss function. This process is critical for the network to learn from data and improve its performance over time.

Training a neural network using backpropagation involves two main phases: the forward pass and the backward pass.

- **Forward Pass:** During this phase, input data is fed into the network and propagates through each layer until it reaches the output layer. Each neuron processes the input and produces an output. This series of computations results in the network's predictions. The loss function then evaluates these predictions by comparing them to the actual target values, calculating the discrepancy or error.

- **Backward Pass:** In this phase, backpropagation calculates the gradient of the loss function with respect to each weight in the network. Using the chain rule of calculus, it determines how small changes in each weight affect the loss. The error is propagated backward from the output layer towards the input layer, and gradients are computed for each weight. These gradients indicate the direction and magnitude of the required adjustments to reduce the loss. By adjusting the weights based on these gradients, the network learns to minimize the error.

For example, if a neural network predicts "cat" with a value of 0.2 but the correct label is "dog" (which should be 1), the error is 0.8. Backpropagation uses this error to adjust the network's weights. It calculates how each weight contributed to this error, propagates the error backward through the network, and updates the weights accordingly. This iterative process enhances the network's accuracy and overall performance.

2.2.5 Fine-tuning neural networks: optimization algorithms

Optimization Algorithms are crucial in training neural networks as they are responsible for finding the optimal set of weights that minimize the loss function, thereby enhancing the model's performance. These algorithms are mathematical techniques used to update the weights of the network during training, with the aim of improving accuracy and efficiency.

Common optimization algorithms include Stochastic Gradient Descent, Adam, and RMSprop, each of which has its own approach to handling weight updates:

- **Stochastic Gradient Descent (SGD):** SGD updates weights based on the gradients of the loss function computed from individual or small batches of training samples. It is straightforward and computationally efficient but can be sensitive to the choice of learning rate and may get stuck in local minima. this means that the algorithm has converged to a point where the loss is lower than in the surrounding area but not the lowest possible point overall. This can lead to suboptimal model performance and reduced generalization.

- **RMSprop (Root Mean Square Propagation):** RMSprop addresses the issue of varying learning rates by normalizing the gradient updates. It computes an exponentially weighted moving average of the squared gradients, which helps to adapt the learning rate for each parameter, ensuring stable and efficient training.

- **Adam (Adaptive Moment Estimation):** Adam combines the advantages of two other extensions of SGD: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSprop). It maintains a moving average of both the gradients and their squared values, which helps adjust the learning rate for each parameter dynamically. This typically leads to faster convergence and better handling of noisy data.

The goal of these optimization algorithms is to accelerate convergence towards a minimum of the loss function while avoiding local minima and improving training efficiency. Each algorithm has unique strategies for adjusting the learning rate and handling different training conditions, making them suitable for various types of neural networks and data distributions. Each of these algorithms will be tested in this research. These methods are further elaborated on in Appendix B.3.

2.2.6 Preventing overfitting: regularization techniques

Regularization Techniques are essential for preventing overfitting in neural networks and ensuring that the model generalizes effectively to new, unseen data. Overfitting occurs when a model performs well on training data but poorly on new data, often due to the model being too complex or fitting noise in the training data. Regularization methods help mitigate this by adding constraints or modifications during training.

Common regularization techniques include:

- **L1 Regularization:** Adds a penalty proportional to the absolute values of the weights. This encourages sparsity, meaning it drives some weights to exactly zero, effectively performing feature selection and reducing model complexity.

- **L2 Regularization:** Adds a penalty proportional to the squared values of the weights. This discourages large weights and helps distribute the weight values more evenly, contributing to a smoother, more generalized model.
- **Dropout:** Involves randomly "dropping out" a subset of neurons during training. This prevents neurons from co-adapting too closely, which helps the network learn more robust features and reduces overfitting.
- **Batch Normalization:** Normalizes the inputs of each layer by adjusting and scaling activations. This technique stabilizes and accelerates training by reducing sensitivity to initial weights and improving convergence rates.

By incorporating these regularization techniques, neural networks become more robust and better equipped to generalize to new data. This helps ensure that the model performs well not only on the training set but also on unseen examples, ultimately leading to improved performance in real-world applications. These techniques are further explained in Appendix B.4.

2.2.7 Flowchart of training neural networks

The training of neural networks is organized into epochs, with each epoch involving running the entire dataset through the network. Typically, data is processed in smaller batches within each epoch. Using mini-batches balances memory constraints, improves computational efficiency, accelerates convergence, and can enhance generalization. Figure 4 illustrates the training process of a neural network for one iteration.

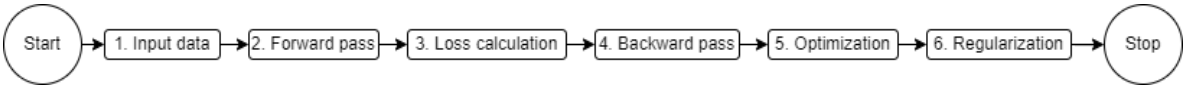


Figure 4: Flowchart of training a neural network

This process is repeated for each batch of data across multiple epochs, iteratively improving the network's performance.

2.3 Reward-based learning: Reinforcement Learning

The information on RL is based on the book "Reinforcement Learning: An Introduction" by Richard Sutton and Andrew Barto [8]. Reinforcement Learning focuses on training an agent (learner or decision-maker) to make decisions within an environment to maximize rewards. RL revolves around learning from the consequences of actions by exploring and exploiting the environment to learn an optimal policy. The general workings of RL are depicted in Figure 5.

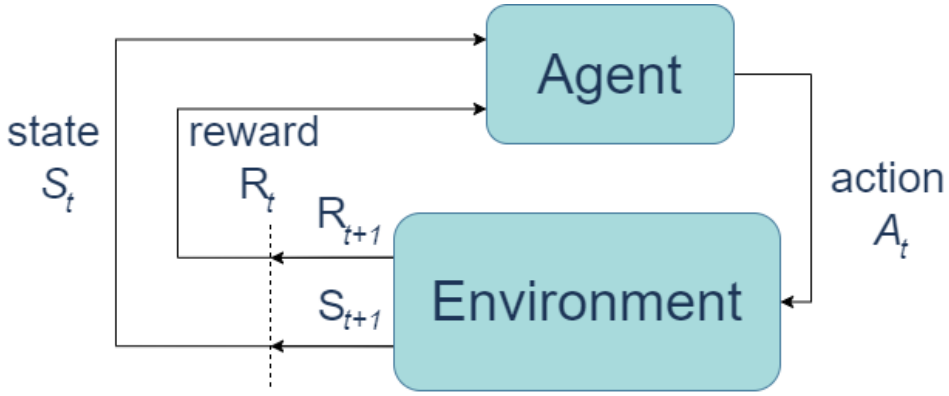


Figure 5: The interaction between an agent and environment in RL

An important difference to note for RL is that it uses episodes, different from epochs in DL, which refers to a complete sequence of states, actions, and rewards that ends in a terminal state. It starts

from an initial state and continues until a predefined condition is met, such as reaching a goal state or a time limit. Each episode represents one instance of the agent interacting with the environment.

2.3.1 The environment: enabling the exploration of the problem

At the core of RL is the environment, usually modeled through Markov Decision Processes (MDPs). MDPs define the interaction between an agent and its surroundings, specifying states (S), actions (A), and rewards (R). These elements are essential for formalizing the decision-making process and enabling the agent to learn optimal policies to maximize rewards. They are represented by a tuple (S, A, R).

Balancing exploration and exploitation is crucial to ensure the agent experiences different states without endlessly traversing the environment. Exploration involves trying new actions to discover their effects and potential rewards, while exploitation uses known actions that yield high rewards. A common approach to manage this balance is the ϵ-greedy method. Here, ϵ represents the probability of choosing a random action, ensuring exploration, while (1-ϵ) represents the likelihood of exploiting the agent's current knowledge. High ϵ values encourage exploration, and as learning progresses, reducing ϵ shifts the focus towards exploitation. An example is using linear decay of ϵ, which ensures a smooth transition from exploration to exploitation as the agent refines its policy based on gained experiences.

2.3.2 Model-based and model-free approaches

RL methods can be categorized into two types: model-based and model-free approaches.
Model-based methods use a model of the environment to simulate and plan actions. The agent creates a model of transition probabilities and rewards, allowing it to predict outcomes and make decisions. Examples include Dynamic Programming (DP) methods like value iteration and policy iteration, which rely on an accurate model to compute optimal policies. For instance, navigating a car to a candy shop involves creating a map (model) of the town (environment), and then deciding the best route to reach the shop.
Model-Free methods learn value functions or policies directly from interactions with the environment without requiring a model. These methods learn from experiences via trial and error. Examples include Q-learning and SARSA (State-Action-Reward-State-Action). Model-free methods are more flexible since they don't need an environment model, making them widely applicable. Imagine driving a car to a candy shop without a map, relying solely on memory and past experiences to find the way.

2.3.3 The learned policy and value functions

In RL, a policy (π) is a strategy that the agent uses to determine the next action based on the current state. Policies can be deterministic or stochastic:

- **Deterministic Policy** ($\pi(s) = a$)**:** Chooses the best action for a state with certainty.
- **Stochastic Policy** $\pi(a \mid s) = P(A_t = a \mid S_t = s)$**:** Selects an action based on a probability distribution over actions for each state.

To make informed decisions, RL methods use value functions, which estimate the expected cumulative reward:

- **State Value Functions (V(s)):** Estimates the expected cumulative reward starting from state s following a certain policy.
- **Action Value Function (Q(s, a)):** Estimates the expected cumulative reward starting from state s, taking action a, and following a certain policy.

Value functions help RL agents evaluate the long-term benefits of different actions and states, enabling informed decision-making to maximize cumulative rewards. Examples of RL algorithms that use value functions include Q-learning, SARSA, and policy iteration.

Value predictions are refined using loss functions, which measure the difference between predicted and actual rewards. Common loss functions used are Mean Squared Error (MSE) for value prediction and Cross-Entropy Loss for classification tasks. Optimization algorithms like Stochastic Gradient Descent (SGD), Adam, and RMSprop update the weights during training, ensuring efficient learning and convergence. The loss function and optimization algorithms are detailed earlier in Chapter 2.2.

2.4 The hybrid approach: Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines the strengths of Reinforcement Learning and Deep Learning, creating a framework for solving complex, high-dimensional problems. This synergy leverages RL's ability to learn optimal policies through interactions with the environment and DL's capacity to handle raw, high-dimensional input data.

Combining RL and DL is useful, as learning from raw data can be useful as traditional RL methods often struggle with high-dimensional input spaces, like images. DL's deep neural networks are able to extract meaningful features from raw data, allowing DRL models to learn effective representations of states, actions and rewards without manual feature engineering. Secondly, DL models can generalize across different tasks and environments due to the capacity to learn complex patterns and representations. Hence the combination is very flexible. Finally, DRL models can approximate value functions and policies more effectively than traditional RL methods, enhancing the decision-making process and enabling the agent to handle more sophisticated tasks.

DRL applies deep neural networks as function approximators for value functions, policies, or both. The process involves:

- **State Representation:**
  - The environment provides the agent with raw data (e.g., images, sensor readings).
  - A deep neural network processes this data to extract meaningful features, creating a high-dimensional representation of the current state.
- **Action Selection:**
  - Based on the state representation, the agent selects an action using a policy network.
  - The policy can be deterministic or stochastic, depending on the algorithm used.
- **Reward Processing:**
  - After taking an action, the agent receives a reward from the environment.
  - This reward signals how well the action performed concerning achieving the goal.
- **Learning and Updating:**
  - The agent uses the reward to update its policy or value function through backpropagation.
  - Optimization algorithms such as Stochastic Gradient Descent (SGD), Adam, or RMSprop are used to adjust the neural network weights.

By combining RL's interaction-based learning with DL's capability to handle complex data, DRL provides a robust framework for developing intelligent agents capable of solving intricate decision-making problems.

2.5 Summary

As job shop scheduling is a complex problem, methods such as deep learning, reinforcement learning, and a hybrid approach of deep reinforcement learning are well-suited to tackle these challenges. **Deep learning** is adept at learning from datasets, extracting complex relationships and patterns through neural networks with multiple layers. This capability allows DL to automatically discern intricate features from raw data, making it effective in understanding and predicting scheduling requirements. **Reinforcement learning** could frame the scheduling problem as an environment where an agent learns iteratively from its interactions. By making decisions, such as planning an operation, and learning from the consequences (rewards and penalties), RL develops an optimal policy over time. The hybrid method of **deep reinforcement learning** combines the strengths of DL and RL. DRL uses deep neural networks to process high-dimensional data and extract meaningful features, while simultaneously leveraging RL's framework to traverse and learn from the environment. This synergy enables the agent to efficiently learn policies or value functions, making DRL a robust approach for solving job shop scheduling problems. In summary, DL, RL, and DRL show significant promise in addressing complexities similar to that of job shop scheduling. DL's ability to extract patterns from

data, RL's iterative learning in environments, and DRL's combined strengths make these methods powerful tools for exploring and solving job shop scheduling challenges. The next chapter will conduct a literature review to assess current research and identify research gaps, guiding future studies in this domain.

3 Literature Review

Job shop scheduling is a problem widely researched for many years. Currently, more and more of this research transitions to using machine learning to try and solve and/or optimize the job shop scheduling. Over the last couple of years, a spike in articles and papers published about reinforcement learning is seen, with papers about dynamic scheduling of multiple deadline constrained tasks in a serving system [9], dynamic scheduling for multi-level air defense with contingency situations [10], Scheduling for the Flexible Job-Shop Problem with a Dynamic Number of Machines [11] and Dynamic Resource Allocation in Wireless MEC Networks [12] just to name a few recently published researches. All these papers have the same focus: solve a scheduling problem with machine learning. In this chapter, standardized problems will be discussed, different approaches to solving job shop scheduling problems will be looked at, being traditional methods, heuristic approaches and reinforcement learning methods.

3.1 Standardized problems

For static job shop scheduling problems, well-known problems, called instances, have been created over time. These instances all have the same layout, creating jobs with an equal amount of operations to the number of machines and having a processing time per operation. These instances known are created by Adams, Balas and Zawack [13], Demirkol, Mehta, and Uzsoy [14], Fisher and Thompson [15], Lawrence [16], Applegate and Cook [17], Storer, Wu and Vaccari [18], Taillard [19] and Yamada and Nakano [20]. Together, they create a strong library of 242 job shop scheduling problems, with a big range of different numbers of jobs and machines as well as having different ranges of processing times. These are widely used in the research mentioned in the upcoming parts of the chapter. To elaborate on how these instances are created, the research by Taillard is examined. First, the instances created by Lawrence and by Fisher and Thompson are mentioned. Based on these instances and found results at that time, it is concluded that instances up to ten machines can be solved satisfactory. Hence, Taillard proposes instances with at least 15 machines and 15 jobs, up to 20 machines and 100 jobs. The problems are created with a uniform distribution for the processing times, between 1 and 99, like the instances by Lawrence. By using random number generation (RNG), these problems are generated and can be reproduced. Finally, using Tabu Search the instances are tested to create a lower and upper bound for the found makespan. These are created with the goal to create a comparison base for future resolution methods. A representation of the job shop problems is given in Table 4, where the rows are the jobs, the uneven columns depict the machine of the specific operation and the even columns the processing time of that operation on that machine.

2	1	0	3	1	6	3	7	5	3	4	6
1	8	2	5	4	10	5	10	0	10	3	4
2	5	3	4	5	8	0	9	1	1	4	7
1	5	0	5	2	5	3	3	4	8	5	9
2	9	1	3	4	5	5	4	0	3	3	1
1	3	3	3	5	9	0	10	4	4	2	1

Table 4: Example of job shop instance (6 jobs, 6 machines) from Fisher and Thompson [15]

3.2 Traditional methods

At the start of the research into scheduling problems, traditional methods have been developed and proposed as solutions. In the context of job shop scheduling, traditional methods are the classical approach to solving scheduling problems. Traditional methods typically are algorithmic, naturally relying on mathematical optimization principles or heuristic rules to find (near-)optimal schedules for manufacturing processes. These traditional methods for JSSP can be split into two groups, exact algorithms and dispatching rules. Well-known exact algorithms are branch and bound, mixed-integer linear programming (MILP) and constraint programming (CP).

Exact algorithms in the context of job shop scheduling are designed in such a way that all possible solutions are systematically explored within the solution space, to find the optimal solution [21]. Unlike

heuristic or approximation methods, which aim to find good solutions within a reasonable time frame, exact algorithms guarantee that the found solution provides the best possible solution, according to the defined criteria. These methods are often computationally intensive and may become impractical for very large or complex problems.

Exact algorithms

Branch and bound is an algorithm which is well-explored in job shop scheduling. The method explores branches of a decision tree and prunes those that do not lead to feasible solutions [22]. From different researches it is found that branch and bound is a commonly explored algorithm in (flexible) job shop scheduling [23] [24] [17] [25] as well as being used for scheduling trains in a railway network [26]. For the job shop scheduling, it is seen that the exact algorithms improve upon other methods, within a reasonable computational time of about 19 minutes for a 10 jobs by 10 machines job shop, for example. However, it is found that for bigger job shops the methods processing time increases and the effectiveness decreases. This shows that this method is well suited for solving smaller problems, while not being effective for bigger job shop scheduling problems.

MILP makes the scheduling problem as a set of linear equations and inequalities, solving them to find optimal schedules [27]. The method is a robust and precise method, making it a suitable method for different scheduling contexts, while being impractical for very large problems. Surgical case scheduling [28], flexible job shop scheduling [29] [30] [31] and scheduling in a container terminal [32] are usage examples from research. From these papers about flexible job shop scheduling, it is found that small flexible job shops (starting from 2 jobs, 2 machines and 2 operations per job) can be solved with an optimal solution within seconds. However, the computational time increases from seconds to nearly an hour, with sometimes not even finding an optimal solution, when the size of is increased (starting from 8 machines, 4 jobs and 7 operations per job). It is seen that overtime better models are developed that decrease the computational time while increasing the performance. CP is mainly found to be used for flexible job shops.

CP is a method using constraints to reduce the search space, focusing on the feasible solutions. This method is effective for handling complex constraints and has been applied in numerous scheduling applications to enhance efficiency and feasibility [33]. The models specify the constraints that need to be satisfied rather than being an objective function to be optimized, making it a flexible and powerful tool for solving combinatorial problems. CP is for example used to solve vehicle routing problems [34] and (flexible) job shop scheduling [30] [35]. From [35] it is found that benchmark instances from Taillard [19] can be solved with a combinatorial approach of using local search with CP. This method finds six new best makespan results. The maximum computational time for each experiment done, solving a single problem, in this research is set to 3600 seconds, or one hour. Other hybrid approaches are used for static job shop scheduling, for example, using supervised learning to improve CP [36].

Dispatching rules

Dispatching rules are used to make real-time decisions within scheduling environments. These rules, also known as sequencing rules, are heuristic approaches aimed at determining the order in which jobs are processed on machines [37]. Unlike exact algorithms, which aim to find an optimal solution by exploring all options, dispatching rules provide quick and often effective scheduling decisions based on predefined criteria. Some examples of dispatching rules are:

- **First in, first out (FIFO):** This rules is used to prioritize jobs based on their arrival time, where the job that entered first is processed first. FIFO is easy to implement and simple in use, making it suitable for scenarios where job arrival times play an important role.
- **Shortest processing time (SPT):** SPT prioritizes jobs with the shortest processing times, aiming to minimize the total processing time of jobs in the system. By selecting jobs with shorter processing times, SPT can help reduce job waiting times while enhancing overall system efficiency.
- **Earliest due date (EDD):** EDD selects jobs based on the due date, prioritizing jobs with the earliest deadlines. The rule is particularly useful for situations where meeting deadlines is critical, as it ensures timely completion of high-priority tasks.

Dispatching rules offer simplicity and efficiency in scheduling decisions. Thus, research is done into exploring and defining new dispatching rules with the availability of computational power [38]. Next to creating new rules, rules are combined to create multiple priorities into singular rules as well as getting different results with those rules [39]. Most importantly for this research, the dispatching rules are widely used in research involving machine learning [40] [41]. These machine learning approaches use multiple dispatching rules, so that the model can use different rules, defined as actions, at different moments, or states, thus being able to be flexible with the way the jobs are being scheduled, while maintaining a simple way of scheduling.

3.3 Heuristic methods

Heuristic methods, unlike exact algorithms, are focused on finding good, not necessarily perfect, solutions within a set timeframe. These methods prioritize getting things done efficiently on a large scale, rather than finding the optimal solution without considering time. Some of the well-known techniques include Genetic Algorithm (GA), Simulated Annealing (SA), Tabu Search, Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and simple greedy algorithms.

Genetic algorithms solve problems by mimicking the process of evolution. GA starts by creating a population of potential solutions, called chromosomes. Through the process of selecting the best suited solutions, using crossover, which mixes parts of good solutions, as well as mutation, which randomly changes solutions to maintain exploration, GA aims to find better solutions over time [42]. Genetic algorithms have been used to solve both flexible job shop problems [43] [44] as well as job shop scheduling problems [45] [46] [47]. Older research [45] shows that these genetic algorithms can solve Muth-Thompson problems (in this paper known as Fisher-Thompson [15]) of 10 by 10 in in 135 seconds, while a 20 by 5 problem is solved in 147 seconds, measured on a Sun10 workstation, which is a very old desktop. Slightly more recent research [47] shows comparison to different researches on both Fisher-Thompson [15] as Lawrence [16] instances. The computational results show that the algorithm is able to find optimal or near-optimal solutions for every of the 43 tested instances, with a range of about a minute of computational time for smaller problems (10 by 5), while taking about an hour to solve bigger problems (30 by 10).

SA is based on annealing found in metallurgy. The process involves heating and cooling to make metal stronger. SA applies an idea similar to that process for problem-solving. It starts of with a solution, allows for some "heating", meaning it accepts worse solutions early on. Over time, the cooling down starts, where the defects of the metal are removed, where the method becomes more selective to avoid getting stuck in local optima [48]. Different researches use SA for job shop scheduling [49] [50] [51] [52]. The second mentioned research [50] uses both Fisher-Thompson [15] as well as Lawrence [16] instances to test the proposed SA method. As this research is older, it only shows that the SA is promising in terms of found solutions in comparison to other research [13] [49], it does not compete in terms of computational time in comparison to Adams [13]. For example, the 6 by 6 instance is solved in 52 seconds by Van Laarhoven [50] but in 1 second by Adams [13]. More recent research [52] shows improvements upon the earlier proposed SA method of Van Laarhoven [50]. However, it does not state computational times.

Tabu search is a method that equals exploring a maze, while keeping track of where you have already been. This memory structure is used to avoid revisiting the same solutions. By using this tabu list of forbidden moves, it efficiently guides through complex solution spaces, finding high-quality solutions [53]. In [54] [55] [56] Tabu search is researched to solve job shop scheduling problems.

ACO mimics behavior of ant, which needs to find the shortest path to food by leaving pheromone trails. ACO creates artificial ants which build solutions by following pheromone trails. These trails are updated based on the quality of the found solutions. ACO is especially useful for problems involving finding optimal routes, like scheduling problems [57]. Different researches of job shop scheduling using [58] [59] [60] have been done. Earlier research [58] shows that GA performs better than ACO. Only [60] discuss the ACO for JSSP problems, testing on known instances. The paper shows that the proposed ACO method solves different instances within smaller timeframe in comparison to other methods, like GA and PSO, while also outperforming them. For example, the FT10 instance is solved optimally,

within 14.5 seconds.

PSO mimics the behavior of groups of particles, moving through a solution space, each particle adjusting its position based on its own experience as well as their neighbors. PSO optimizes found solutions by simulating this movement, with particles adjusting their positions until a satisfactory solution is found [61]. Different researches into using PSO for job shop scheduling have been executed [62] [63] [64]. First, [62] shows three different PSO approaches, where two are different PSO methods and one is a hybrid PSO with Tabu search. Based on 43 different benchmark instances of Fisher-Thompson [15] and Lawrence [16], the hybrid approach finds 41 of the 43 best known solutions, showing promise for PSO as a solver for JSSP. With this, it also shows that this method needs little time to solve these problems, for example needing 157 seconds to solve FT10. Another research [63] shows another approach, focusing on both minimizing makespan and minimizing tardiness, called multi-objective PSO. However, due to the addition of tardiness there is no real comparison to other research. Finally, Lin [64] shows another improvement using a hybrid PSO to solve benchmark instances, without mentioning the computational time per problem.

Finally, simple greedy algorithms make decisions based on immediate benefits without considering the future consequences. The best option at each step is chosen, hoping for it to lead to a satisfactory solution overall. While simple, it does not always find the best solution for certain types of problems [65]. Only two researches [66] [67] have been found that focus on using greedy algorithms to solve job shop scheduling. Only [66] shows real comparable data using the known benchmark instances. The research shows that the proposed population-based iterated greedy method (PBIG) has low computational time, with the highest found of 540 seconds for a 30 by 10 instance, while also being able to solve them finding values near the best known solution.

To summarize, these methods all show promise in solving benchmark instances. However, the focus is on solving static job shop scheduling problems, with little to no research about dynamic job shop scheduling. A comparison of the methods found that show both the best found makespan and computational time for certain instances is given in Table 43 in Appendix C.

3.4 Reinforcement and deep learning methods

There is significant less research into using AI, mainly reinforcement learning and deep learning, to solve job shop scheduling. This shows that AI is a newer approach to the job shop scheduling problem. Methods used are both RL and DL techniques, as well as combinations of traditional or heuristic methods with RL/DL. RL and DL techniques used to solve static as well as dynamic job shop scheduling are (deep) Multi-Agent Reinforcement Learning (MARL) [68] [69] [70], Deep Q-learning Neural network (DQN) [71] [72], double dueling Q-network with prioritized experience replay (D3QPN) [73], PPO with deep learning [74] [75], MARL combined with PPO [76], self-learning discrete salp swarm algorithm based on DRL [77], spatial pyramid pooling-based DRL [78], combining RL with GNN [79] [80] and graph reinforcement learning with auxiliary strategy (GRL-AS) [81].

It is found that a lot of different methods are used in RL and DL to solve JSSP and DJSP. These methods are deployed for both static as dynamic job shop scheduling problems. Like the aforementioned methods, the approach is on solving one problem at a time with the specific technique. It shows that RL and DL methods offer potential to learn scheduling policies directly from the data, can adapt to changing environments learning from experience and can handle high-dimensional state and action spaces efficiently. This is shown by some proposed methods being able to find optimal solutions for many of the standardized problems from Chapter 3.1. However, due to the complexity of DL and RL methods, they may require more computational resources and time, and need a well designed environment to be able to work effectively, while ensuring robustness and the ability to generalize of learned policies on different problems.

For this research the most important part is using a method that is relatively simple to implement, while being able to assess the scalability and limitations of RL and DL for JSSP. Hence, the focus of this part of the literature review will be about the definitions of DJSP as well as the approach and build-up of the models and the problems.

3.4.1 Definitions of dynamic job shop scheduling

To understand the way these methods are deployed, the definition of the dynamic job shop scheduling problem is important. Three different definitions can be found. The first uses (random) arrival times and due dates, among others, to define the environment as dynamic. Second, the environment itself adds events that occur, such as machine breakdowns and material shortage, to define a dynamic job shop. The third representation found is a combination of the first two representations. The first definition is a simpler representation of DJSP, as these attributes can be predefined in a job shop scheduling problem. The second one can not be predefined and is an environment specific attribute, creating randomness in the job shop scheduling problem. Both of these are considered dynamic, and occur in real-world problems.

3.4.2 Processing the problems: representation, scheduling and performance metrics

In job shop scheduling, different representations are used to provide input for solving the problems. While more complex methods, such as multi-agent reinforcement learning, involve multiple agents processing data simultaneously, the focus is on more straightforward input representations. Generally, there are two main types of state representations, the first being custom vectors or matrices to facilitate data to the agents, relevant for their decision-making, usually depicting information about the jobs and machines. Here, the data is both relevant and manageable, allowing the agents to make informed decisions based on job and machine information.

The other representation of inputs is a disjunctive graph, depicted in Figure 6. Disjunctive graphs visualize the order of tasks and dependencies in a job shop scheduling problem. The dashed lines in Figure 6 show the dependency, where the arrows represent conjuctive edges and the dashed lines represent disjunctive edges. The arrows are used to show operations that need to be planned in that order, while the dashed lines represent tasks that can be done in any order. To use the disjunctive graph in RL, transformers are used to transform the disjunctive graph into input data for the environment and agent, being the state. Hence, it is a clear representation but complex in use.

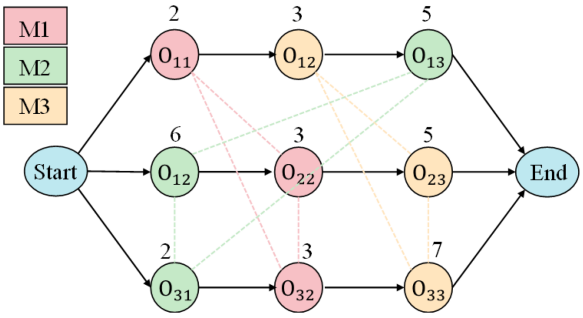


Figure 6: Representation of a disjunctive graph [73]

To schedule the jobs, different types of actions are defined. The first method is simply choosing the next job to plan based on the state, which is common when using a disjunctive graph. This is done by selecting a job from the pool of pending jobs and determining its sequence in the schedule. Second, dispatching rules are used to give the agent options to plan different jobs. The agent learns to mimic or improve upon the dispatching rules to make informed decisions.

Finally, the performance metrics used also differ to that of static job shop scheduling, as well as per research. Mostly used in dynamic job shop scheduling is the lateness of a job, or tardiness (T_i), given in Eq. 2, where C_i is the completion time of job i and d_i is the due date of job i.

$$T_i = \max(0, C_i - d_i) \tag{2}$$

Next to tardiness, a common metric is earliness, or slack (S_j) given in Eq. 3, where S represents the total slack of job j, d the due date of job j and C the makespan of job j.

$$S_j = \max(0, d_j - C_j) \tag{3}$$

Other used metrics are the makespan, machine utilization rate and idle time.

3.5 Conclusions and research gap

It is found that from the traditional methods, the exact algorithms work well for small problems. However, when increasing the size of a job shop, the performance decreases. Dispatching rules tackle the scalability problem by suggesting simple and effective rules which can be applied for all sorts of job shops. While being very efficient, the performance can be low, not finding optimal solutions. Heuristic methods find better performance in both finding optimal solutions as well as scalability. They are not always as efficient as dispatching rules, but can outperform them. Finally, RL and DL methods are used to solve both static and dynamic JSSP problems. Creating these methods is more complex than others methods to solve job shop scheduling. However, they offer promising solutions for dynamic job shop scheduling problems, with possibilities such as real-time adaptation on changing circumstances. However, further research is needed to assess the scalability, robustness and generalization of such models to deploy them in industrial settings. Often, retraining or further training is needed to adapt to new problems.

4 Methodology

As the research gap is identified, the goal of this study is to use machine learning to solve job shop scheduling problems and test different approaches to assess their applicability to real-world environments. This chapter details the methodology to achieve this goal. First, in section 4.1 the practical considerations for scheduling in real-world environments are outlined, highlighting key assumptions and constraints. Next, the research design is described in section 4.3, followed by the selection of the machine learning method in section 4.2. The process of collecting and generating job shop scheduling problems is detailed in section 4.4, and ethical considerations are addressed in section 4.5. Section 4.6 gives the validity and reliability of this research methodology and finally, the limitations of the research are discussed in section 4.7.

4.1 Practical considerations for scheduling

To address the job shop scheduling problem effectively, several practical considerations and assumptions are outlined below.

General Assumptions

1. **Machine availability and maintenance:** Machine downtimes and maintenance are included within job processing times, simplifying the scheduling process.
2. **Size of job shops:** The research focuses on job shops with 5 to 20 machines, a range common in the literature (see Chapter 4).
3. **Comparison with established methods:** Scheduling methods will be benchmarked against standard practices like dispatching rules to evaluate their effectiveness.
4. **Categorization by AI expertise:** Recommendations are tailored to companies based on their AI expertise (None, Basic, Expert) to ensure practical applicability.

Dynamic Job Shop Scheduling Assumptions

1. **Daily scheduling:** The scheduling algorithm must produce solutions that fit within daily planning cycles, reflecting real-world operational constraints.
2. **Dynamic job arrivals:** Jobs start arriving at $t = 0$ with more jobs added over time, requiring the scheduling agent to handle ongoing arrivals and adjust schedules in real time.
3. **Job priorities and due dates:** Jobs have varied priorities and due dates, necessitating effective prioritization to meet deadlines and manage job importance.
4. **Predetermined arrival times:** Arrival times are based on an arrival probability to control the simulation environment and avoid randomness.

4.2 Choice of machine learning method

o select the most suitable machine learning method, we compare several approaches, including Proximal Policy Optimization (PPO), Multi-Agent Reinforcement Learning (MARL), Graph Neural Networks (GNN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) & Long Short-Term Memory networks (LSTM), and Deep Q-Learning Neural Networks (DQN). This comparison is based on three main criteria: complexity, generalization capabilities, and computational resource requirements.

- **Complexity:** This encompasses the sophistication of algorithms, learning challenges, and data requirements. Complexity also involves trade-offs such as exploration versus exploitation in reinforcement learning (RL) and depth versus computational cost in deep learning (DL).
- **Generalization Capabilities:** These depend on the algorithms, network architectures, and regularization techniques used. Effective generalization allows a model to perform well on new, unseen data.

- **Computational Resource Needs:** These are influenced by model complexity, data volume, and training efficiency, often necessitating advanced hardware.

A detailed comparison of these methods, as summarized in Table 5, is presented in Appendix E.

Method	Complexity	Generalization	Computational Time
PPO [82]	High	Good	Moderate to High
MARL [83]	Very High	Excellent	Very High
GNN [84]	High	Excellent	Moderate to High
CNN [85]	Moderate	Moderate	Moderate
RNN & LSTM [86]	High	Good	High
DQN [40]	Moderate	Good	Moderate

Table 5: Comparison of different possible DL and RL methods

Based on the comparison in Table 5, CNN and DQN are the most suitable in terms of balancing complexity and computational time. While CNNs are effective for visual recognition tasks, they are less versatile for other tasks. Therefore, DQN is selected as the preferred method for this model due to its overall balance between complexity, generalization, and computational efficiency.

4.3 Research Design

The research design is developed based on the goals, assumptions, and the selected DQN method. The focus is on both static and dynamic job shop scheduling using a quantitative approach. MATLAB has been chosen for implementing the DQN due to its comprehensive toolboxes for reinforcement learning and deep learning, which facilitate the development and training of complex models with extensive documentation.

To establish a proof of concept and set a benchmark for the research, the following steps are outlined:

- **DQN Implementation:** The DQN method will first be implemented and tested to ensure its functionality. This involves creating a simulation environment that accurately represents the job shop scenario, allowing the DQN to interact with and navigate through it effectively.
- **Benchmarking:** The performance of the DQN, integrated with the simulation environment, will be evaluated to establish a performance benchmark. This benchmark will serve as a reference for comparing the effectiveness of other scheduling agents.

To further analyze the DQN's effectiveness and its applicability to real-world environments, the following tests will be conducted:

- **Hyperparameter Tuning:** To enhance the DQN's performance beyond the benchmark, hyperparameters and the neural network architecture will be tuned using random hyperparameter optimization. Detailed procedures are discussed in Chapter 7.
- **Problem Size Evaluation:** Agents will be trained on problems of varying sizes, as described in the standardized problems outlined in Chapter 3.1. This aims to assess the applicability of the tuned hyperparameters across different problem scales.
- **Generalization Assessment:** Agents will be trained on specific job shop problems with varying amounts of training data to evaluate their ability to generalize and perform effectively with different data volumes.
- **Transfer Learning:** Transfer learning will be investigated as a method to leverage previously acquired knowledge to improve performance on new, related problems. This involves training a pre-trained neural network on a new problem, rather than training a new network from scratch. The effectiveness of this approach will be examined by applying pre-trained agents from the generalization assessment to new, singular problems. This test will evaluate the benefits of transfer learning and its potential for enhancing real-world applicability.

Additional details on these tests and methodologies are provided in Chapter 7.

4.4 Data Collection and Problem Generation

The data for this research is sourced from both standardized static job shop scheduling problems and newly generated problems. The aim is to cover a broad range of job shop configurations while also assessing performance in comparison to other literature.

4.4.1 Generating Static Job Shop Scheduling Problems

Static job shop scheduling problems are created using a method adapted from Taillard [19]. The process involves:

- **Specification:** Define the number of jobs, machines, and the range of processing times.
- **Generation:** Create matrices for job sequences and processing times using uniform distributions.
- **Output:** Produce a set of problems characterized by job sequences and processing times.

A more detailed explanation on generating static job shop scheduling problems as well as pseudo-code is given in Appendix D.1.

4.4.2 Transforming Static to Dynamic Problems

To ensure consistency, standardized and generated problems are transformed, instead of generating new dynamic ones. To transform these, the following characteristics need to be defined:

- **Arrival Times:** Each job is assigned an arrival time based on a predefined arrival rate (change of a job arriving per time unit), indicating when it becomes available for processing.
- **Priority Values:** Jobs are assigned priority weights to influence scheduling decisions, with higher priority jobs receiving more urgent attention.
- **Due Dates:** Due dates are calculated based on the job's arrival time, processing times, and a slack time that allows for scheduling flexibility.

These characteristics are explained in greater detail in Appendix D.2.

The transformation from static to dynamic problems is represented in Table 6. This figure depicts how arrival times, due dates, and priorities are integrated into the static problem matrix to create a dynamic scheduling environment, where in Table 6b the first digit represents arrival time, second is the due date and third the priority value, or weight.

2	1	0	3	1	6	3	7	5	3	4	6
1	8	2	5	4	10	5	10	0	10	3	4
2	5	3	4	5	8	0	9	1	1	4	7
1	5	0	5	2	5	3	3	4	8	5	9
2	9	1	3	4	5	5	4	0	3	3	1
1	3	3	3	5	9	0	10	4	4	2	1

(a) ft06 matrix

0	34	1	2	1	0	3	1	6	3	7	5	3	4	6
2	57	1	1	8	2	5	4	10	5	10	0	10	3	4
4	62	3	2	5	3	4	5	8	0	9	1	1	4	7
8	51	1	1	5	0	5	2	5	3	3	4	8	5	9
10	43	1	2	9	1	3	4	5	5	4	0	3	3	1
11	65	3	1	3	3	3	5	9	0	10	4	4	2	1

(b) Dynamic ft06 matrix

Table 6: Transformation from static to dynamic matrix

For detailed methodologies and pseudo-code used for problem generation and transformation, refer to Appendix D.2.

4.4.3 Sampling Strategy

The sampling strategy involves:

- **Static Problems:** Utilize the 242 predefined static problems, found in Chapter 3.1, and newly generated instances.
- **Dynamic Problems:** Extend static problems by adding arrival times, due dates, and priority values.
- **Sampling Units:** Each problem instance, whether generated or predefined, is treated as an individual sampling unit.
- **Sample Size:** Includes both the predefined and newly generated problem instances. The exact number used for training and testing will be specified based on experimental needs.

This approach ensures a comprehensive evaluation of the proposed machine learning models across various job shop scenarios.

4.5 Ethical Considerations

No ethical considerations arise in this research. The study only involves simulation-based research and data analysis in job shop scheduling using machine learning techniques. No human participants are involved, and the data used for training and evaluation are generated through MATLAB or obtained through open-source libraries.

4.6 Validity and Reliability

To ensure that the experiment is designed to produce accurate and reliable results, validity and reliability are defined in this research.

4.6.1 Validity

The validity ensures that the experiment will be set up in a way that there is confidence in our results. This is done by:

- **Randomization:** To ensure that no bias is created towards certain variables, mainly in terms of hyperparameters, these are randomly chosen.
- **Balanced design:** The experiments are designed in a way to ensure that each configuration of the RL agent is tested on an equal number of job shop instances. If the experiment requires varying complexities and characteristics, that will be included and equal per agent as well. This balanced approach helps to ensure that there is less influence of external factors on the results.
- **Controlling variables:** As much external factors and variables as possible are tracked, to ensure that any changes in our results can be determined based on these. By controlling these variables, the effects of the reinforcement learning agent on the job shop scheduling performance can be isolated.

4.6.2 Reliability

The reliability is about ensuring that the results are consistent and trustworthy. This is done by:

- **Multitude of trials and analyses:** When testing the eventually created agent(s), multiple agents with the same parameters and variables are trained on the exact same data, to ensure that the training of the agent is stable and consistent.
- **Validation procedures:** The performance of the agent is validated by comparing results across different training iterations or agents. This helps verify the generalisation of the findings across different experimental settings.

- **Transparency and reproducibility:** Documentation of the experimental procedures will be made, including data of the preprocessing steps, hyperparameter settings and evaluation metrics. This transparency enables independent verification and replication of the results by other researchers.

4.7 Limitations

The potential limitations in this research can be found in model and data assumptions and simplifications, simulation vs. real-world scenarios, algorithm constraints, performance metrics and computational resources.

The simplification and assumptions are found in the environment and data. The environment to emulate the job shop scheduling problems is simplified due to the complexity of the problem already being substantial. As there is no information on real-world problems available at this point, it is assumed that the defined dynamic manufacturing environment is sufficient enough to represent real-world environments.

RL and DL algorithms have issues in convergence problems and sensitivity to hyperparameters. These models require significant hyperparameter tuning, which requires significant computational resources and time, especially using neural networks. This computational complexity also means that training and testing of these models necessitates substantial computational resources. As the size and complexity of the problems grow, the computational resources required grow exponentially.

Interpretation of these models can be hard due to the "black box nature", especially for neural networks. While they produce outputs from the given inputs, understanding internal decision-making processes is difficult. This lack of transparency makes it hard to diagnose the issues of suboptimal performance.

Complex reward structures found in RL and DRL can make it hard to understand the reward signal. While makespan is a good performance metric, it may not always indicate the best-performing model. Considering additional metrics such as the average utilization rate of machines might indicate that one agent makes better use of the resources despite a higher makespan, depending on the definition of the average utilization rate. Hence, it is important to use multiple performance metrics to fully assess the effectiveness of created models.

By acknowledging these limitations, the research recognizes the constraints and potential areas for improvement while highlighting the robustness of the current findings withing these constraints.

5 Defining and explaining DQN

In this chapter, the DQN method is explained, elucidating its foundational principles, defining its key components, and outlining its implementation. DQN uses episodes, explained in Chapter 2.3, rather than epochs, explained in Chapter 2.2.

5.1 The DQN method explained

DQN is a hybrid method combining both DL and RL. DQN is a combination of Q-learning and neural networks. Q-learning is a model-free RL method, which learns the quality of possible actions in a given state. The Q-value, thus representing the quality, of taking action a in state s is denoted by Q(s,a). This method is able to learn an optimal policy which maximizes the cumulative reward by updating Q-values based on the Bellman optimality equation, given in Eq. 4. DQN can be combined with different neural networks, like CNN and GNN. However, in this research a general neural network is used.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{4}$$

5.1.1 Neural networks: Target and Q-network

The DL part of DQN is found in the use of deep neural networks. In DQN, the neural network is used to approximate the Q-values. The neural network takes the state as an input and outputs the Q-values for all possible actions. This network is trained to minimize the temporal difference error between the predicted Q-values and target Q-values obtained from the Bellman equation.
A DQN utilizes two different neural networks, a value network and a target network. The value network, also referred to as the Q-network, estimates the Q-values for each state-action pair. Instead of using the Bellman equation, the network itself estimates the Q-values, based on the inputs, that go through the layers to the outputs, which represent the Q-values, of the network. Thus, the layers and the connection between the layers, being controlled by the weights and biases, work together to make the Q-value estimations. The target network is a separate network, estimating the target Q-values. It has an equal architecture as the Q-network, with frozen parameters that are updated less frequently. The purpose of this target network is to stabilize the training by providing consistent target Q-values during update of the Q-network.
The interaction between these is based on the Q-network estimating the Q-value Q(s_t, a_t) for given state s_t and action a_t. The target Q-value is computed using the gained reward r_t, received from taking action a_t in state s_t, plus the discounted maximum Q-value of the next state, predicted by the target network, given by Eq. 5.

$$y_t = r_t + \gamma \max_{a'} Q_{\text{target}}(s_{t+1}, a') \tag{5}$$

Here, γ is the discount factor, $Q_{target}(s_{t+1}, a')$ are the Q-values estimated by the target network for the next state s_{t+1} and possible action a' and y_t is the target Q-value.

The Q-network is updated by training it to minimize the difference, also known as error, between predicted Q-values and target Q-values. This is the TD error. The loss function for the Q-network is given in Eq. 6. Here, θ represents the parameters of the Q-network and D is the replay buffer containing past experiences.

$$L(\theta) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D} \left[\left(y_t - Q(s_t, a_t; \theta) \right)^2 \right] \tag{6}$$

To conclude, this dual neural network model is used to improve upon the stability of the learning process as well as avoiding divergence. The parameters of the target network are updated periodically by copying the parameters from the Q-network. Hence, the target network parameters θ' are updated to match the parameters of the Q-network θ after a fixed number of steps:

$$\theta' \leftarrow \theta$$

5.1.2 Creating a memory: Experience replay

The experience replay, also known as a replay buffer, is a technique used to improve the stability and efficiency of learning. Experience replay involves storing past experiences (state, action, reward, next state) in a replay buffer. Instead of updating the Q-network parameters using the most recent experience, DQN samples mini-batches of experiences from the replay buffer during training. By sampling experiences randomly from the replay buffer, experience replay breaks correlations in the data, to prevent the network from being biased by the sequential nature of the data. Reusing these past experiences allows the agent to learn from each experience, which leads to improved data efficiency and faster learning. The experience replay also helps stabilize training by providing a diverse set of experiences for the network to learn from, which can prevent the network form overfitting to recent experiences.

The replay buffer has a fixed capacity, and the new experiences replace the oldest experiences once the buffer is at max capacity. Experiences are stored as tuples (state, action, reward, next state) in the replay buffer.

5.1.3 Minimizing loss function: Optimization algorithms

Optimizer algorithms are used to update the weights of the Q-network, which is a critical component of training a DQN. The choice of optimization algorithm affects the performance and convergence of the model. SGD, Adam and RMSprop, as discussed and explained in Appendix B.3. The optimizer minimizes the loss function, given in Eq. 6, by adjusting the network parameters. The performance of these three algorithms will be tested during the experimental research.

5.1.4 Combining all components

The interaction of all the aforementioned components is depicted in Figure 7. Here, the replay memory stores the experiences gained by the agent. These experiences are used to feed the value and target network, as well as the optimizer. First, the mini-batch of N number of experiences is randomly chosen from the replay memory. From this batch, the states are used as input for the value network. This value network calculates the Q-values of these states for all the actions, which are then used in the optimizer. The future states of the batches are used in the target network, to compute the future Q-values. Finally, the chosen action and reward from the batch are used as input to the optimizer. This optimizer finally updates the network parameter based on these Q-values and interactions with the loss function from Eq. 6. Finally, every n episodes the parameters of the value network are transferred to the target network.

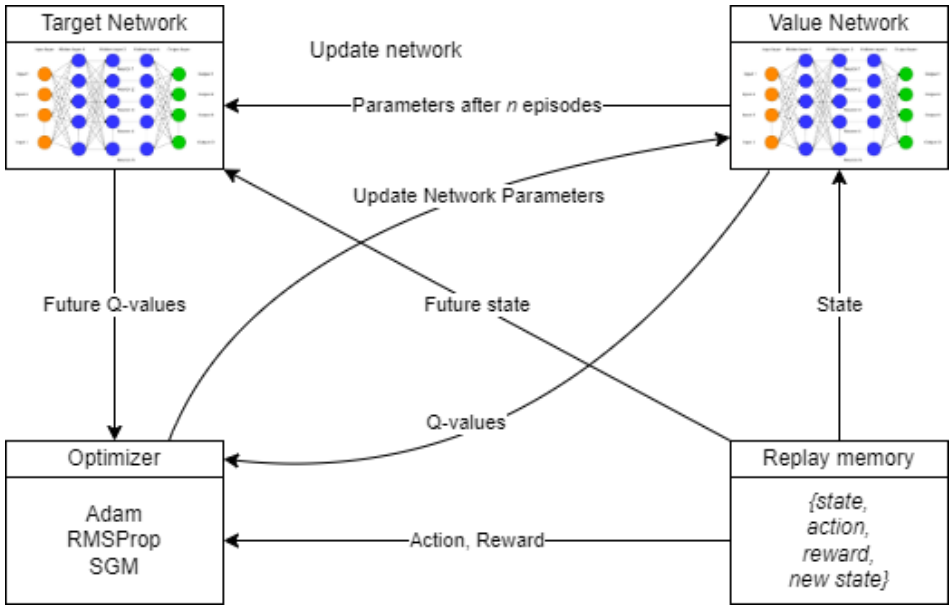


Figure 7: Interaction between components in DQN

5.2 Other definitions for implementing DQN

In this subsection, the important parts of a DQN in general is given and further defined. In Table 7 the components needed to define DQN are given. In this section, the neural network architecture will be elaborated on, and finally the other parameters such as hyperparameters will be defined.

Component	Description
Neural Network Architecture	Structure of the DQN's neural network, including input, hidden, and output layers as well as activation functions are defined
Loss Function	Function to measure error between predicted and target Q-values
Regularization Function	Techniques such as L2 regularization or dropout to prevent overfitting are used
Hyperparameters	Parameters controlling the training process, such as the learning rate and discount factor.

Table 7: Components needed to define and deploy a DQN agent

5.2.1 Creating Neural Network Architecture

To define the neural network for the DQN, several components must be specified, as outlined in Table 8.

Component	Description
Number of Layers	Total number of layers in the neural network, including input, hidden, and output layers
Number of Neurons	Number of neurons in each layer, determining the capacity and complexity of the network
Activation Functions	Functions applied to the output of each neuron, such as ReLU, sigmoid, or tanh, to introduce non-linearity
Input layer	The features or state representations fed into the input layer of the neural network
Output layer	The predicted Q-values for each possible action, produced by the output layer of the network

Table 8: Components that define the architecture of a neural network within DQN

The number of layers and neurons cannot be predefined as they significantly influence the performance and must be tuned according to the specific task and dataset. Inputs and outputs are closely tied to the environment, depicting the state for the inputs and the actions for the output, which will be defined in the next chapter. At this point, the activation functions between hidden layers are considered based on the problem requirements.

For job shop scheduling, the activation function ReLU (Rectified Linear Unit) is preferred due to its computational efficiency and ability to mitigate the vanishing gradient problem. The vanishing gradient problem occurs when gradients become very small, effectively stopping the network from learning. ReLU addresses this by allowing gradients to pass through unchanged for positive values, thus maintaining gradient flow and improving training efficiency. Further explanation and considerations for this choice are given in Appendix F, where multiple activation functions are explained, including ReLU in greater detail.

As "dying ReLU" is a common problem, where neurons effectively become inactive by constantly outputting zero due to receiving negative inputs, this issue can be tackled in multiple ways. For this research, the dying ReLU problem is addressed by using He-initialization. He-initialization sets the initial weights of the neurons using a Gaussian distribution with a mean of zero and a variance of $\frac{2}{n_{in}}$, where n_{in} is the number of input units in the layer. This initialization method ensures that the variance of the activations remains consistent across layers, reducing the likelihood that the neurons will output zero and thus become inactive. A more detailed explanation and other solutions to the dying ReLU problem can be found in Appendix F.1.

With this neural network, both the target and value network for the DQN are defined.

5.2.2 Other definitions for DQN

To finalize the definitions for the DQN, and thus enable deployment in MATLAB, yet undefined key components and their specifications are outlined in Table 9.

Variable/Component	**Definition**
Batch Size	The batch size determines the number of past experiences sampled during each training step. A larger batch size can increase computational time significantly, while a smaller size may reduce performance due to less diverse experiences.
Replay Buffer Size	The size of the experience replay buffer, which stores past experiences (state, action, reward, next state). A larger buffer allows the agent to store more experiences, potentially leading to better learning but requires more memory. A smaller buffer might not capture the diversity of experiences.
Learning Rate (α)	The learning rate (α) controls the step size during gradient descent. This crucial hyperparameter influences the convergence speed and stability of the training process.
Discount Factor (γ)	The discount factor determines the importance of future rewards in the Q-value estimation.
Exploration Rate (ϵ)	The exploration rate usually starts high to encourage exploration of the action space and decays over time to favor exploitation of learned policies.
Exploration Decay Rate	The rate at which the exploration rate (ϵ) decays over time. It affects how quickly the agent shifts from exploration to exploitation.
Minimum Exploration Rate	The lower bound for the exploration rate (ϵ) after decay. Ensures that the agent continues to explore to some extent even after the exploration rate has decayed.
Target Network Update Frequency	How often the target network parameters are updated to match the Q-network parameters. Frequent updates can destabilize training, while infrequent updates might slow down learning.

Table 9: Key components and definitions for deploying the DQN in MATLAB

For this research the batch size, replay buffer size and target network update frequency are not changed. Specific values will be given in Chapter 7. To deploy DQN in MATLAB, a brief step by step explanation is given in Appendix G. As the DQN method is now defined, in the next chapter the environment to explore will be defined.

6 Defining the job shop: the environment

In this chapter, the environment for the DQN agent to explore is proposed. First, the general scheduling is presented. Next, the components of the environment are identified and their purposes are explained, and then the detailed definitions of these components are given to show how they are implemented.

Pseudo-code of the general outline for an environment is given in H.1. The environment is implemented using the reinforcement learning toolbox of MATLAB to create a custom environment using the rlCreateEnvTemplate function [87].

6.1 Definitions of the environment

MDP is used to describe the environment. In Table 10 the necessary components to define the environment are given, as well as a brief explanation with regards to job shop scheduling. Next the state, action space and reward are further elaborated on. The step function will be elaborated on in the next section, as this entails the whole scheduling process. The reset function resets the whole environment, meaning the created schedule is cleared and the environment starts at the beginning of a job shop scheduling problem. A more detailed explanation about the reset function is given in Appendix H.3

Component	Description
State	Represents the current status of the job shop, including job and machine statuses
Action	Decision made by the agent, such as selecting the next job or operation to schedule
Reward	Feedback signal indicating the quality of the action, aiming to optimize scheduling performance
Step Function	Updates the state based on the chosen action, returns new state, reward, and if the episode is completed
Reset Function	Resets the environment to the initial state at the start of each episode
Observation Space	Defines the format and bounds of the state observed by the agent
Action Space	Defines the possible actions the agent can take

Table 10: Necessary components to define the job shop scheduling environment

6.1.1 Inputs for the agent: defining the state

In this research, the observation space and state represent the same data. Hence, the agent has full availability to the state. Hence, the state is essential for guiding the agent's decision-making process based on available data. It ensures the agent can make informed decisions by providing comprehensive insight into both jobs and machines within the job shop environment. Given the distinct requirements of static and dynamic job shop scheduling problems, two distinct state spaces are defined for each.

The state space is structured as matrices, with each row representing a single input node for the neural network used in DQN.

Static JSSP inputs

For the static JSSP, the job information is represented in the following matrix:

Job 1	s_1	m_1	t_1	o_1	c_1	f_1
2	s_2	m_2	t_2	o_2	c_2	f_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Job i	s_i	m_i	t_i	o_i	c_i	f_i

Here, i represents the job ID, s indicates job status (0 for non-existing/finished, 1 for active), m specifies the machine for the next operation, t denotes the processing time on the next machine, o shows the number of operations left to complete the job, c represents remaining time to complete the job, and f indicates finished status (0 for not finished, 1 for finished).

The machine state matrix for static JSSP is represented as:

Machine 1	s_1	p_1	u_1	0	0	0
2	s_2	pm_2	u_2	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Machine j	s_j	p_j	u_j	0	0	0

Here, j represents the machine ID, s indicates machine status (similar to job status), p specifies the number of operations completed, and u represents utilization rate.

In the matrix, the zeros are placeholders to ensure the job and machine state matrices are the same size, as the DQN agent requires a single matrix input.

Dynamic JSSP inputs

Dynamic job shop scheduling (DJSP) incorporates additional data such as arrival time a, due date d, and priority weight w into the state space matrix. The state space matrix for DJSP is defined as follows:

Job 1	s_1	w_1	a_1	d_1	m_1	t_1	o_1	c_1	f_1
2	s_2	w_2	a_2	d_2	m_2	t_2	o_2	c_2	f_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Job i	s_i	w_i	a_i	d_i	m_i	t_i	o_i	c_i	f_i

To accommodate these variables, the machine state matrix for DJSP includes additional placeholders.

These matrices provide the agent with comprehensive information necessary for effective decision-making in dynamic job shop scheduling scenarios.

6.1.2 Scheduling of operations: defining the action space

When defining the action space, several considerations must be taken into account. Notably, DQNs are constrained to discrete action spaces. In this case, dispatching rules are utilized for their efficiency in conjunction with the complex learning capabilities of DQNs, a combination discussed in Chapter 3. In Table 11 the chosen dispatching rules as depicted.

Action	**Definition**
SPT	Selects the job with the shortest processing time for the next operation.
LPT	Selects the job with the longest processing time for the next operation.
SRPT	Selects the job with the shortest remaining total processing time.
LRPT	Selects the job with the longest remaining total processing time.
STPT	Selects the job with the shortest total processing time.
LTPT	Selects the job with the longest total processing time.
FIFO	Selects the job that arrived first in the queue.
LIFO	Selects the job that arrived last in the queue.
LOR	Selects the job with the least number of remaining operations.
MOR	Selects the job with the most number of remaining operations.

Table 11: Chosen actions: Dispatching Rules

Based on the Q-values estimated from the current state by the agent, an action is given based on the highest Q-value, determining which operation to schedule next according to the chosen dispatching rule.

6.1.3 Incentive to optimize performance: defining the reward structures

The reward structure is crucial for guiding the agent's behavior by providing feedback on its actions. In static job shop scheduling, where minimizing makespan is the primary objective, the reward structure focuses on maximizing the utilization rate. In dynamic job shop scheduling, multiple objectives

are considered, such as minimizing makespan, maximizing utilization rate, minimizing tardiness, and managing slack.

To address these objectives, a reward structure is proposed for maximizing utilization rate. Detailed mathematical equations for four distinct just-in-time reward structures, as well as a structure for maximizing slack, are provided in Appendix H.2. Here, "just-in-time" refers to minimizing both slack and tardiness to deliver jobs as close to their due dates as possible, thus reducing storage time.

Reward for maximizing utilization rate

The proposed reward for maximizing the utilization rate is determined based on the change in the utilization rate resulting from scheduling a job on a specific machine at a given time. Since the utilization rate affects the makespan, this approach indirectly minimizes the makespan. Equation 7 represents the reward represents the reward r gained per step, where u is the utilization rate for machine j. Subsequently, Equation 8 provides the accumulated reward R, where n is the total number of machines. As the reward is calculated based on the change in utilization per machine and does not directly account for the total completion time at each step, the makespan is only indirectly influenced by the reward. Therefore, a higher received reward does not necessarily correspond to a lower makespan.

$$r = \Delta u_j \tag{7}$$

$$R = \sum_{j=1}^{n} u_j \tag{8}$$

6.2 Defining the scheduling for the environment

With the environment now defined in terms of an MDP, the job shop scheduling needs to be specified. This involves detailing the functionality of the step function, enabling the agent to navigate the environment based on its chosen actions.

6.2.1 Flowchart of basic scheduling

To illustrate the basics of creating a job shop scheduling environment, a flowchart is presented in Figure 8. The input data is a job shop scheduling problem, structured as described in Table 6 from Chapter 4.4.2. The job shop is initialized with an empty schedule. Operations of jobs are scheduled sequentially until all jobs are scheduled, resulting in a feasible schedule for the job shop scheduling problem.

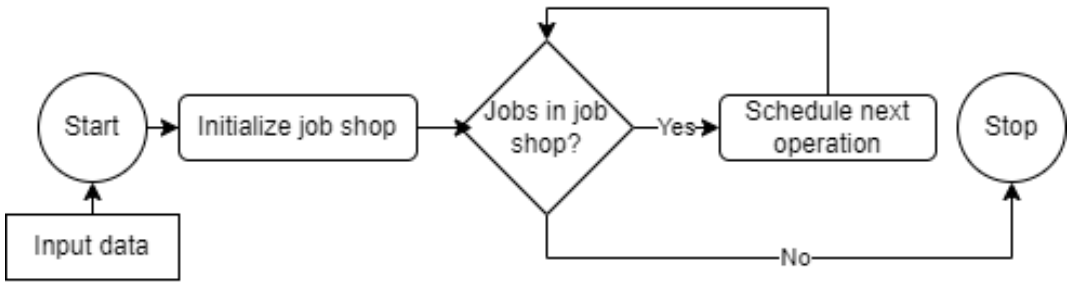


Figure 8: Flowchart of general job shop scheduling process

6.2.2 Algorithm for scheduling: static job shop

Algorithm 1 provides the pseudo-code for solving a general static job shop scheduling problem for a single episode. An episode, in this context, is the sequence of interactions between the agent and the environment, starting from the initial state and concluding when all operations are scheduled.

Algorithm 1 General workings of a job shop

Input: General job shop problem
Output: Scheduled jobs of job shop problem
while not all jobs have been processed **do**
  Choose an action
  Based on the action, determine the job and its operation to schedule
  Determine the start and end time:
  if starting time of the job $\geq$ starting time machine **then**
    Starting time of operation = starting time of the job
  else
    Starting time of operation = starting time machine
  end if
  End time = starting time + processing time of operation
  Schedule the job
end while

During the later stages of this study, an inefficiency was identified in Algorithm 1. The dispatching rules produced different makespan results for the same problems compared to Liu [81]. The issue was that operations were not being planned before other operations, thus not filling planning gaps. Due to time constraints, this research continues to use Algorithm 1. The issue and an updated algorithm are detailed in Appendix H.4, where the revised algorithm is given in Algorithm 6. This update is applied to the dynamic job shop scheduling algorithm described next.

6.2.3 Algorithm for scheduling: dynamic job shop

In Algorithm 2, the workings of dynamic job shop scheduling is given. The changes from static, Algorithm 1, to dynamic are as follows:

- Initializing a current time at $t = 0$ to simulate real-time scheduling.
- The scheduling of jobs checks for gaps in the schedule, identifying if the planning gaps are suitable for the next operation to be scheduled at that time while keeping the operation sequence in mind, ensuring efficient resource allocation.
- To simulate real-time scheduling, if no more operations can be planned for that time, the current time is incremented. During this time increment, the algorithm checks if new job can e added based on the arrival times.

By handling gaps and new job arrivals, this algorithm adapts to the dynamic nature of the job shop scheduling problem, aiming to optimize the schedule continuously.

Algorithm 2 Job shop scheduling algorithm for dynamic job shop

```
Input: General dynamic job shop problem
Output: Scheduled jobs of dynamic job shop problem
Initialize the current time at t = 0
while not all jobs have been processed do
  Choose an action
  Based on the action, determine the job and its operation to schedule and which machine to
  schedule on
  Determine if there are gaps in the planning for the machine
  if there are gaps in the planning for the machine then
    Determine where the gaps in the planning are
    Determine how big the gaps in the planning are
    Determine if the processing time of the operation fits in the gap(s)
    if processing time fits in one of the gaps then
      Find the earliest possible gap to plan the job in based on arrival rate, starting time and
      processing time
      Schedule the job in the gap
    else
      Determine the start and end time:
      if starting time of the job ≥ starting time machine then
        Starting time of operation = starting time of the job
      else
        Starting time of operation = starting time machine
      end if
      End time = starting time + processing time of operation
      Schedule the job
      while Current Time is smaller than each of the last planned time of all jobs do
        Current time +1
        check if the current time is smaller than each of the last planned time of all jobs
        check if jobs can be added based on the arrival time in comparison to current time
      end while
    end if
  end if
end while
```

With the definitions of the DQN and the job shop scheduling environment now established, the groundwork has been done for implementing and testing our approach. The next chapter will detail the experimental setup, including how the environment and learning algorithm will be tested, evaluated, and validated to ensure their effectiveness in solving job shop scheduling problems.

7 Experimental setup

This chapter details the experimental setup used to investigate capabilities and limitations of the proposed method in solving job shop scheduling problems. The experiments are designed to validate the effectiveness of the proposed model, given in Chapter 5, in both static and dynamic job shop scheduling environments. The primary goal is to assess the usefulness of the method under various conditions and identify the limitations.

7.1 Setting a benchmark

Each experiment starts by creating a benchmark agent with standard values provided by MATLAB. The agent's learning behavior is evaluated using the training graph, which displays its performance over time. Following the training phase, a Gantt chart is generated when testing the created agent, to assess the interaction between the environment and the agent as well as the performance and statistics in terms of the found makespan and utilization rate, as well as rewards. This benchmark serves as a reference point for further experiments.

To elaborate, the standard settings for hyperparameters and other values are given in Table 12. Here, α represents the learning rate, γ denotes the discount factor, ϵ_{init} is the initial exploration rate, ϵ_{min} is the minimum exploration rate, and ϵ_{decay} refers to the rate at which exploration decreases. D indicates the size of the experience replay buffer, B represents the number of experiences used in each training step, L specifies the number of layers in the neural network, N denotes the number of neurons per layer in the network, U_{freq} indicates how often the neural network parameters are updated and E represents the total number of episodes.

Other options to change can be found in documentation from MATLAB on options for DQN agents [88].

Parameter	α	γ	ϵ_{init}	ϵ_{min}	ϵ_{decay}	D	B	L	N	U_{freq}	E
Value	0.001	0.99	1.0	0.01	0.005	1e5	64	1	64	10,000 steps	8000

Table 12: Standard hyperparameter values for DQN in MATLAB

Other values are for the maximum steps per episode, which is set to a really big number, as the steps per episode should depend on completing the job shop scheduling problem, as well the number of future rewards used to estimate the value of the policy, specified as a positive integer, which is default set to 1. Finally, a reward scaling factor is introduced to change the size of the reward signal given to the agent. This influences how the values of the agents are changed.

7.2 Hyperparameter tuning

To improve upon a created benchmark agent, the objective is to identify hyperparameter settings that enhance agent performance while exhibiting desired training behaviors. To find fitting results in comparison to literature, hyperparameter tuning is done by solving one specific problem.

The hyperparameter tuning process involves the following steps:

1. Define the hyperparameters and other relevant variables for the agent and neural network.
2. Initialize a set of agents, each with a unique configuration of the defined hyperparameters.
3. Evaluate the agents' learning behavior by analyzing training performance graphs.
4. Assess agent performance using various metrics such as makespan, utilization rate, reward gained, and in the case of dynamic environments, slack and tardiness.
5. Identify the agents that exhibit the best performance and desired learning behavior.
6. Adjust the hyperparameters and variables based on the performance and learning behavior of the agents.

This iterative process continues until an optimal set of hyperparameters is identified, resulting in desired learning behavior and performance that is competitive with other methods. Desired learning behavior is characterized by the agent's reward converging to the Q0 estimate.

The Q0 estimate is calculated by performing inference on the critic at the beginning of each episode. It represents the expected long-term reward based on the current observation. Ideally this estimate would match the actual total reward collected during the episode. However, in practice, it is not always necessary for the Q0 estimate to align perfectly with the actual reward, as the actor may converge before the critic. Further elaboration on data analysis is given at the end of the chapter.

7.3 Training per standardized problem

With the optimized hyperparameters established, an agent is trained on each problem from a set of standardized job shop scheduling problems, creating multiple agents to each solve one specific problem. The performance is then compared to results found by Liu [81]. This approach helps to evaluate the performance of DQN with hyperparameters optimized for one specific problem across various other problems.

This evaluation aims to determine if the hyperparameters are effective across different problems when the inputs of the neural network are changed. As the number of jobs and machines vary, the state, as defined in Chapter 6.1.1, changes and serves as the input for the neural network. Consequently, the architecture of the neural network adapts accordingly. Given that hyperparameter tuning can be intensive and time-consuming, it is crucial to assess their transferability to other problems, resulting in different neural network architectures.

In practical terms, understanding whether hyperparameters need to be tuned for each specific problem or job shop type, rather than for job shop scheduling in general, significantly impacts the labor and time required to implement machine learning, specifically DQN in this research. If the hyperparameters are transferable, it simplifies the application of machine learning techniques, making them more feasible and efficient for diverse job shop scheduling scenarios.

7.4 Training on multiple problems

After evaluating the performance and transferability of the hyperparameters, the next step is to test the generalization capability by training on multiple problems. Depending on the results from the standardized problems, hyperparameters may be further tuned, and the neural network architecture might be adjusted if necessary.

Categories are created based on the number of jobs and machines. For each category, a multitude of problems is generated, forming a comprehensive dataset. Within each category, multiple agents are trained on varying amounts of samples from the dataset, such as 2, 5, or 10 different problems. This step aims to assess whether training on a diverse set of problems enhances the agents' performance and robustness compared to standardized methods.

Evaluating generalization helps determine whether the developed model and hyperparameters are universally applicable or require customization for specific problem types. Demonstrating that an agent can perform comparably to dispatching rules indicates the method's potential for real-world deployment. As machine learning techniques are capable of understanding complex environments, there is potential for these methods to surpass traditional dispatching rules with continued research and development.

7.4.1 Testing of the agents on different problems

To evaluate generalization capability, the trained agents are tested on several problem sets:

- Problems they were specifically trained on.
- Problems with the same number of jobs, machines, and processing time distributions, but not seen during training.
- Problems with different processing time distributions, with the same number of jobs and machines.

This testing phase evaluates the agents' ability to solve similar job shop type problems that are not exactly the same, as well as unseen problems with different processing times.

In practical terms, if an agent is unable to solve problems with a different range of processing times, it indicates the need for training on a broader variety of problems to handle unexpected data in job shop scenarios. Ensuring the agent can tackle diverse and unforeseen issues is important for its effective deployment in real-world environments.

7.5 Transfer learning

Transfer learning is a technique where knowledge gained from training an agent on one or more related problems is applied to a new, related one. This approach leverages pre-existing knowledge to enhance the efficiency and effectiveness of training on new tasks.

In the context of job shop scheduling, transfer learning involves two main phases. First, agents are pre-trained on a set of job shop scheduling problems to develop a more broad understanding of various scenarios. Once this pre-training is complete, the agents are further trained, or fine-tuned, on a specific problem. The objective of this fine-tuning phase is to determine whether the general knowledge acquired from the more diverse training can improve performance and reduce training time for the new, specific problem.

The practical application of transfer learning can reduce the time required to train an agent compared to starting from scratch for a new problem. If an agent trained on general job shop scenarios can be adapted to specific scheduling tasks, this could lead to time savings. For example, instead of requiring several hours to train from scratch, the training time might be be reduced with transfer learning which would enhance the efficiency of production operations.

This method could facilitate less time-consuming deployment of effective solutions, thereby improving operational efficiency in production settings. Due to time constraints, this study only assesses the transfer learning capabilities within the same type of job shop but with different problems.

7.6 Data Analysis

Data analysis is done for evaluating the performance of the trained agents. Several methods are employed to analyze the gathered data and assess the agents' effectiveness in solving job shop scheduling problems.

The first part of data analysis involves examining the training behavior of the agents. This is visualized through training graphs, given in Figure 9, which illustrate key metrics over the course of training. During training, several elements are monitored:

- **Reward per Episode:** This shows the reward received by the agent at each episode, reflecting the agent's immediate performance.
- **Average Reward:** The average reward calculated over a specified number of episodes provides a smoother view of performance trends, helping to identify overall learning progress.
- **Q0-Value:** The Q0-value represents the critic's estimate of the expected long-term reward based on the current observation at the beginning of each episode.

Together, these components reveal how rewards evolve, the stability of learning, and the agent's convergence towards optimal behavior. Desired learning behavior can be inferred from these graphs:

- **Convergence of Q0-Value:** The Q0-value should converge towards the average reward, indicating that the critic's estimates are becoming more accurate and consistent with the agent's performance.
- **Stable Average Reward:** A stable average reward with minimal fluctuations suggests that the agent has effectively learned the task and is consistently performing well.
- **Reduced Oscillations:** As training progresses, the reward per episode should show reduced oscillations, signaling that exploration is decreasing and the agent is focusing more on exploiting its learned knowledge.

Figure 9 illustrates a learning behavior, demonstrating how the reward trends and Q0-values help assess whether the agent has achieved the desired learning behavior.

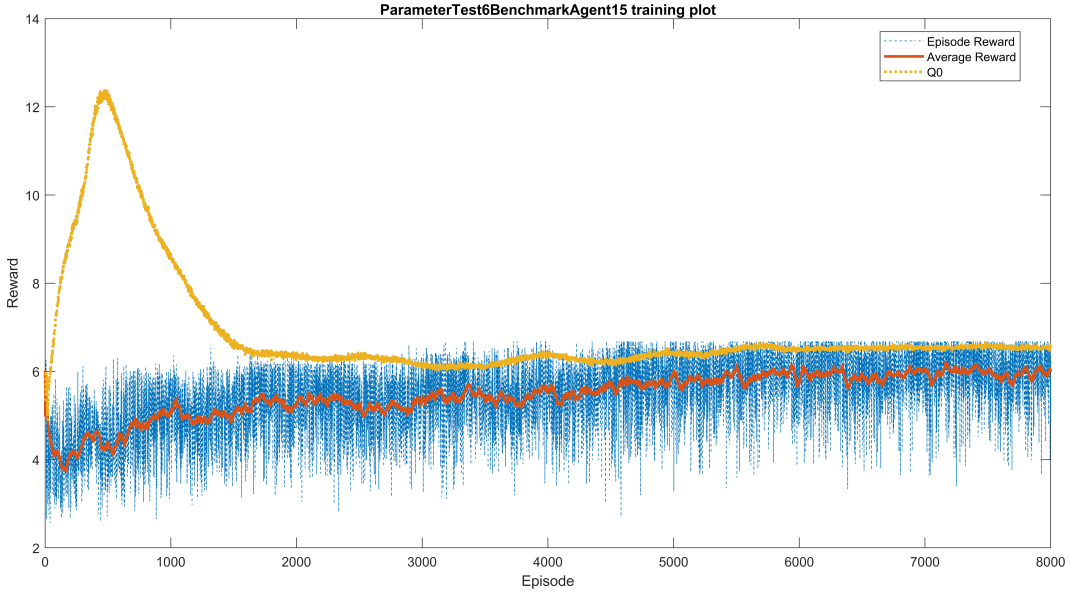


Figure 9: Example training graph for an agent

In addition to training behavior, agents are tested in the environment to gather various performance metrics. The primary metrics include makespan, utilization rate per machine and reward, as well as slack and tardiness for dynamic environments.

Next to these performance metrics, analyzing the actions taken by the agents provides valuable insights. The frequency and types of actions (e.g., specific dispatching rules) reveal how the agent's learned policy differs from standard methods and whether certain actions dominate. This analysis can highlight whether the agent is favoring particular strategies or whether it has developed a more nuanced approach to problem-solving.

To further evaluate and compare the performance, statistical methods are applied. Descriptive statistics such as mean, standard deviation, and variance are calculated to assess the consistency of the results, as mean values indicate the central tendency of performance metrics, while standard deviation and variance measure the variability and consistency across different problems and training runs.

These data analysis techniques ensure a thorough assessment of the agent's performance, highlighting both strengths and areas for improvement. By analyzing training behavior and performance metrics, a detailed understanding can be gained of how well the agents learn and generalize across various job shop scheduling scenarios.

With the experimental setup detailed, the next chapter will present the execution of these experiments and discuss the results obtained, to provide insights into the effectiveness of the training approaches and the practical implications of the findings.

8 Results

In this chapter, the experiments described in Chapter 7 are executed and the results are discussed. First, an initial testing is done of the whole model in a static job shop environment, with testing on standardized problems. After that, a second iteration is done to change according to the results of the first iteration, with training on multiple problems as well as transfer learning. Finally, the DQN method is assessed in a dynamic environment where agents are trained on multiple problems as well.

8.1 Initial testing of proposed method: Static job shop scheduling

In this section the initial testing of the proposed method is executed to assess the performance of the proposed method as well as learning from interacting with deep reinforcement learning. Here, the method proposed uses a neural network where the size of the inputs is based on the number of jobs and machines in a job shop, thus changing the architecture of the neural network based on the problem to give as much information on the problem as possible.

8.1.1 Testing of optimization algorithms and creating a benchmark agent

Three benchmark agents are created, with different optimization algorithms to assess which of the three to use. In this section, the agents are trained on the ft06 problem defined in Chapter 3.1. As this is a smaller problem (6 jobs, 6 machines) computational time can be kept to a minimum. First, the comparison is made between the optimization algorithms. All three agents trained with different optimizers use default settings as defined in Table 12 in Chapter 7.1. After choosing the optimization algorithm, the chosen agent will be assessed on performance as well the functioning of the environment.

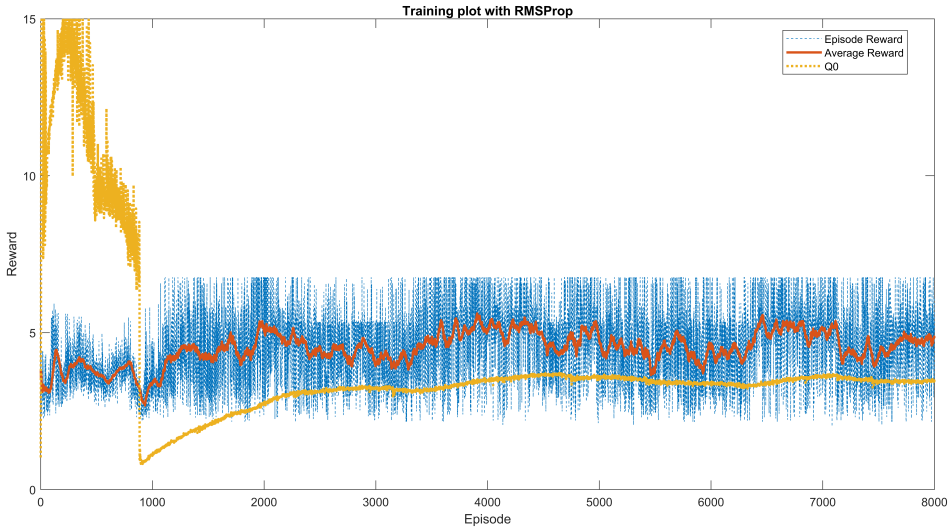
To assess the difference for the optimization algorithms, Figure 10 shows a comparison in learning behavior. From these graphs, it is found that RMSprop shows unstable and unwanted learning behavior, with no actual learning. Both SGDM and Adam show promising and stable learning behavior, although the Q0 estimations of Adam are more stable. Due to the fact that Adam is a more commonly used optimizer and shows stable Q0 estimations, Adam will be used as optimizer algorithm.

To create a benchmark, the agent is tested on the ft06 problem it was trained on. In Table 13 some statistical data is shown, where it is found that the reward found is consistent over multiple runs of the same problem. In this table, UL (upper limit) represents the highest value found, LL (lower limit) represents the lowest value found, var is the variance found and StD the standard deviation. The found makespan is only 79, which in comparison to other solutions found in Chapter 3 underperforms in comparison to other methods. Although the agent does not perform in comparison to other methods, it is able to learn an optimal policy and find consistent results. The agent is able to find this solution in 3071.8 seconds, or 51 minutes and 12 seconds.

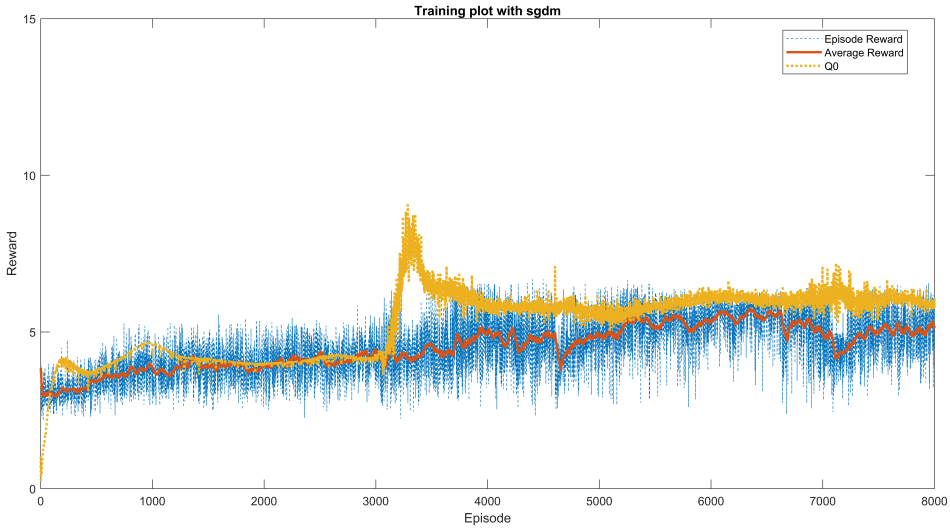
Variable	C	R
UL	79	4.6851
LL	79	4.6851
Mean	79	4.6851
Var	0	0
StD	0	0

Table 13: Found statistics for makespan and rewards

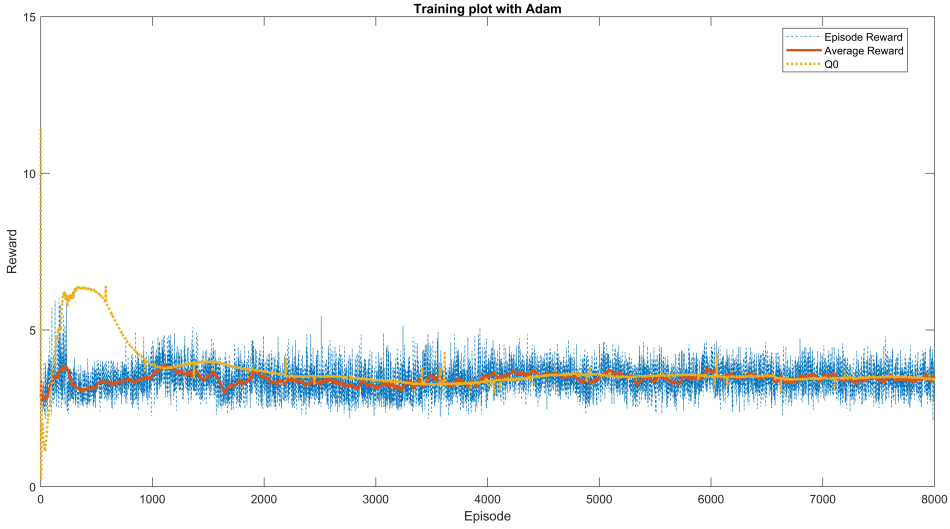
In Figure 11 the Gantt chart is shown, depicting the found solution by the agent. From this Gantt chart it can be concluded that the agent schedules the jobs and their operations in the correct order, when looking at the ft06 problem in Table 4 from Chapter 3.1.



(a) RMSprop



(b) SGDM



(c) Adam

Figure 10: Behavior of different optimizer algorithms

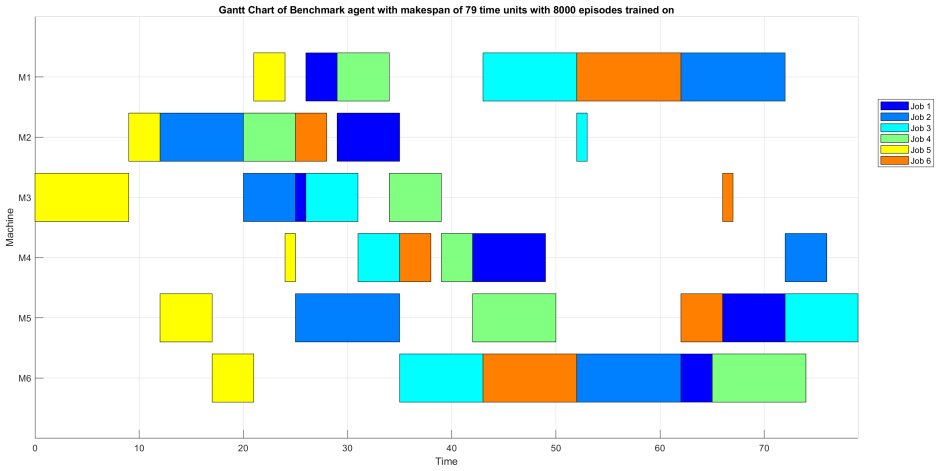


Figure 11: Gantt Chart illustrating the best schedule and makespan achieved by the benchmark agent.

In the pie chart given in 12 it can be seen that the approach mostly uses MOR, as well as SPT and LPT, while the other 7 defined actions are not used. An important note, looking at the specifics, the agent uses SPT and LPT for the first 16 actions, and afterwards only uses MOR. As per this data, it can be concluded that the agent does not balance the dispatching rules to find an optimal policy.

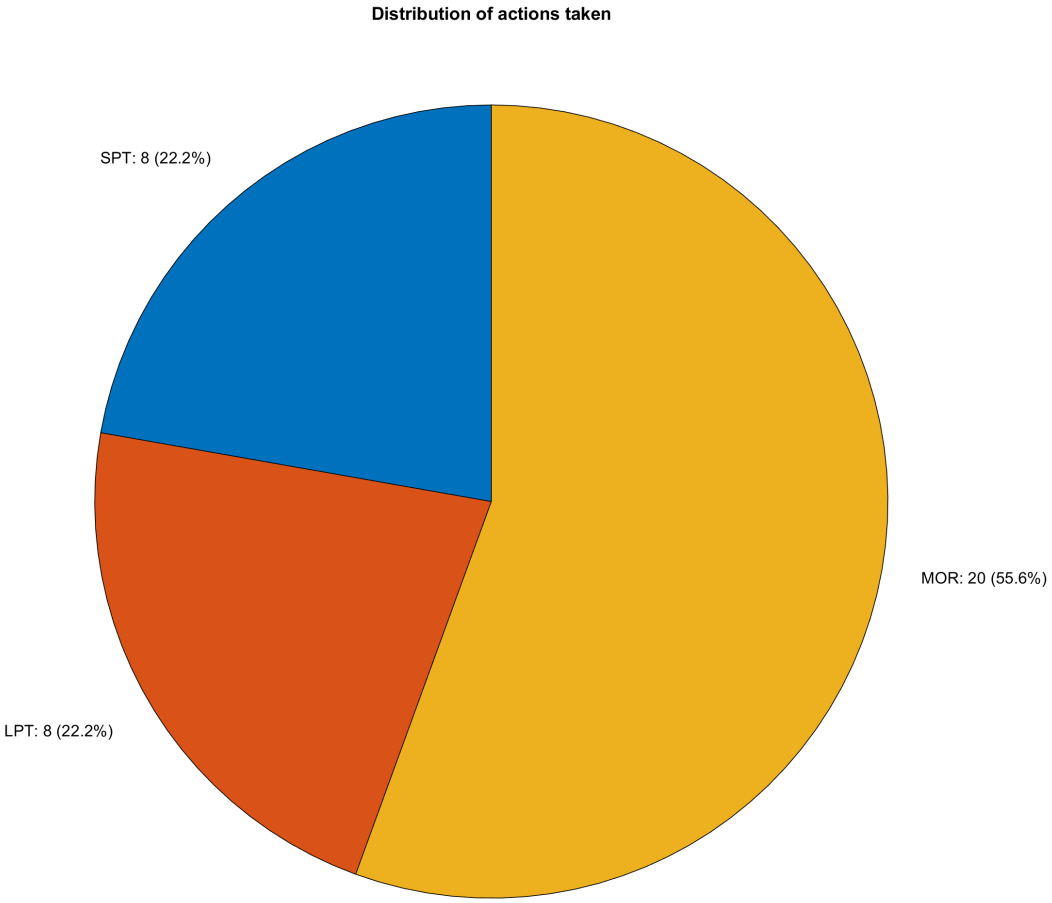


Figure 12: Pie chart of the different actions taken by the agent

It can be concluded that although performance is not yet up to standard, the environment works

correctly and the agent can be trained and tested on its performance. The agent does not balance certain actions, but prefers three specific actions. The next focus is on fine-tuning the hyperparameters to improve learning behavior and performance of the agent. New agents will be created by iteratively testing different sets of parameters and changing the range of parameters, while analysing their effectiveness, consistency and generalisation capabilities.

8.1.2 Hyperparameter tuning

As the benchmark is trained on the ft06 problem, the hyperparameter tuning will be performed on ft06 as well. For the hyperparameter tuning, the goal is to converge from a wide range of hyperparameters to a small set. Hence, multiple iterations are done to converge to a set with the best found performance and desired learning behavior. To compare performance, a bar graph is made, depicted in Figure 13. From this graphs data, three agents are found which show good performance, with agent 18 finding a reward of 6.22 and a makespan of 70, agent 26 finds 5.73 and 72, and finally agent 29 finds 5.73 and 67. Although not finding a low makespan, the performance is already improved in comparison to the benchmark agent.

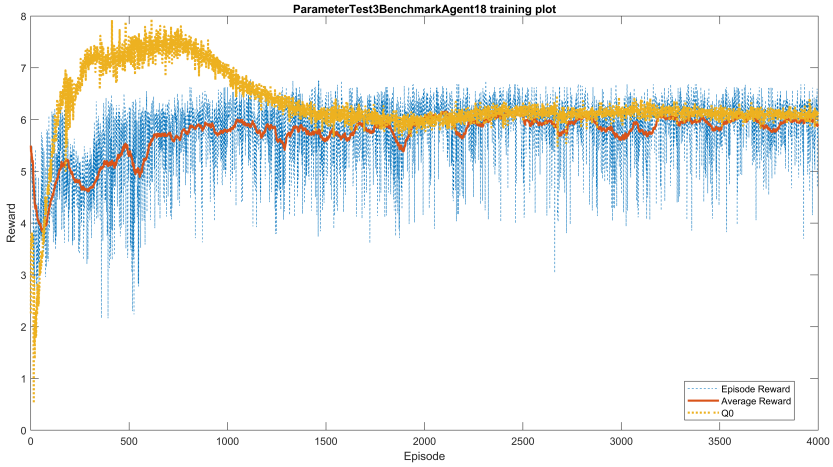
Figure 13: Comparison of agents from the second iteration

Next to performance, the learning behavior is assessed based on the training graphs, shown in Figure 14. In this figure, the y-axis of the sub figures are not equal. Here it is found that although they show comparable performance, the learning behavior of agent 18 seems to show the most desired behavior, as defined in Chapter 7.6. Hence, the hyperparameters of agent 18 are focused on for the next iteration. The values of all three agents are given in Table 14, where RS is the reward scaling factor and CT is the computational time during training.

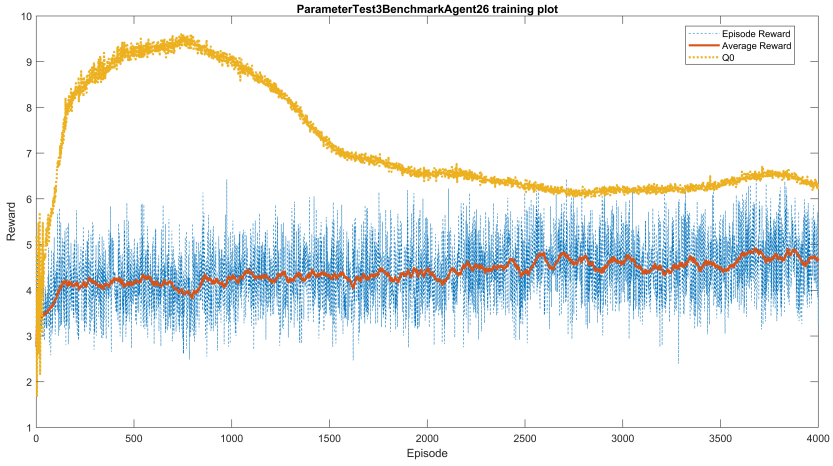
Agent	α	γ	$\epsilon_{\textbf{init}}$	$\epsilon_{\textbf{decay}}$	**D**	**N**	**L**	**CT (s)**
18	1e-3	1	1E-3	1e-6	10	128	5	1765.6
26	1e-4	1	0.5	1e-10	10	128	4	1435.4
29	1e-3	1	0.5	0.01	10	64	2	1061.3

Table 14: Parameters of agents of second iteration

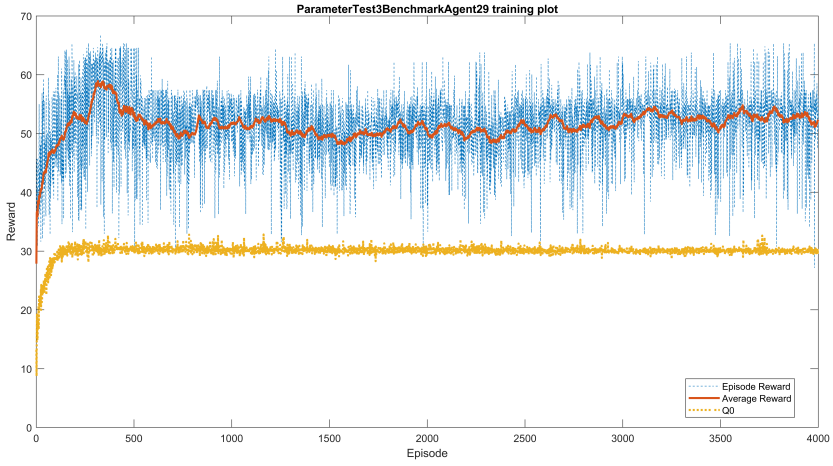
This method, as defined in Chapter 7.2, is used for each iteration. For hyperparameter tuning in this initial test, five iterations have been done, which are described in Appendix I. In Table 15 the range of hyperparameters are given for each iteration.



(a) Learning behavior agent 18



(b) Learning behavior agent 26



(c) Learning behavior agent 29

Figure 14: Comparison of learning behavior from agents of second iteration

It.	α	γ	$\epsilon_{\textbf{init}}$	$\epsilon_{\textbf{decay}}$	N	L	D	E
1	1e-1, 1e-4, 1e-7	1e-3, 0.5, 1	1e-3, 0.5, 1	1e-2, 1e-6, 1e-10	64, 128, 256	1, 2, 3, 4, 5	1, 10, 100	4000
2	1e-3, 1e-4, 1e-5	1e-3, 0.5, 1	1e-3, 0.5, 1	1e-2, 1e-6, 1e-10	64, 128, 256	2, 3, 4, 5	10, 100	4000
3	1e-3, 1e-4, 1e-5	0.5, 0.75, 1	1e-5, 0.1, 0.25, 0.5	1e-2, 1e-6, 1e-10	64, 128, 256, 512	4, 5, 6	10	4000
4	1e-3, 1e-4, 1e-5, 1e-6	1	1e-5, 0.1, 0.2, 0.3	1e-2, 1e-6, 1e-10	64, 128, 256	4, 5, 6	10	4000
5	1e-4, 1e-5	1	0.1, 0.2, 0.3, 0.4	1e-2, 1e-6, 1e-10	128, 256	5, 6, 7	10	8000

Table 15: Hyperparameter settings for each iteration

For the final iteration, it was found that agent 13 performs best in terms of rewards, with a reward of 6.64 and a makespan of 61 time units, with desired learning behavior depicted in Figure 15. The settings for the agent are given in Table 16, which enables the agent to find an improved solution in 52 minutes and 49 seconds. Hence, the computational time is comparable while being able to improve upon the found makespan.

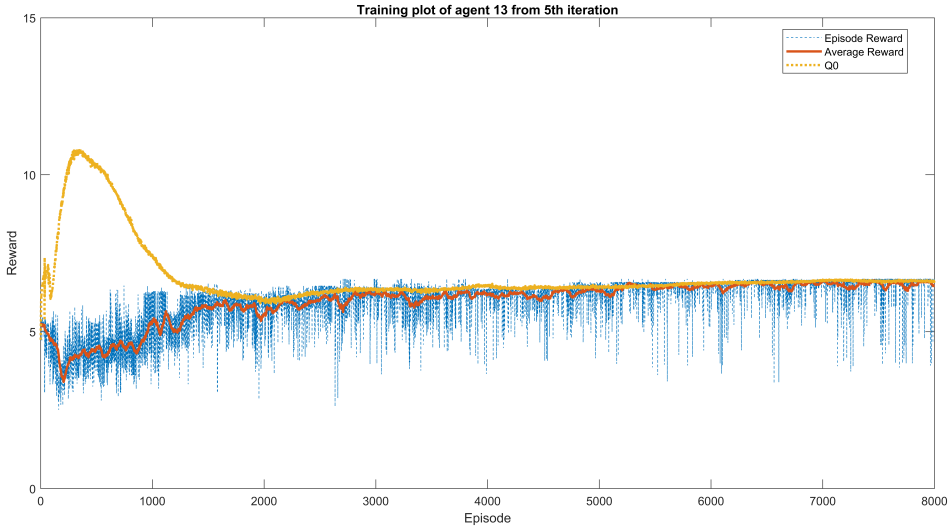


Figure 15: Training graph of best found agent

α	γ	$\epsilon_{\textbf{init}}$	$\epsilon_{\textbf{decay}}$	D	N	L	CT (s)
1.0000e-03	1	0.25	1.0e-10	10	128	5	1674.8

Table 16: Parameters of best found agent

To analyze the found results, statistical data is presented in Table 17. Here it is found that the agent is consistent in found makespan and reward, based on the both the standard deviation and variance.

Variable	C	R
UL	61	6.648
LL	61	6.648
Mean	61	6.648
Var	0	3.22e-30
StD	0	1.79-15

Table 17: Found statistics for makespan and rewards for agent 13 of the fifth iteration

Based on the result found by this agent, a Gantt chart is produced in Figure 16. This Gantt chart again shows a feasible solution in terms of job sequences. Although some improvements might still be possible, the performance has already increased by reducing the makespan by 18 time units. Another consideration here is that the lower bound, which is a makespan of 55, might not be a solution possible to be found with the combinatorial use of the dispatching rules.

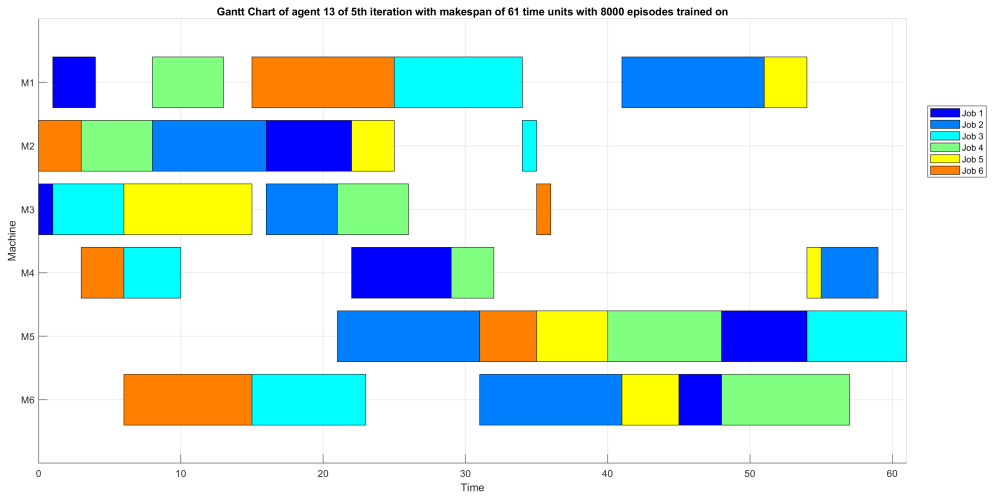


Figure 16: Gantt chart of found solution by agent 13 of the fifth iteration

In Figure 17 a pie chart is presented, where it shows that the agent now uses 9 of the 10 defined actions, leaving out LOR. Although some actions are more preferred than others, it shows the agent learns to balance the action to improve the result of the benchmark agent.

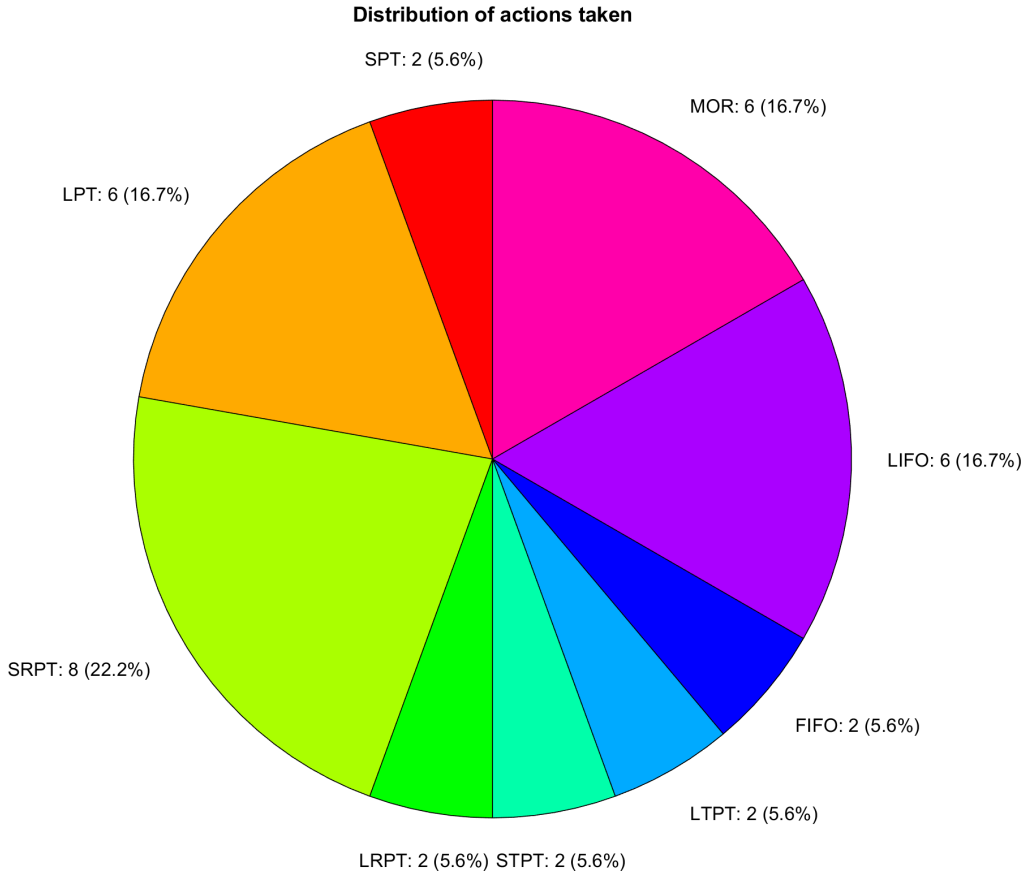


Figure 17: distribution of action taken by final agent

An agent has been found that can solve the ft06 problem, with an increased performance compared

to the benchmark agent while balancing the given actions. The performance now also aligns more with the found methods in literature, given in Chapter 3. Hence, the chosen hyperparameters together with the proposed method will be tested on other standardized problems next.

8.1.3 Testing the hyperparameters on standardized problems

The found hyperparameters with the proposed method are used to train multiple agents, each on a different benchmark problem, to compare the performance with research by Liu [81]. In Table 18 the results of the created agents are given in terms of reward, makespan and computational time. Further details, such as training graphs, are given in Appendix J.

agent	instance	R	C	machines	jobs	CT (h)
1	abz8	5.014	1010	15	20	5.8
2	ft06	6.661	61	6	6	0.76
3	la04	6.473	853	5	10	0.92
4	la09	8.761	1021	5	15	1.38
5	la15	7.582	1636	5	20	1.91
6	la25	5.511	1427	10	15	2.81
7	la35	6.099	3756	10	30	5.60
8	svw01	5.224	3013	10	20	3.79
9	svw15	4.978	8104	10	50	13.16
10	ta01	4.827	1990	15	15	4.26
11	ta31	5.418	3122	15	30	9.71
12	ta41	4.595	3603	20	30	16.66

Table 18: Agents created for standardized problems

To further elaborate on the computational time increase, the input size is set out against the computational time in Figure 18. From this it can be concluded that superlinear growth is seen for the computational time as the input size increase. This is both due to the increase of inputs, creating more connection to the first hidden layer, as well as the increase in total operations, thus steps, needed to solve a job shop when the number of machines and/or jobs grows. When comparing the graph with the data from Table 18 it is found that problems with a higher number of machines also take more computational time with the same number of inputs.

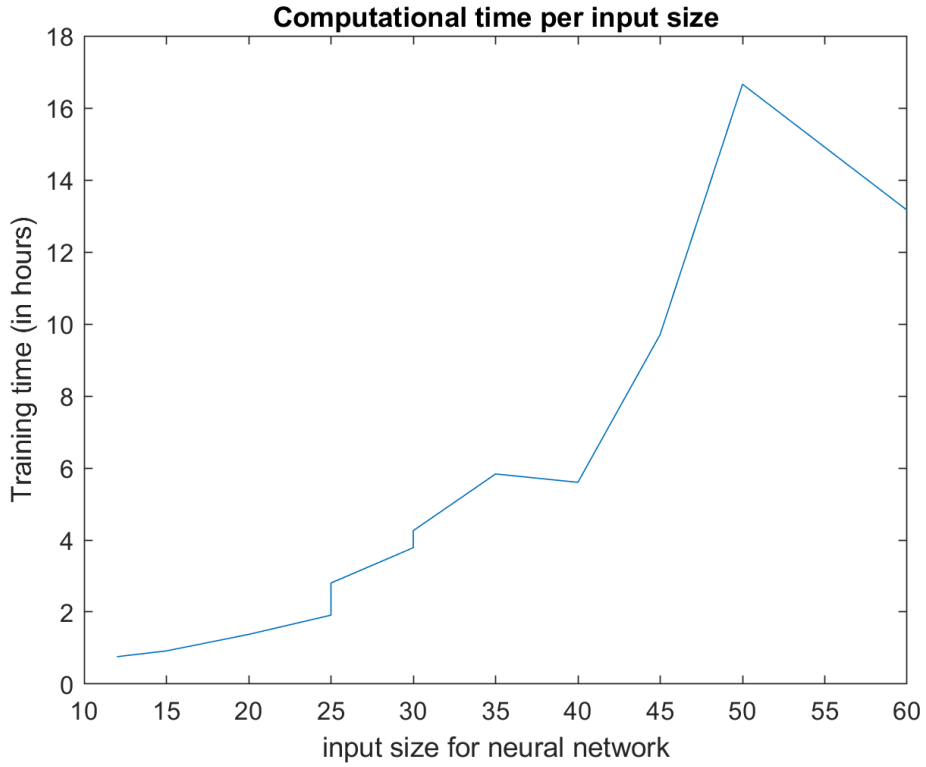


Figure 18: Computational time per input size for training on different problem sizes

From these results it is concluded that with the proposed hyperparameters the ft06 problem can be solved to a reasonable degree as this is the problem the hyperparameters were tuned on. However, for the other problems this is not the case. The bigger the problem, the greater the computational time increases as well the performance decreases. Hence, the agents are not able to solve the problems to a reasonable degree when comparing to other research, given in Table 19. Finally, based on the training graphs given in Appendix J, the bigger the problem becomes, the less compatible the hyperparameters become for the learning process. Hence, such difference in problems, changing the architecture of the neural network, needs change in hyperparameters to accommodate for this change.

		Dispatching rules									DRL methods			Ours
Instance	Size	SPT	LPT	FIFO	LIFO	SWT	LWT	MWKR	LWKR	GA	D3QPN	L2D	P3OR	DQN
ft06	6 × 6	84	73	65	70	83	62	59	68	55	59	64	57	61
orb07	10 × 10	504	520	502	500	512	487	482	519	426	438	470	415	
la04	10 × 5	711	832	758	741	864	712	706	885	617	635	736	624	853
la09	15 × 5	1045	1183	997	1073	1135	1012	973	1149	954	978	1015	952	1021
la15	20 × 5	1339	1612	1282	1345	1587	1312	1258	1598	1128	1241	1295	1235	1636
la25	15 × 10	1297	1374	1283	1352	1471	1336	1172	1425	1160	1153	1204	1148	1427
la35	30 × 10	2133	2324	2004	2215	2368	2274	1962	2287	2019	1994	2085	1927	3756
abz8	20 × 15	929	949	879	938	957	936	810	992	744	778	861	736	1010
yn1	20 × 20	1196	1115	1123	1177	1214	1163	1045	1205	926	1053	1121	997	
swv01	20 × 10	1737	2145	1889	2123	2005	1923	1971	1838	1732	1712	1845	1645	3013
swv15	50 × 10	3501	4404	3603	3573	4133	4026	4905	3919	3422	3431	3516	3328	8104
ta01	15 × 15	1462	1701	1830	1627	1712	1523	1438	1737	1457	1405	1521	1412	1990
ta31	30 × 15	2335	2417	2436	2417	2754	2218	2143	2962	2237	2116	2231	2044	3122
ta41	30 × 20	2499	2925	2973	2760	2814	2609	2538	2976	2739	2475	2613	2387	3603
ta51	50 × 15	3856	3880	3717	3391	3702	3624	3567	3596	3250	3151	3224	3018	
ta61	50 × 20	3606	3989	4046	3870	3827	3568	3376	4073	3658	3365	3441	3256	
ta71	100 × 20	6232	7038	6704	6767	6735	6524	5938	6993	6524	5938	6993	5624	

Table 19: Table comparing proposed DQN with other methods from Liu [81]

8.1.4 Conclusion of the first experiment

To summarize, the goal of these tests was to assess if the DQN method with proposed environment work, as well as assessing the transferability of the hyperparameters to other problems with a neural network that changes based on the number of machines and jobs per problem.

As per the DQN working with the environment, it is concluded that these work as expected. The DQN is able to learn as well as the environment processing the taking actions by the DQN correctly, resulting in feasible schedules.

From hyperparameter tuning it was found that the performance can be increased, while maintaining a comparable computational time, of below one hour to solve the ft06 problem. Also it was learned that a lot of data needs to kept track of. This all needs to be stored, for losing it will take a lot of time to redo each test. Finally, a lot of behavior for hyperparameter tuning can be assessed in the earlier stages of training, around 500 to 1000 episodes. Afterwards, agents mostly stabilize their training behavior. Hence, earlier hyperparameter tuning can be done on less episodes, reducing the time for hyperparameter tuning and/or increasing the number of hyperparameter combinations that are tested.

Finally, training different agents per problem with the found hyperparameters gave unwanted results. The performance was below expectation, as well as taking more time (up to 16 hours) for problems with a higher number of jobs and machines. Hence it is assumed that changing the architecture of neural network, in this case the size of the input matrix, greatly influences applicability of hyperparameters as well as the computational time needed. Hence the input size of the neural network should be fixed on a set number, to possibly improve applicability of hyperparameters to problems of different sizes.

For the next test, the hyperparameter tuning is redone, fixing the input size on 10 jobs and 20 machines to be able to assess multiple sizes of job shops. For the hyperparameter tuning, the first 500 episodes of training are done for the early stages of tuning, thus saving time per set of hyperparameters.

8.2 Second iteration: static job shop scheduling

In this second iteration, the recommended changes are used to redo hyperparameter tuning and execute further experiments. The goal is to assess the generalization capabilities when agents are trained on multiple problems, as well as using transfer learning to improve performance and reduce training time, utilizing gained knowledge of pre-trained agents.

8.2.1 Benchmark of second iteration

Again, an agent is created to compare the performance to. Here, the only change in comparison to Chapter 8.1.1 is the number of inputs, from 6 jobs and machines to 10 jobs and 20 machines.

In Figure 19 the training graph is shown. When compared to the benchmark in Chapter 8.1.1, a big difference found is that at the start of training the agent already performs quite well. However, over time this performance reduces. Although this reduction, the agent does seem to learn an optimal policy as the Q0 value and average found reward become somewhat in line with one another, much like the first benchmark agent.

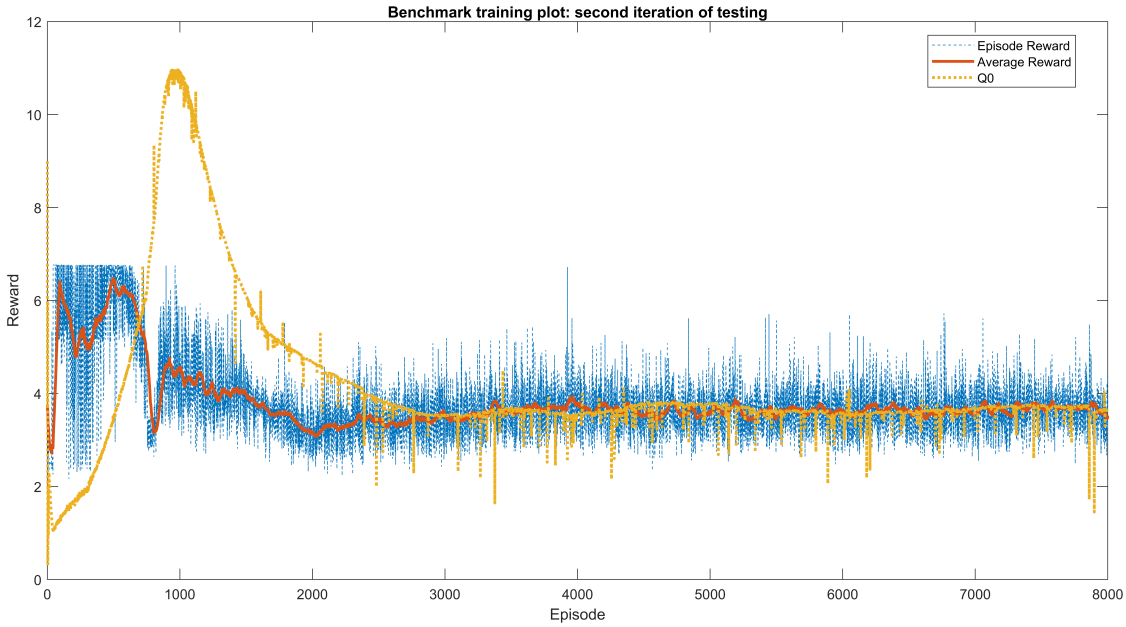


Figure 19: Training graph of benchmark agent for second iteration

In terms of actual performance, the statistical results have been given in 20. Here it is found that in comparison to the previous benchmark, the performance is greatly reduced as expected from the change in learning behavior. Although the agents performance is reduced, the agent is able to perform consistently, thus learning an optimal policy. The agent is able to be trained in 3407.5 seconds, or 56 minutes and 47 seconds, keeping a comparable computational time comparable to earlier found solutions.

metric	**R**	**C**
UL	3.31	123
LL	3.31	123
Mean	3.31	123
Var	0	0
StD	0	0

Table 20: statistical results of benchmark agent of second iteration

The Gantt chart of the best found solution is given in Figure 20. As expected based on the found makespan, a lot of gaps can be identified which should be easily solvable, even manually.

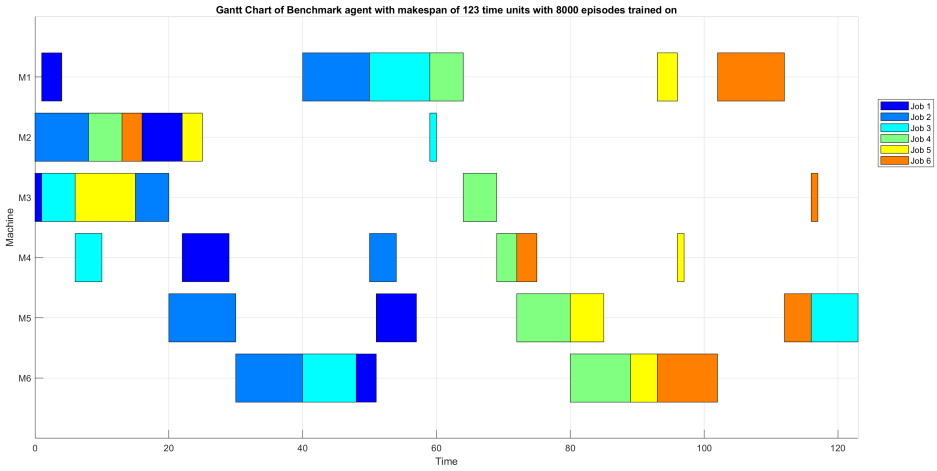


Figure 20: Gantt chart of benchmark agent of second iteration

Finally, the distribution of chosen actions is given in Figure 21. Here it is found that the benchmark agent does not balance the possible actions to come to a feasible solution, only using LRPT and MOR.

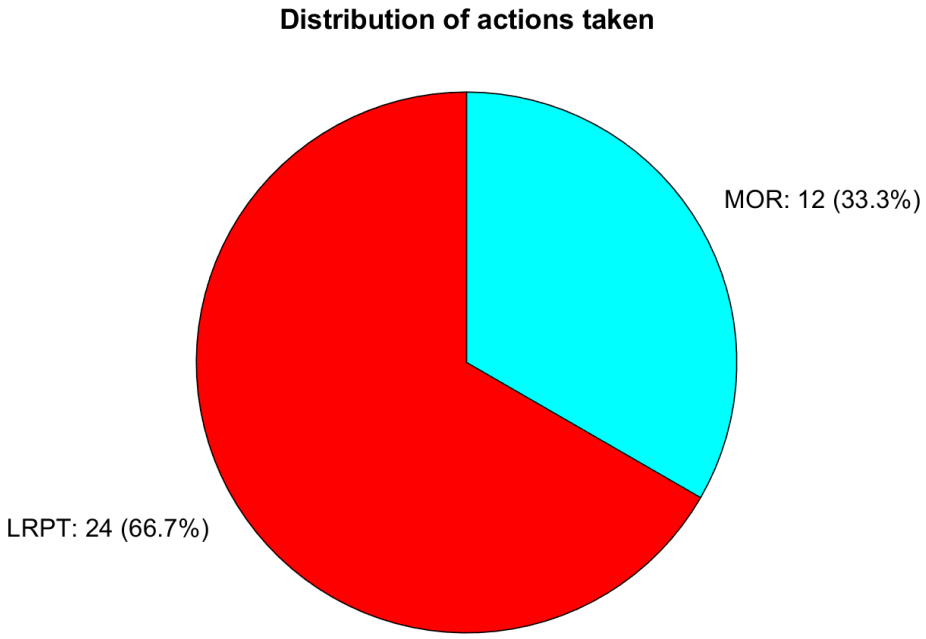


Figure 21: Distribution of chosen actions by benchmark agent

As the benchmark is now defined, random hyperparameter tuning is done.

8.2.2 Random hyperparameter tuning

The hyperparameter tuning method is adjusted according to recommendations from the previous iteration, again tuning for the ft06 problem. Here, the training graphs found are categorized on behavior and based on the hyperparameters found for each category, the hyperparameters are adjusted

per iteration. With this adjusted approach, more hyperparameters can be tested and assessed in less time. All these steps are further defined and executed in Appendix K. The hyperparameter ranges for each iteration are given in Table 21. Here, the reward scaling factor is fixed to be 10.

It.	α	γ	ϵ_{init}	ϵ_{min}	ϵ_{decay}	**N**	**L**	**E**
1	1e-1, 1e-4, 1e-7	0.1, 0.5, 0.99	1e-7, 1e-4, 1	1e-13, 1e-7, 0.01	1e-10 1e-6 1e-2	64 512 2048	1 4 7	500
2	1e-3, 1e-4, 1e-5	1e-4, 0.5, 0.99	1e-7, 1e-4, 1	1e-13, 1e-7, 0.01	1e-10 1e-6 1e-2	64 128 256	1 to 5	1000
3	1e-3, 1e-4, 1e-5	0.75 0.875 0.99	1e-7, 1e-4, 1	1e-13, 1e-7, 0.01	1e-10 1e-6 1e-2	64 128 256	1 to 5	1000
4	1e-3 1e-4 1e-5	0.75 0.875 0.99	0.1 0.5 1	1e-13, 1e-7, 0.01	1e-3 5e-4 1e-4	64 128 256	3 to 5	1000
5	1e-4, 1e-5	0.99	0.1 0.5 1	1e-13, 1e-7, 0.01	5e-4 1e-4	64 128 256	3 to 5	1000
6	1e-4, 1e-5	0.99	0.5 0.75 1	1e-13, 1e-7, 0.01	1e-4	64 128 256	3 to 5	8000
7	1e-5	0.99	0.75 1	1e-13, 1e-7	1e-4	64 128 256	3 to 5	8000

Table 21: Hyperparameter settings for each iteration per the second iteration test

From the results of the eventual chosen agent given in Table 79 it is found that this agent improves greatly on the both the benchmark agents. Although the reward is slightly lower to the other agent found by hyperparameter tuning, the makespan is reduced by 2. Hence it could be said that performance is found to be comparable. With these changes in hyperparameters, the computational time changes to 6394.1 seconds, or one hour, 46 minutes and 34 seconds. This is almost double of earlier found solutions, which is to be expected as the number of inputs is more than doubled (from 12 to 30).

metric	**R**	**C**
UL	6.625664	59
LL	6.625664	59
Mean	6.625664	59
Var	0	0
StD	0	0

Table 22: statistical results agent 6 iteration 7

In Figure 22 the Gantt chart is given depicting the feasible schedule found by the agent. From this Gantt chart it is hard to find any big gaps in scheduling as well as direct improvements. The agent finds a comparable performing solution to found results in literature, given in Chapter 3.

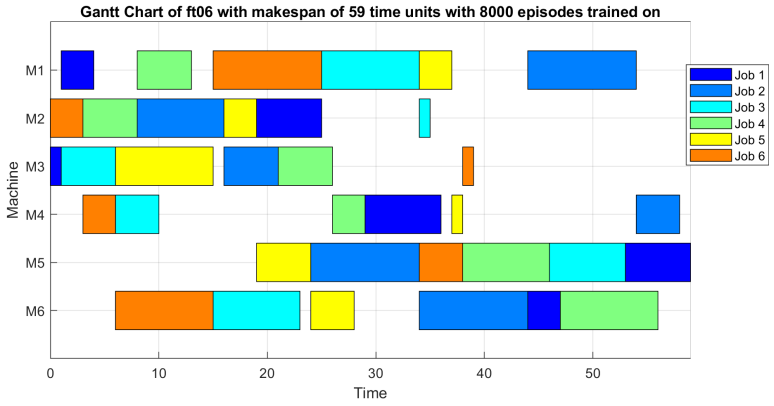


Figure 22: Gantt chart of solution found by agent 6 of seventh iteration

In Figure 23 a pie chart is shown of the chosen actions. From this pie chart it can again be concluded that the agent uses more of the given actions in comparison to the benchmark agent, thus exploring possible actions and finding a feasible solution combining the given actions. This agent ops to only use seven of the 10 optional actions, showing that while balancing is needed, not all actions are necessary to use.

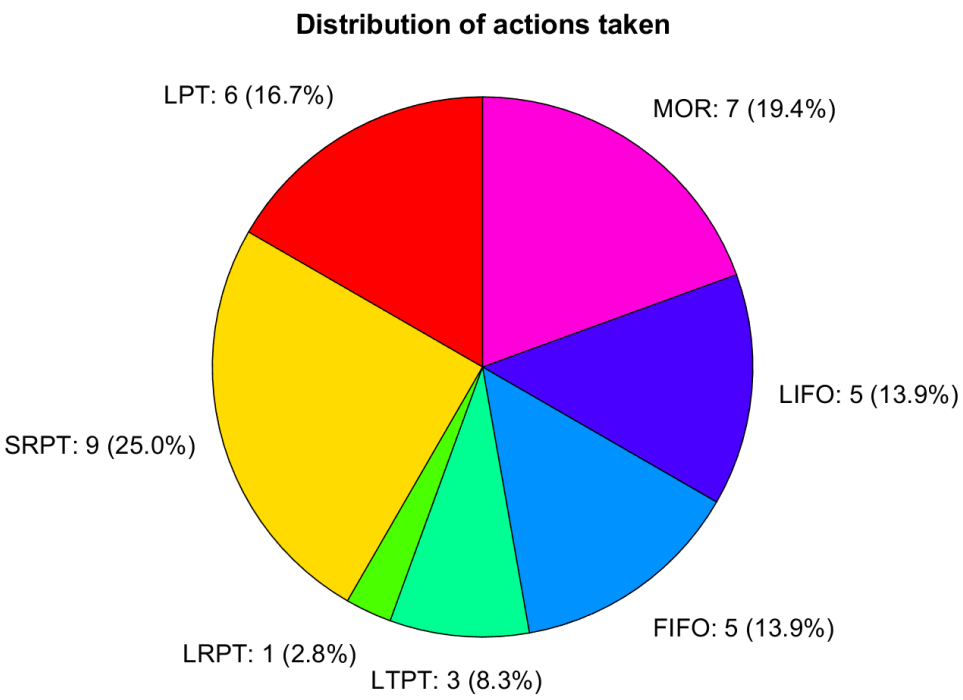


Figure 23: Pie chart of chosen actions by best found agent

With the hyperparameters, agents will now be trained on multiple problems, assessing the generalization of the DQN method as well as the transferability of hyperparameters on other problems with a fixed neural network architecture.

8.2.3 Training on multiple problems

In this section, agents are trained on a different number of problems, as well as different job shops. Three job shops are defined with 5 jobs, while differing in having 10, 15 or 20 machines respectively. Per job shop type, 50 problems are generated. For each job shop type, agents are trained on 1, 5, 10, 25 and 50 of the generated problems, in numerical order. Thus, no randomization for choosing problems is done. Using this method, the performance of the created agents is assessed on generalization, comparing the agents in overall performance as well as comparing them to the agent trained on a single problem.

As the proposed experimental method, the created agents are tested on the problems they were trained to solve, problems with the same parameters which they have not been trained on, as well as problems with a different processing time distribution, in this case being a uniform distribution between 20 and 99 instead of between 1 and 10. All of the results of these tests are given in Appendix L. A summary of the results is given in Table 23. Here, the results are expressed a percentage, being the difference of the found makespan by the solver in comparison to the actual lower bound of the problem found. The percentages are an average of the overall found difference per problem set. The agents here are defined by the number of problems they were trained on, and per job shop type a different agent was trained for the same number of problems.

From the table it is found that training agents on multiple problems does increase overall performance for each job shop type. A difference is found in the 10 and 15 machines where the agent trained on 25 problems performs better compared to 20 machines, where the agents trained on 50 problems generalizes better. However, it is seen that dispatching rules defined, mainly LRPT and MOR, still are able to outperform the created agents.

	10 by 5			15 by 5			20 by 5		
solver	**trained**	**untrained**	**U[20,99]**	**trained**	**untrained**	**U[20,99]**	**trained**	**untrained**	**U[20,99]**
SRPT	-246%	-246%	-258%	-262%	-256%	-274%	-268%	-271%	-281%
LRPT	-35%	-36%	-33%	-25%	-26%	-23%	-18%	-20%	-17%
SPT	-172%	-156%	-177%	-186%	-185%	-185%	-197%	-198%	-186%
LPT	-181%	-183%	-187%	-201%	-202%	-201%	-212%	-215%	-201%
LTPT	-250%	-240%	-264%	-267%	-260%	-276%	-268%	-278%	-278%
STPT	-246%	-246%	-258%	-262%	-256%	-274%	-268%	-271%	-281%
FIFO	-249%	-252%	-262%	-262%	-259%	-271%	-272%	-275%	-283%
LIFO	-242%	-243%	-254%	-266%	-256%	-275%	-270%	-276%	-281%
LOR	-249%	-252%	-262%	-262%	-259%	-271%	-272%	-275%	-283%
MOR	-29%	-33%	-28%	-24%	-25%	-22%	-18%	-21%	-16%
1 problem	-140%	-125%	-159%	-97%	-97%	-151%	-171%	-174%	-23%
5 problems	-129%	-118%	-176%	-113%	-112%	-147%	-154%	-157%	-213%
10 problems	-79%	-95%	-194%	-123%	-129%	-124%	-86%	-102%	-190%
25 problems	-57%	-65%	-189%	-55%	-57%	-237%	-145%	-152%	-26%
50 problems	-91%	-87%	-64%	-121%	-117%	-140%	-71%	-73%	-80%

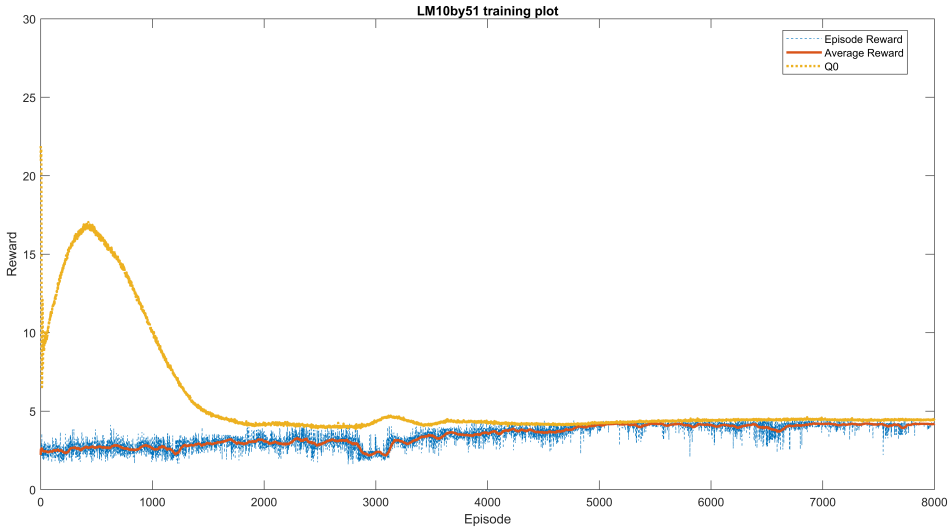
Table 23: Overview of performance per dispatching rule and agents, in comparison to the found lower bound

As per the training graphs, partly shown in Figure 24, further given in Appendix L, it is found that as the number of machines increases, the difference between Q0 estimation and found average reward becomes bigger, thus the hyperparameters being less suitable for problems different to the 6 by 6 problem. Hence, a second iteration is done on problems with 6 jobs and 6 machines, to further assess performance of training on multiple problems.

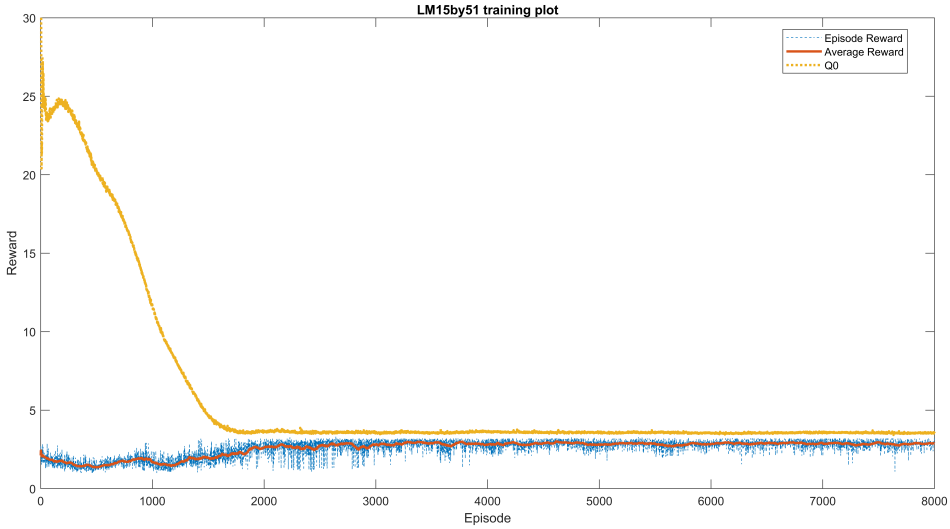
The training time of these agents are given in Table 24. Here it is found that per job shop type some differences in computational time are found when trained on a different number of problems, but is not significant. Secondly, as expected due to the increase of steps per episode, the computational time increases as the number of machines increases.

Trained on	**10 machines**	**15 machines**	**20 machines**
1 problem	9005	13789	16682
5 problems	8748	10935	14727
10 problems	7499	9563	14696
25 problems	7791	9709	12642
50 problems	7483	13064	14675

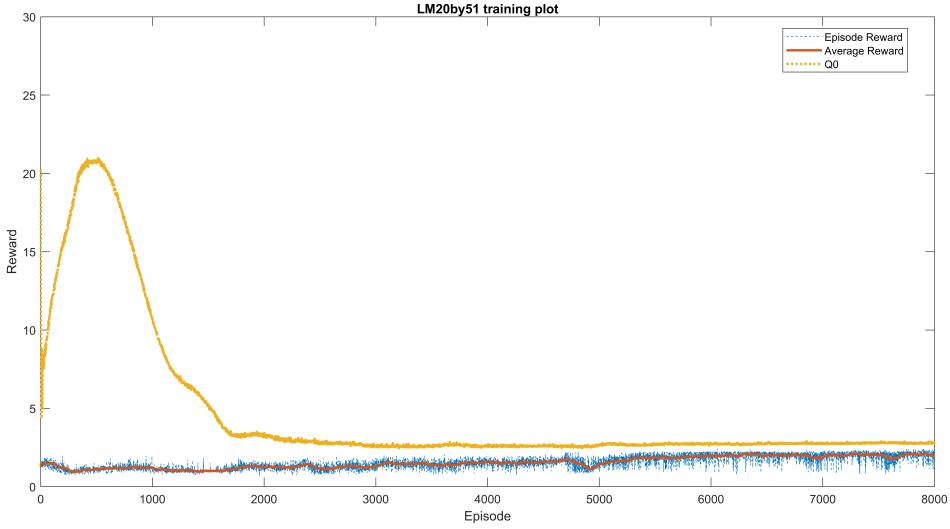
Table 24: Comparison of computational time (in seconds) per agent



(a) Learning behavior with 1 problem and 10 machines



(b) Learning behavior with 1 problem and 15 machines



(c) Learning behavior with 1 problem and 20 machines

Figure 24: Learning behavior of agents trained on 1 problem with different machines

Second iteration: 6 by 6 problems

From the found results, explained in more detail in Appendix L.2, it was found that these agents perform comparably to what was found for the other sizes of problems. However, as expected the training graphs show better cohesion to the hyperparameters than of the previous trained on problems, as depicted in Figure 25. Here, the Q0 value lies closer to the found average reward, showing better applicability of hyperparameters.

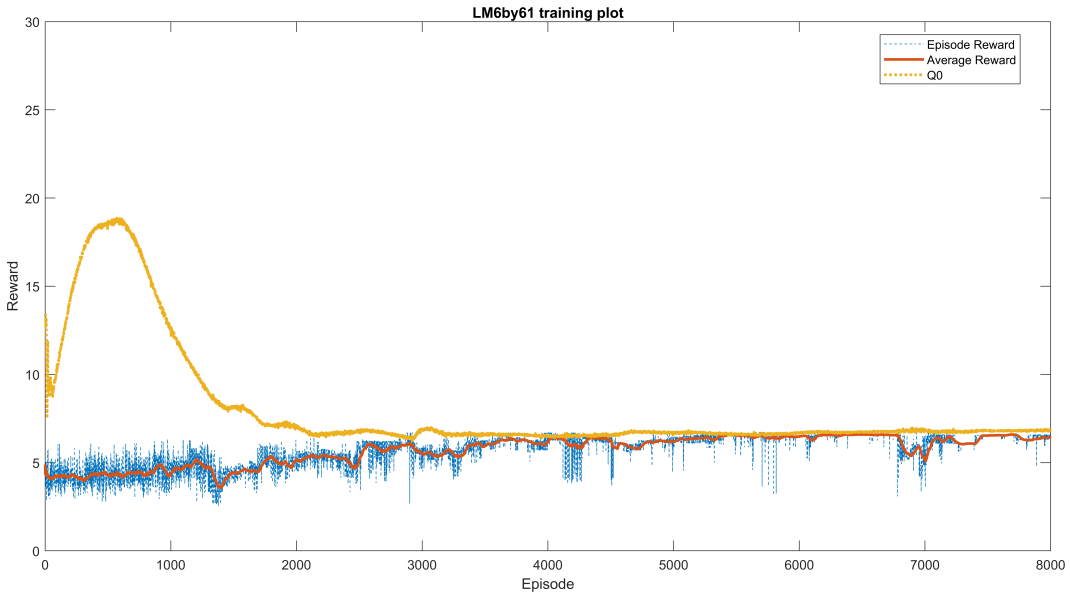


Figure 25: Training graph of agent trained on 1 problem on a 6 by 6 problem

In Table 25 the same comparison has been made as in the previous section. Here, the found results are comparable to the ones found for the 10 and 15 machine problems. Although reduced in performance, the LRPT and MOR dispatching rule still are able to outperform each of the agents. Next to that, it can now be concluded that data outside of the training scope, being a changed distribution of processing times, greatly influences the performance. Hence it can be concluded that it is necessary for the agent to be trained on specific job shop data they eventually have to solve.

solver	**trained**	**untrained**	**U[20,99]**
SRPT	-251%	-247%	-270%
LRPT	-59%	-60%	-63%
SPT	-165%	-173%	-182%
LPT	-200%	-198%	-207%
LTPT	-259%	-245%	-282%
STPT	-251%	-247%	-270%
FIFO	-255%	-259%	-276%
LIFO	-253%	-257%	-274%
LOR	-255%	-259%	-276%
MOR	-49%	-50%	-54%
1 problem	-143%	-143%	-95%
5 problems	-104%	-102%	-80%
10 problems	-106%	-129%	-180%
25 problems	-77%	-76%	-229%
50 problems	-84%	-91%	-203%

Table 25: Overview of performance per dispatching rule and agents, in comparison to the found lower bound for 6 by 6 problems

For these tests the computational time found is comparable per agent, given in Table 26, and is

about equal to that of the hyperparameter tuned agent. Hence it is concluded that the computational time does not change when comparable job shop problems are solved.

# problems	1	5	10	25	50
CT (s)	6641	6113	5815	5503	5663

Table 26: Computational time per trained agent

At this point of the research, the issue mentioned in Chapter 6.2.2 has been found, and is taken into account for comparing the results from the dispatching rules with the results found by the agent for the ft06 problem. For consistency, the environment is not updated. To compare the agents created with both the benchmark agent as well as the agent found with hyperparameter tuning, the agents and dispatching rules are tested on solving the ft06 problem with an optimal makespan of 55. In Table 27 the results per dispatching rule as well as agent is given, with both the used environment as well as the updated one. Looking at the used environment results, it is found that again only LRPT and MOR outperform the created agents. Although these agents improve upon the benchmark agent, the agent found by hyperparameter tuning as well as other standard methods outperform these agents. Next, the updated environment shows that both the dispatching rules and generalized agents are able to improve their performance due to the more efficient scheduling, resulting in comparable and sometimes better performance than the dispatching rules. Although no real conclusion can be drawn from those results, it might be possible to further improve general performance when hyperparameter tuning is done for the new environment with improved scheduling efficiency.

	Used environment		Updated environment	
action	**C**	**% of lowest possible C**	**C**	**% of lowest possible C**
SRPT	154	-180%	94	-71%
LRPT	74	-35%	67	-22%
SPT	109	-98%	83	-51%
LPT	129	-135%	79	-44%
LTPT	160	-191%	67	-22%
STPT	154	-180%	94	-71%
FIFO	152	-176%	71	-29%
LIFO	170	-209%	86	-56%
LOR	152	-176%	71	-29%
MOR	60	-9%	60	-9%
1 problem	92	-67%	68	-23%
5 problems	89	61%	68	-23%
10 problems	105	-91%	79	-43%
25 problems	119	-116%	69	-25%
50 problems	83	-51%	76	-38%

Table 27: Results of dispatching rules on ft06 problem in two different static environments

Concluding from these results, it can be said that although general performance can be improved in this way, the actual specific problem solving does leave to be desired. Simple and fast dispatching rules can outperform the created agents. However, as the scheduling of these agents was inefficient, it might be possible to further improve general performance as found with deploying the trained agents in a more efficient environment, by hyperparameter tuning for a new environment.

8.2.4 Transfer learning

As now the conclusion can be made that these agents can generalize problems to a certain point, transfer learning is proposed to see how these agents can be retrained on a new specific problem, using the ft06 problem to be able to compare results with the benchmark and hyperparameter tuned agents. Here, the goal is to see how the agents can increase performance on the ft06 problem as well as the found result by the hyperparameter tuned agent for this section.

The tests are divided into three different parts. The first is deploying the agents instantly into new training, without adjusting any parameters. Secondly, the experience buffer used for the training is emptied, making sure the neural network is only updated with states, rewards and actions from the specific problem that is being trained on. Finally, the experience buffer is emptied as well as the hyperparameters defined by hyperparameter tuning are reset, as they can be changed for example by optimization algorithms during training, or the decay of ϵ. With these three tests, it can be determined in what way the learning behavior changes for the pre-trained agents with transfer learning. A more detailed analysis on this transfer learning section is given in Appendix M.

Test one: instantly deploying agents

For this test, the agents are instantly deployed to retrain. Hence no parameters are changed and the agent will also still update the neural network with old experiences as well as new ones found in the experience replay buffer. As per Table 25 of Chapter 8.2.3, the agents trained on 5, 25 and 50 problems are expected to give the best results based on the generalization capabilities, which are represented by agent 2, 4 and 5. The statistical results of transfer learning are given in Table 28, where it is found that each agent is able to find consistent results, although with different levels of performance. The general knowledge gained by the agents ensured that they are able to further explore a single problem and all are able to improve upon their performance in comparison to Table 27 in Chapter 8.2.3.

Agent	**Agent 1**		**Agent 2**		**Agent 3**		**Agent 4**		**Agent 5**	
metric	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**
UB	5.71	66	6.24	60	4.90	76	6.76	60	6.31	60
LB	5.71	66	6.24	60	4.90	76	6.76	60	6.31	60
Mean	5.71	66	6.24	60	4.90	76	6.76	60	6.315	60
Var	0	0	0	0	8.05E-31	0	3.22E-30	0	8.05E-31	0
StDev	0	0	0	0	8.97E-16	0	1.79E-15	0	8.97E-16	0

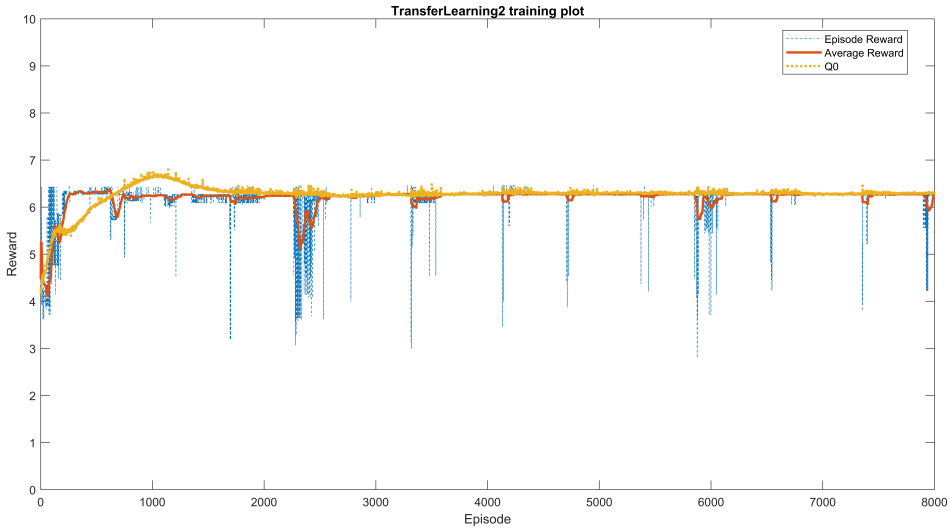
Table 28: Statistical results of 50 runs on ft06 from transfer learning agents

In Figure 26 the training graphs of agents 2, 4 and 5 are given, as these are the best performing agents. Here it is found that although agent 4 looks less stable in terms of found rewards, it is able to find the highest possible reward consistently as per this training. Also, little exploration is seen in all graphs due to the epsilon already having been decayed and has not been reset.

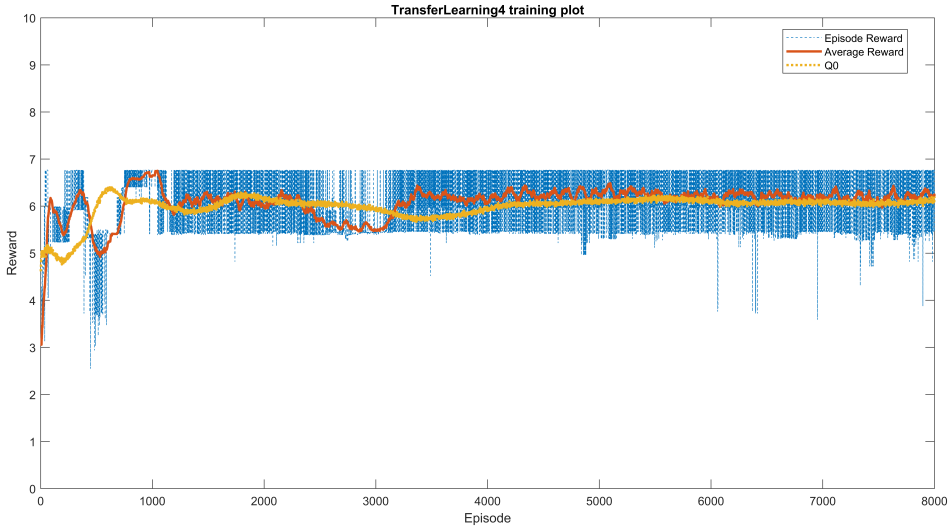
Finally, the computational time of these agents are given in Table 29 where it is found that this method actually increases the training time in this setup. As the settings currently are unknown, not being able to determine them from MATLAB, no real conclusion can be drawn here other than the increase in exploitation also increases computational time. Although this increase, it can be seen that agent 2 becomes stable between 2000 and 3000 episodes, hence a decrease in episodes might be feasible. This will be assessed after the other two tests are completed.

Agent	1	2	3	4	5
CT (s)	10162	9760	9402	8996	10267

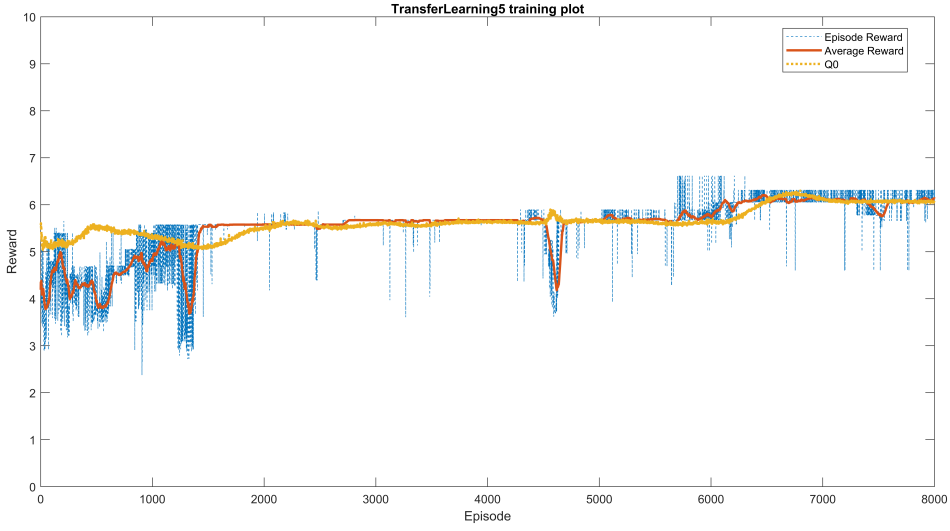
Table 29: Computational time per trained agent with transfer learning



(a) Learning behavior of agent 2 (5 problems)



(b) Learning behavior of agent 4 (25 problems)



(c) Learning behavior of agent 5 (50 problems)

Figure 26: Learning behavior of agents 2, 4 and 5 with transfer learning

Test two: emptying experience buffer

As the experience buffer is used to train the neural network, adjusting the weights and biases, in this test the experience buffer is emptied to ensure only learning of scenarios found in the new problem. Here, unexpected results are found, comparing the previous results to the results found in Table 30. Here it is seen that agents 2, 3 and 4 are able to improve upon their found reward, while the performance of agents 1 and 5 are reduced. It could be said that the general knowledge gained from training on 5, 10 and 25 problems is enough to improve without changing any other hyperparameters, while the specific knowledge of 1 problem as well as the general knowledge of 50 problems is not enough to learn a good policy without adjusting any hyperparameters.

Agent	**Agent 1**		**Agent 2**		**Agent 3**		**Agent 4**		**Agent 5**	
metric	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**
UB	5.15	75	6.46	60	5.45	68	6.77	60	4.53	80
LB	5.15	75	6.46	60	5.45	68	6.77	60	4.53	80
Mean	5.15	75	6.46	60	5.45	68	6.77	60	4.53	80
Var	0	0	7.24E-30	0	0	0	3.22E-30	0	3.22E-30	0
StDev	0	0	2.69E-15	0	0	0	1.79E-15	0	1.79E-15	0

Table 30: Statistical results of 50 runs on ft06 from transfer learning agents with emptied experience buffer

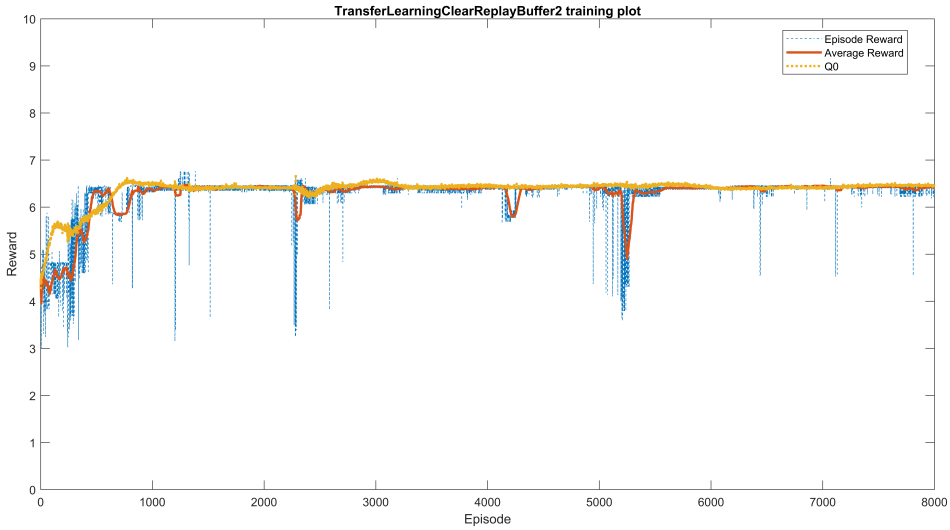
Looking at the change in learning behavior, training graphs for agents 2, 4 and 5 are given in Figure 27. Here it is found that agents 2 and 4 shows quite comparable behavior to before, with agent 2 able to find a slight increase in reward. Agent 5 however shows real different behavior in the Q0 not aligning and the found reward greatly reduced by 2 points.

Finally, computational time for each agent is given in Table 31. Here it is found that although slightly lower, probably due to the memory management being less in need of resources, the change in time is not significant.

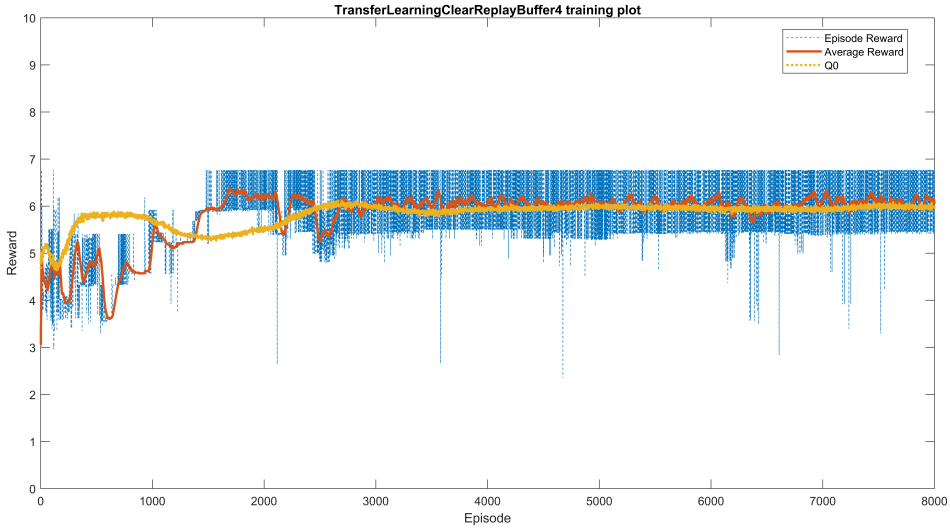
Agent	1	2	3	4	5
CT (s)	9788	9277	9253	8408	9846

Table 31: Computational time per trained agent with transfer learning with emptied experience buffer

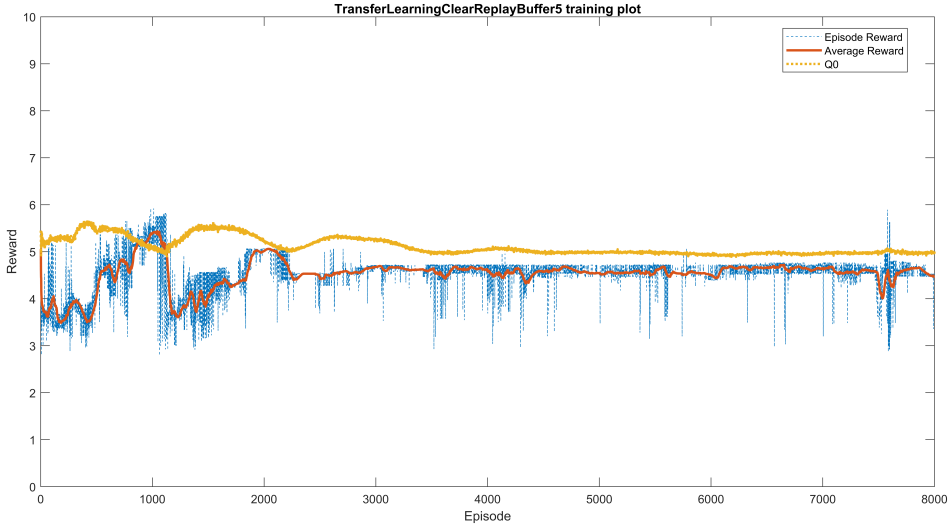
From this second test it is concluded that agents that are trained on 5 to 25 problems are able to improve upon the gained knowledge, while an agent with 1 problem trained on might have too specific knowledge and the agent with 50 problems might have too general knowledge for this method too be applicable. Hence, the next test will assess the difference a pre-trained neural network makes when all other variables and hyperparameters are reset.



(a) Learning behavior of agent 2 (5 problems)



(b) Learning behavior of agent 4 (25 problems)



(c) Learning behavior of agent 5 (50 problems)

Figure 27: Learning behavior of agents 2, 4 and 5 with transfer learning and emptied experience buffer

Test three: resetting all but the neural network

Finally, the agents hyperparameters are all reset and the experience buffer is emptied, while maintaining knowledge gained by the neural network from pre-training. In Table 32 the statistical data shows consistency in found results, and overall each agent is able to perform in comparable manner to each other. However, the performance of agents 2 and 4 decrease with this method, while agents 1, 3 and 5 improve upon the earlier test.

Agent	**Agent 1**		**Agent 2**		**Agent 3**		**Agent 4**		**Agent 5**	
metric	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**
UB	6.57	59	6.35	60	6.50	61	6.60	60	6.56	61
LB	6.57	59	6.35	60	6.50	61	6.60	60	6.56	61
Mean	6.57	59	6.35	60	6.50	61	6.60	60	6.56	61
Var	0	0	3.22E-30	0	3.22E-30	0	0.00E+00	0	3.22E-30	0
StDev	0	0	1.79E-15	0	1.79E-15	0	0.00E+00	0	1.79E-15	0

Table 32: Statistical results of 50 runs on ft06 from transfer learning agents with cleared hyperparameters and emptied experience buffer

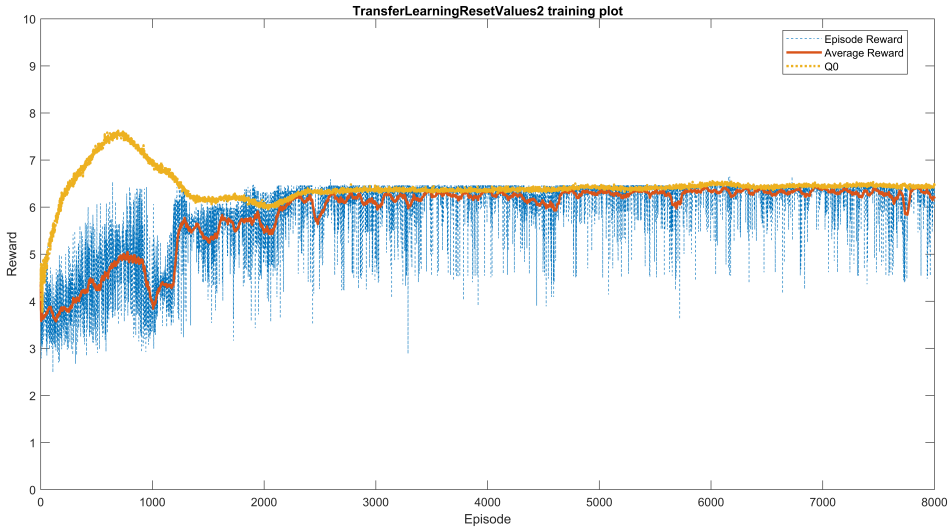
The training graphs are given in Figure 36, where the agents 2, 4 and 5 are depicted again. Due to the change in hyperparameters and the emptied experience buffer, the agents show quite similar behavior. From this it can be concluded that quite similar results can be found when retraining with only the neural network being different.

In Table 33 the computational time per agent is given. Again, a small reduction in computational time is found, except for agent 1.

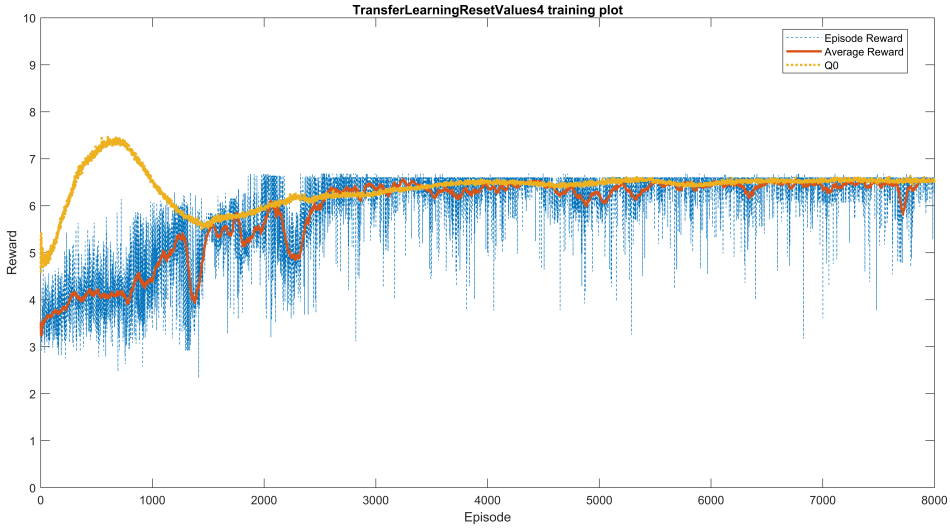
Agent	1	2	3	4	5
CT (s)	9837	8827	7484	7524	8694

Table 33: Computational time per trained agent with transfer learning, emptied experience buffer and reset hyperparameters

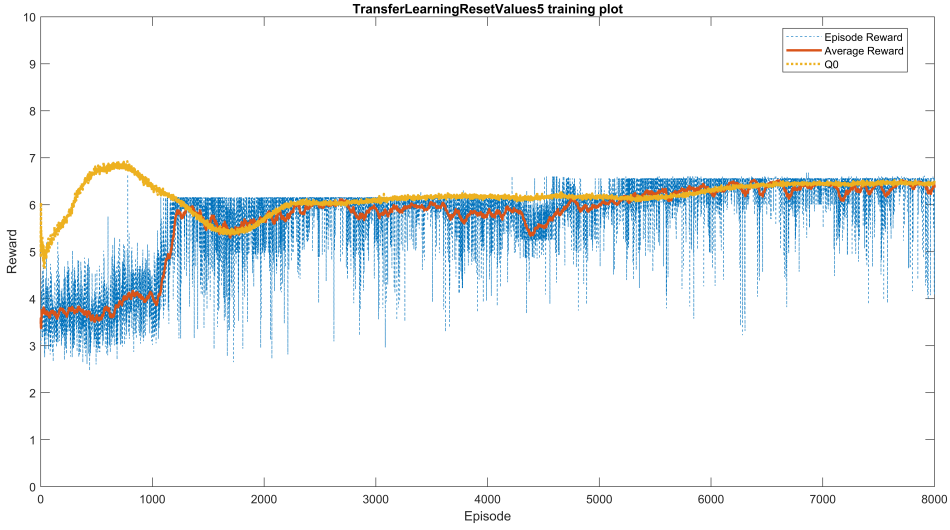
From this third test, it can be concluded that resetting all values changes the behavior of transfer learning to show behavior comparable to the hyperparameter tuned agent, which is to be expected. From this it can be concluded that although not necessary, hyperparameter tuning for transfer learning could possibly further improve the performance and learning behavior.



(a) Learning behavior of agent 2 (5 problems)



(b) Learning behavior of agent 4 (25 problems)



(c) Learning behavior of agent 5 (50 problems)

Figure 28: Learning behavior of agents 2, 4 and 5 with transfer learning, emptied experience buffer and reset hyperparameters

Assessing computational time reduction

As determined in previous tests, the agent trained on 25 problems performs best in each test. Also, the training result of test 2 shows the highest found reward. Based on the training graph from Figure 27b it is found that at around 3000 episodes, the agent does not change its policy much. Hence, to assess if this method can indeed lower the computational time, the agent will be retrained on 1000, 2000, 3000 and 4000 of episodes instead of 8000, to determine if the performance is equal while lowering episodes, thus lowering computational time. In Figure 29 the training graph is shown, showing equal behavior to the earlier found training behavior of this agent.

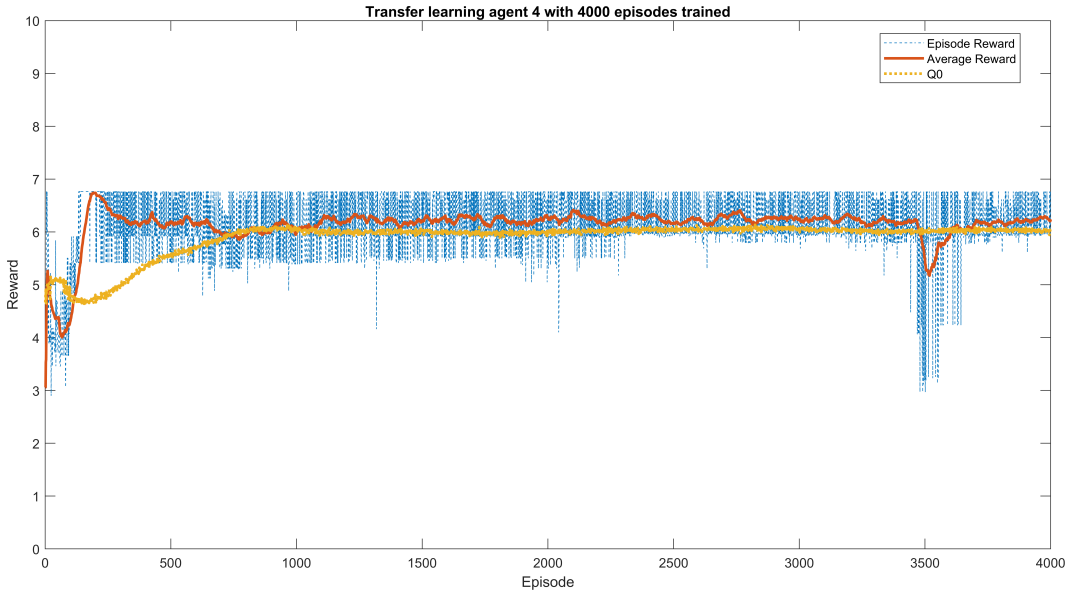


Figure 29: Shortened training graph with 4000 episodes of agent 4 of transfer learning with emptied experience buffer

To compare each 1000 steps in terms of performance, this is represented in Table 34. Here it is found that after 4000 episodes, the agent is able to perform the same as after 8000 episodes assessed previously.

# episodes	**1000**		**2000**		**3000**		**4000**	
metric	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**
UL	5.92	66	5.24	80	5.93	64	6.77	60
LL	5.92	66	5.24	80	5.93	64	6.77	60
Mean	5.92	66	5.24	80	5.93	64	6.77	60
Var	3.22E-30	0	8.05E-31	0	0	0	3.22E-30	0
StD	1.79E-15	0	8.97E-16	0	0	0	1.79E-15	0

Table 34: Results of different number of episodes trained on for agent 4

To compare the computational time for each number of episodes, this is represented in Table 35. Here it is found that with training on 4000 episodes, which increases performance, also reduces the computational time necessary. Hence, transfer learning could be used as a tool to speed up training time while also improving performance.

# Episodes	1000	2000	3000	4000
CT (s)	858	1774	2990	4530

Table 35: Computational time for agent 4 with different number of episodes

From the found results it is given that in general, transfer learning using a pre-trained agent on a

new problem improves the performance of the agent on a new problem. However, comparing to the hyperparameter tuned agent, only the agent trained on 25 problems is able to improve upon the found rewards for the first two tests, while only slightly improving. Based on the fact that this agent showed best generalization capabilities, this outcome seems logical.

Looking at computational time, transfer learning actually increases it. However, it is also shown that the number of episodes can be reduced to further improve computational time. Hence, transfer learning is a powerful tool in DRL to reduce computational time for problem solving as well as improving performance.

8.2.5 Conclusion on second iteration

From these tests it is found that extensive hyperparameter tuning can be done in less time by only using the first 1000 episodes, in this case, to be able to assess more hyperparameter combinations in less or equal amount of time. The benchmark showed a decrease in performance when the input size of the neural network was increased from 6 jobs and 6 machines (12 total) to 10 jobs and 20 machines (30 total). This is to be expected, as the agent now needs to learn from more inputs, thus making the learning process more complex. By hyperparameter tuning, the agent is able to find similar results to the previous hyperparameter tuned agent, although showing an increase in computational time.

Secondly, the training on multiple problems showed promise in the fact that agents are able to increase the generalization capability on multiple problems. However, using DQN, this method is not able to compete with different scheduling methods. As discussed, this might be due to the inefficiency of scheduling in the environment, and as shown the agents are able to be deployed in the efficient scheduling environment and greatly improve. Hence, the conclusion can be made that the generalization capability does improve and that with an efficient environment, the possibility might be there to even outperform the best suited dispatching rules on these set of problems.

Finally, transfer learning was applied to pre-trained agents. Three different methods showed differences in performance for each agent. Overall, the best performing agent was also the one which was able to generalize the best in pre-training, which seems a logical outcome. Although the performance of the hyperparameter tuned agent can be improved in finding better rewards with this method, the computational time also increases. However, the computational time can be reduced by lowering the number of episodes to train on, as the agent is able to complete the learning in less episodes, reducing the computational time in comparison to the hyperparameter tuned agent.

8.3 Third iteration: the dynamic job shops

To finally assess applicability on dynamic job shop environments, better comparable to real-world environments, agents are created with different reward structures to accommodate for different needs for a different environment. These rewards as proposed in Chapter 6.1.3 as well as in the Appendix N are assessed to determine the best performing reward structure. After the rewards are chosen, the hyperparameter tuning will be done. Equal to the static job shop scheduling problems, when hyperparameters are defined, agents are trained on multiple problems to assess if the complexity increases from static to dynamic, the agents can still increase overall performance on multiple problems by training on more problems.

8.3.1 Reward structure testing and the benchmark

To assess the performance of the reward structures as well the benchmark, nine reward structures were tested with different hyperparameters, as well as normalizing the rewards. This is further explained in Appendix N. In Table **??** an overview is given of the results, where three different iterations were done. First, the agents were trained with the reward structures. Secondly, the reward structures are normalized, to give a reward signal between -1 and 1, by dividing the tardiness and slack by the maximum amount found. Finally, some hyperparameters were adjusted, being the number of neurons and layers to 256 and 3 respectively, as well as the learning rate to 1E-4 and the minimum epsilon to 1E-5. For the standard settings, refer to Chapter 12.

From the results, it was determined that reward structure 1, using the utilization rate only, similar to static job shop scheduling, was the best performing. Hence, this structure is used for dynamic job shop scheduling.

Another conclusion can be made from these results looking at reward structures 6 to 9, as these agents focus on minimizing both slack and tardiness. Although maybe not possible to do both, the agents are able to minimize the slack if the reward signal tells it to. Hence reward signals are a powerful tool to change the objective for the agent, as shown.

Reward structure	T_{avg}	S_{avg}	U_{avg}	C	R
First iteration					
1 - Eq. 8	7.7	5.8	0.59	69	3.5
2 - Eq. 37	14.8	3.2	0.45	89	-269.0
3 - Eq. 38	10.3	6.2	0.57	85	-295.0
4 - Eq. 39	9.7	6.2	0.58	68	-247.5
5 - Eq. 40	9.7	6.2	0.58	68	-247.5
6 - Eq. 41	14.8	3.2	0.45	89	-302.0
7 - Eq. 42	6.5	5.5	0.63	63	-368.0
8 - Eq. 43	13.0	0.0	0.52	72	-293.9
9 - Eq. 44	14.8	3.2	0.45	89	-303.3
Normalized rewards					
1	7.7	5.8	0.59	69	3.5
2	5.7	6.0	0.60	68	1.0
3	13.3	3.5	0.54	91	-1.4
4	11.2	4.8	0.59	71	3.9
5	9.8	5.3	0.54	84	3.5
6	9.8	1.3	0.51	77	-3.0
7	10.5	1.3	0.50	75	-2.7
8	4.0	3.2	0.55	78	-1.4
9	9.5	1.3	0.48	86	0.1
Normalized rewards and changed hyperparameters					
1	4.0	3.8	0.61	61	3.7
2	5.7	6.0	0.60	68	2.1
3	7.7	5.8	0.59	69	1.7
4	5.7	6.0	0.60	68	5.6
5	9.7	5.7	0.61	66	5.9
6	10.5	1.3	0.50	75	-2.6
7	10.5	1.3	0.50	75	-2.6
8	7.3	1.8	0.59	68	0.2
9	7.3	1.8	0.59	68	0.2

Table 36: Results from each iteration with different reward structures

The training graph for the final iteration for the reward structure is given in Figure 30, showing desired behavior already.

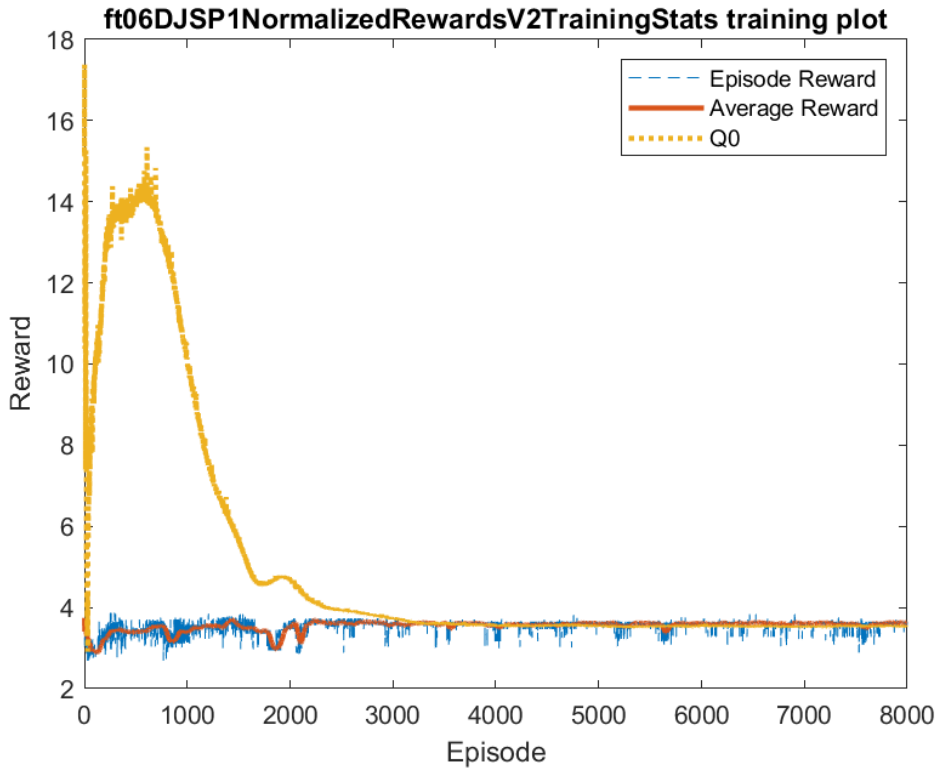


Figure 30: Training graph of best found agent with reward focused on utilization rate

The chosen actions are given in Figure 31 showing that it already combines some of the actions to find a feasible solution. Statistical results are given in Table **??** and the resulting Gantt chart is depicted in Figure 32, showing the ability to combine actions to find a feasible solution.

	R	**C**	$T_{\mathbf{avg}}$	$S_{\mathbf{avg}}$	$U_{\mathbf{avg}}$
UL	3.65	61	4	3.83	0.61
LL	3.65	61	4	3.83	0.61
mean	3.65	61	4	3.83	0.61
Var	8.05E-31	0	0	2.01E-31	0
StD	8.97E-16	0	0	4.49E-16	0

Table 37: Statistical results of benchmark agent

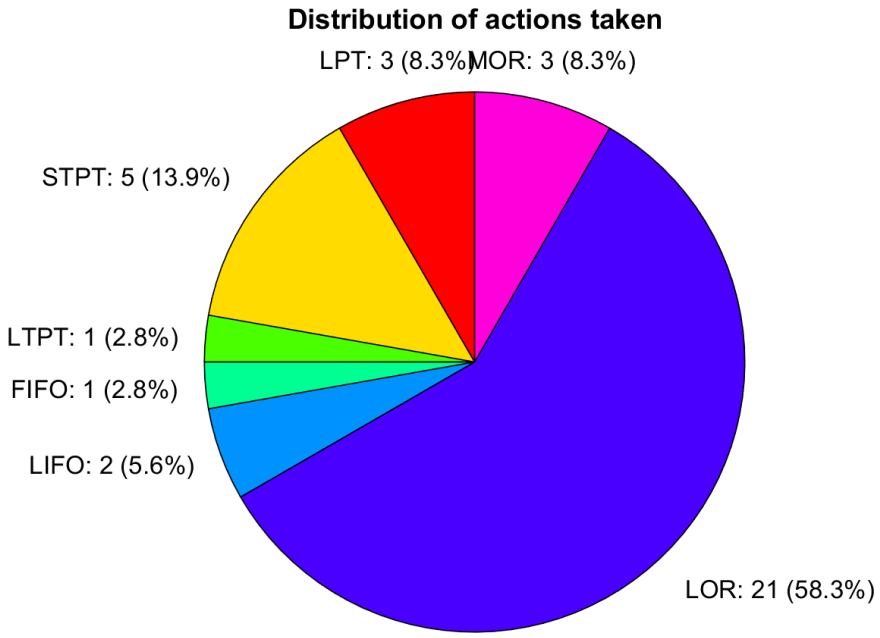


Figure 31: Pie chart of taken actions of the benchmark agent

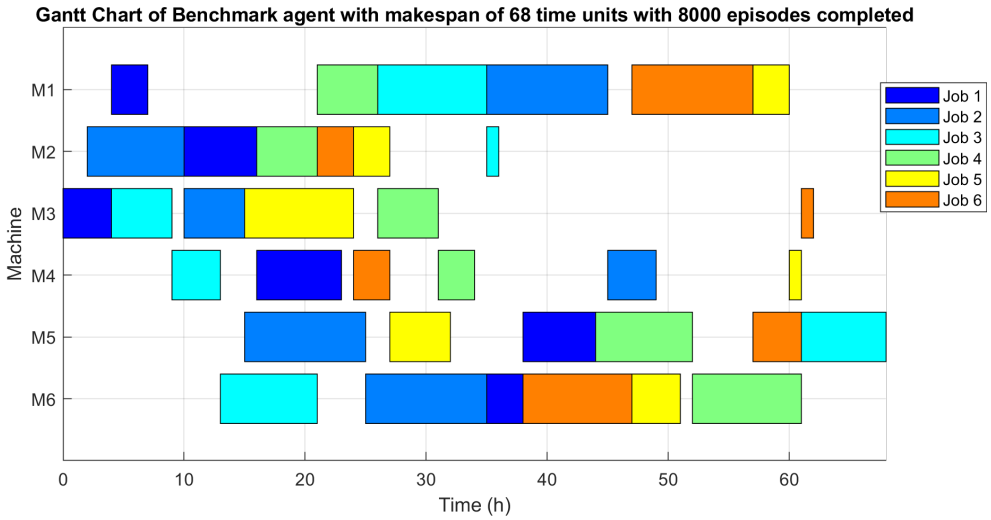


Figure 32: Gantt Chart of benchmark agent in dynamic environment

8.3.2 Hyperparameter testing

For hyperparameter tuning, as some hyperparameters where changed in the previous section, the range of is smaller than before. The gained knowledge of the hyperparameters should ensure faster hyperparameter tuning. In this section, only two iterations of hyperparameter testing where needed, each iteration given in Table 38. Further details of this section are in Appendix O.

It.	α	γ	$\epsilon_{\textbf{init}}$	$\epsilon_{\textbf{min}}$	$\epsilon_{\textbf{decay}}$	**N**	**L**	**E**
1	1e-4 1e-5 1e-6	0.75 0.99	0.75 1	1e-8 1e-7 1e-6	1e-4 1e-5 1e-6	64 128 256	1 to 5	8000
2	1e-4 1e-5	0.99	1	1e-6	1e-3 5e-4 1e-4 5e-5	128 256	3 to 5	8000

Table 38: Hyperparameter settings for each iteration per the second iteration test

For the hyperparameter tuned agent, the settings found are given in Table 39 and the resulting training graph in Figure 33. From the learning behavior, as formulated in Chapter 7.6, it can be assessed that the agent shows desired behavior. Based on the combination on its performance, depicted in Table 40, as well as learning behavior this agent was chosen. From the performance it is found that the reward is increased, decreasing the makespan, slack and tardiness altogether. Hence, the agent is able perform up to a considerable level further improving upon the benchmark agent.

α	γ	$\epsilon_{\textbf{init}}$	$\epsilon_{\textbf{min}}$	$\epsilon_{\textbf{decay}}$	**N**	**L**
1E-05	0.99	1	1E-06	5E-05	256	5

Table 39: Hyperparameter settings for final chosen agent

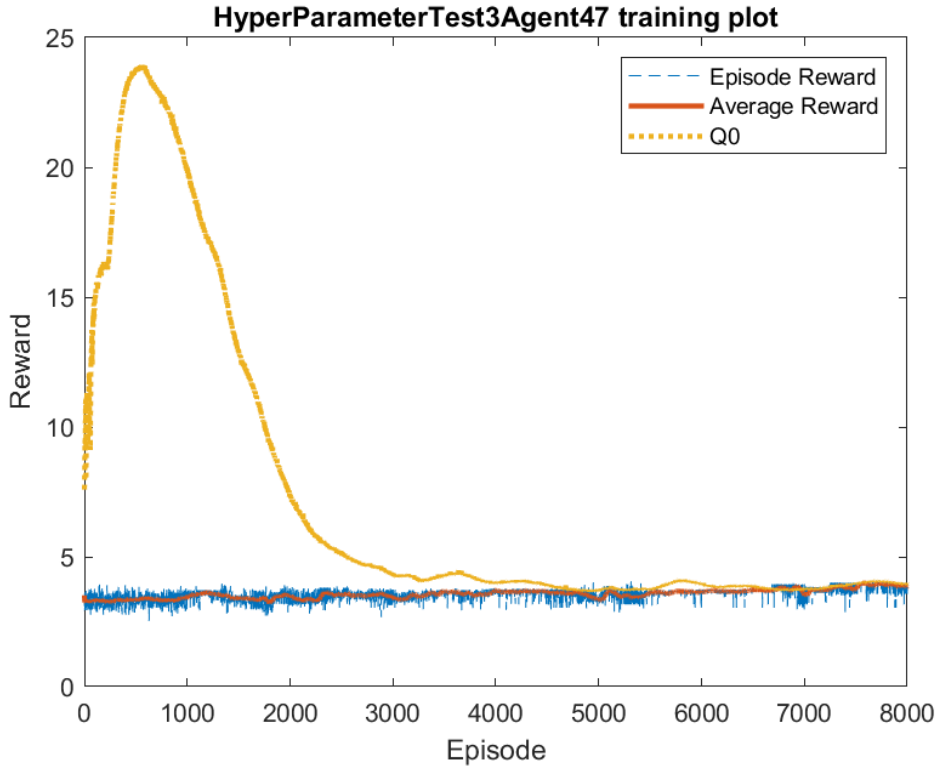


Figure 33: Training graph of chosen agent from hyperparameter tuning

	R	C	T_{avg}	S_{avg}	U_{avg}
UL	3.99	60	6.83	6	0.66
LL	3.99	60	6.83	6	0.66
mean	3.99	60	6.83	6	0.66
Var	1.81E-30	0	8.05E-31	0	5.03E-32
StD	1.35E-15	0	8.97E-16	0	2.24E-16

Table 40: Statistical results of chosen agent

The computational time came down to 8532 seconds, or 2 hours, 22 minutes and 12 seconds. Here, the chosen actions are given in Figure 34, which again indicates learning better balancing of the given actions.

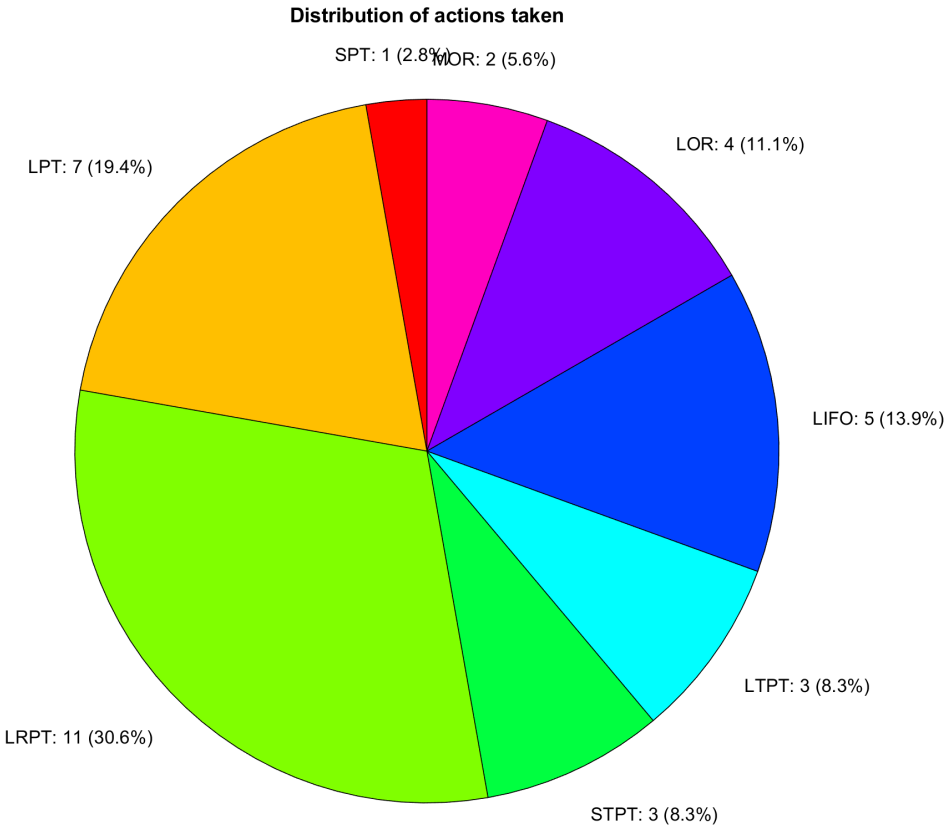


Figure 34: Pie chart of chosen actions by the hyperparameter tuned agent for dynamic environment

Finally, the Gantt chart is given in Figure 35. Here it is found that in comparison to the benchmark agent, a different feasible solution is found. As per the chosen actions, these are also quite different, hence the change in scheduling.

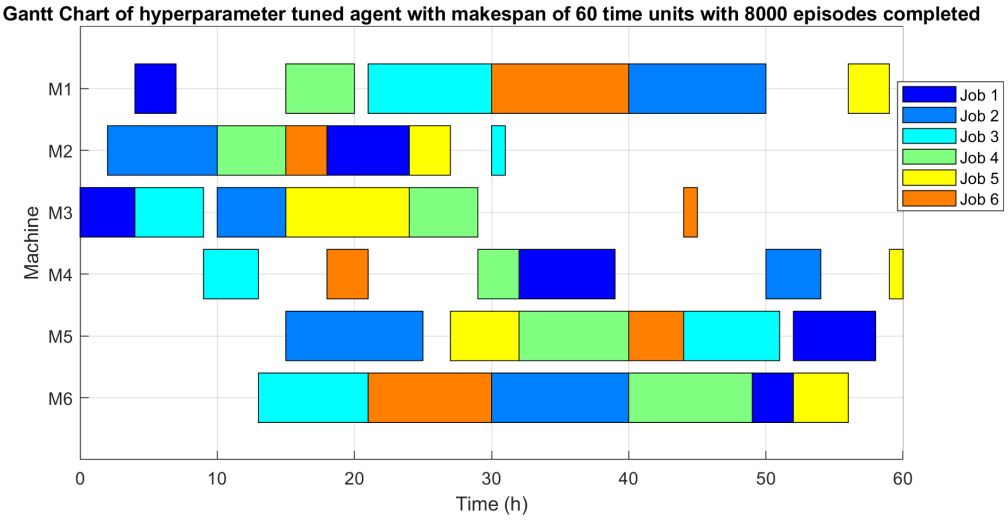


Figure 35: Gantt chart of chosen hyperparameter tuned agent for dynamic environment

8.3.3 Training on multiple problems

The training on multiple problems is done on the same problems of the previous 6 by 6 ones, changed to fit in the dynamic environment. The results are given in Table 41. From these it is found that for the dynamic environment, the generalization is less possible for these problem, as the dispatching rules do not outperform the agents and there is no real better or worse solver. For similar problems not trained on as well as with different time distribution of U[20, 99] the same solutions are found. The exact results are given in Appendix P.

Problems trained on				
	C	U_{avg}	S_{avg}	T_{avg}
SRPT	77.16	0.53	6.2	5.1
LRPT	75.12	0.55	6.8	4.4
SPT	78.1	0.52	6.2	3.8
LPT	75.34	0.54	7.8	7.0
LTPT	75.34	0.54	7.8	7.0
STPT	78.1	0.52	6.2	3.8
FIFO	77.7	0.53	6.9	5.1
LIFO	74.56	0.55	6.6	6.1
LOR	76.96	0.53	6.0	4.3
MOR	78.38	0.53	8.1	6.0
agent 1	77.66	0.53	6.8	5.3
agent 2	76.9	0.53	7.0	5.1
agent 3	76.72	0.53	7.2	5.2
agent 4	77.02	0.53	7.0	5.8
agent 5	76.48	0.55	6.7	5.1

Table 41: Results of dispatching rules and agents of problems trained on

When comparing the agents to dispatching rules on the created dynamic variant of the ft06 problem, better results are found given in Table 42. Here it is found that LRPT is able to solve the problem the best of the dispatching rules. However, agent 4 (trained on 25 problems) is able to perform comparably, giving a lower makespan, with higher slack but also higher tardiness. Hence, the agents are able to solve some of the problems better than the dispatching rules, but generalization of the problems is less possible. This is probably due to the real-time scheduling in the environment, giving less freedom of choice as the first job is always the same one, thus the actions having less influence.

Dynamic ft06				
	C	U_{avg}	S_{avg}	T_{avg}
SRPT	84	0.49	0.0	12.5
LRPT	63	0.63	5.5	6.5
SPT	86	0.48	1.3	9.8
LPT	68	0.58	6.2	9.7
LTPT	68	0.58	6.2	9.7
STPT	86	0.48	1.3	9.8
FIFO	93	0.45	3.2	13.0
LIFO	85	0.57	6.2	10.3
LOR	86	0.48	1.3	9.5
MOR	89	0.45	3.2	14.8
agent 1	76	0.59	6.2	10.0
agent 2	71	0.58	5.3	7.5
agent 3	71	0.58	5.3	7.5
agent 4	62	0.62	5.8	8.3
agent 5	68	0.60	3.2	5.8

Table 42: Results of dispatching rules and agents for dynamic ft06 problem

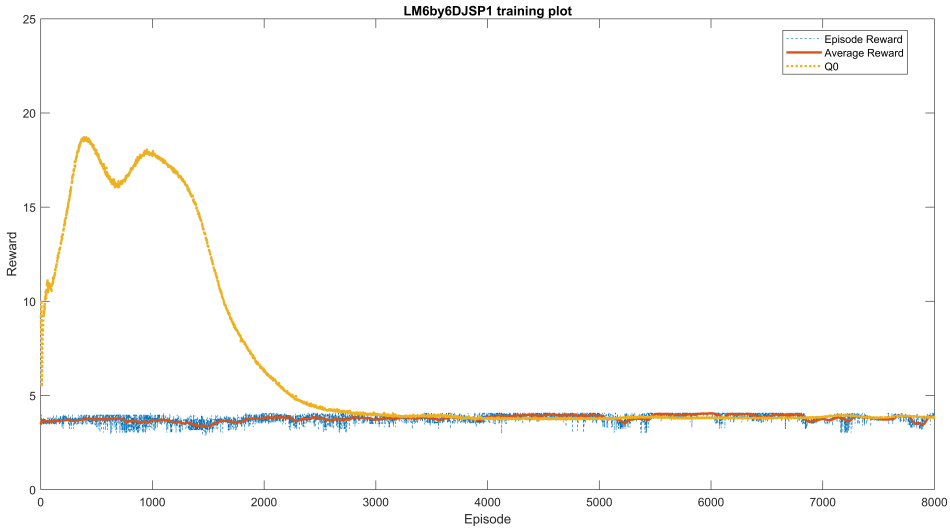
Finally, to compare learning behavior of these agents, the agents 1, 3 and 5 are chosen to show differences in learning behavior. From this it can be concluded by the consistent Q0 estimates that the dynamic environment shows less unpredictable behavior, when comparing to the training graphs found for the static environment. This again confirms that the real-time scheduling for the dynamic environment might reduce complexity instead of increasing it for the setup of the DQN and the environment.

It can be concluded that the difference between the static and dynamic environment in terms of both the efficiency in scheduling as well as simulating in real-time influences the solutions that can be found. As the agent has to schedule in real-time, less influences can be made based on the actions at the start of a job shop, as at time $t = 0$ there is only one job to choose, hence each action giving the same result. Daily or weekly scheduling might be a better method to use for dynamic environments, giving the agent the option to schedule all possible jobs immediately, creating more influence of scheduling actions. Further details of the results found are described in Appendix P.

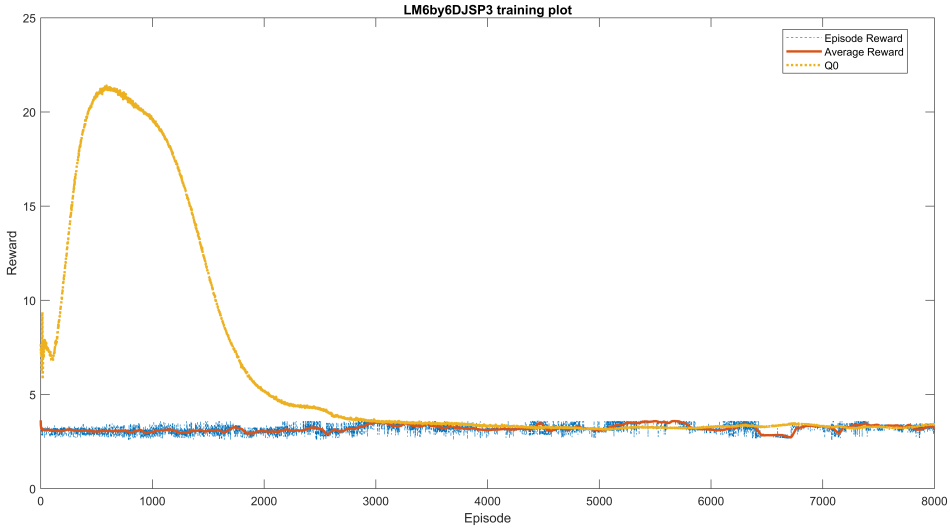
8.3.4 Conclusion on dynamic scheduling

It can be concluded here that different reward structures can lead into different results, as expected. Hence, the possibility to adjust for specific needs of a company is there. The chosen reward structure based on utilization rate showed the most promising result, probably due to being less complex and better defined than using an expected tardiness and slack.

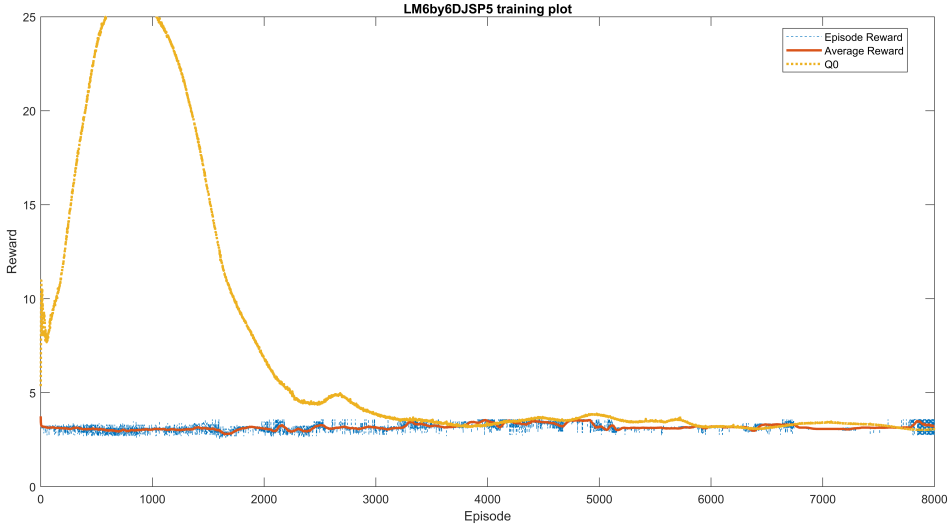
Finally, the generalization capabilities are less than for static job shop scheduling, which is most likely due to the main difference in real-time scheduling versus all jobs at the same time. Hence, instead of using real-time, a daily or weekly scheduling simulation might be better, while adding actions such as the dispatching rule earliest due date (EDD), to accommodate for the dynamic nature. If these adjustments are made, the generalization capabilities are likely to improve and also might be more suitable for companies over real-time scheduling.



(a) Learning behavior of agent 1 (1 problem)



(b) Learning behavior of agent 3 (10 problems)



(c) Learning behavior of agent 5 (50 problems)

Figure 36: Learning behavior of agents 1, 3 and 5 in dynamic environment

8.4 Conclusion

To conclude the research done in this chapter, the static job shop environment shows that the first setup, having the inputs based on the number of machines and jobs, does not work well when wanting to solve problems with different dimensions. The computational time increases in a superlinear way, thus showing the need for more consistency in the architecture of the neural network for using the same hyperparameters.

For the second iteration of the static job shop tests, the neural networks architecture was made consistent, of 10 jobs and 20 machines. After hyperparameter tuning again, the agents were trained on multiple problems. Here it was found that although increasing the difference in training data increases generalization capabilities, the dispatching rules were still able to outperform them. However, if the environment is adjusted to more efficient scheduling, as proposed, the performance of the generalization might also increase, thus improving the change of outperforming the dispatching rules.

Using these pre-trained agents, they were redeployed for transfer learning. Using different setups it was found that transfer learning can indeed improve performance, as expected. Also, the episodes necessary for training could be reduced, to reduce training time by about 2000 seconds for this specific case, or 30%.

Finally, a dynamic environment was creating with real-time scheduling simulation. From this it was concluded that the possibility is there to deploy agents for dynamic environments, outperforming the dispatching rules when comparing the hyperparameter tuned agent to the dispatching rules results. However, training on multiple problems showed that the generalization capabilities are less applicable, as the difference in both dispatching rules as well as the agents are small. This is most likely due to the real-time scheduling, constraining the agent into scheduling the first job on the first action at all times, no matter what action is chosen. Hence, less freedom for the agent is created.

9 Discussion

Throughout this research, various experiments were conducted to evaluate the performance of DRL-based models in static and dynamic job shop scheduling scenarios. Several insights emerged regarding the models' adaptability and efficiency, but also some important limitations were identified.

For static job shop scheduling, different methods were found to be applicable. Although the scheduling environment was not optimally defined for efficiency, the performance of the DRL agents was comparable to traditional methods such as dispatching rules. It was noted that the performance of agents could potentially improve if the environment was better defined for scheduling efficiency. In the context of the efficient environment used in the study, the DRL model outperformed 7 out of 10 dispatching rules. This suggests that with proper training in a well-defined environment, the performance of DRL agents could be significantly enhanced.

Generalization capabilities of the DRL models showed improvement when agents were trained on multiple problems. However, despite this improvement, the performance did not exceed that of dispatching rules. The specific environment used for training might have influenced these results. For instance, when deployed in an efficient environment, DRL models demonstrated better performance, indicating that tailored training environments can lead to better outcomes.

Transfer learning was found to be beneficial, reducing computational time by 30% for specific cases. While pre-training is necessary, it can be conducted beforehand, allowing for reduced training time and increased performance on the specific tasks at hand.

In dynamic scheduling environments, hyperparameter tuning was effective in improving performance. Different reward structures were tested, with machine utilization rate-based rewards yielding the best results for static job shop scheduling. However, generalization across multiple problems remained a challenge. The agents' performance was comparable to dispatching rules, with little variation likely due to the real-time nature of dynamic scheduling. In this setting, where only a few jobs are available at each time increment, the agents had less flexibility in job scheduling, limiting the impact of their decisions. A more comprehensive approach, akin to static job shop scheduling where all jobs are considered simultaneously, might offer better results.

The computational demands of DRL models were found to be significant. DRL methods required longer computational times compared to traditional methods like dispatching rules, without a substantial improvement in performance. Scalability remains an issue, as computational time increases superlinearly with the size of the job shop. To address this, using multiple GPUs or other advanced computational resources could help, though this introduces a trade-off between resource costs and scheduling improvement. The research predominantly focused on 6x6 job shop problems due to compatibility with hyperparameters, which limits scalability to larger problems with more machines and jobs.

In summary, while DRL models show potential, particularly when trained in well-defined environments, their practical application is constrained by high computational demands, sensitivity to hyperparameters, and scalability issues. These limitations must be addressed to fully realize the benefits of DRL in job shop scheduling scenarios.

10 Conclusion

The primary research question addressed in this thesis was: "How can machine learning be applied in real-world dynamic job shops?" To answer this, several sub-questions were explored, given in Chapter 1.4. Firstly, we examined which machine learning methods are suitable for dynamic job shop scheduling. Chapter 4.2 provided an overview of various methods including PPO, MARL, GNN, CNN, RNN, LSTM, and DQN. It was determined that DQN was the most appropriate choice for this research due to its balance between generalization capabilities and computational resource requirements, as well as lower deployment complexity compared to methods like MARL and PPO.

The second question investigated how machine learning methods compare to traditional methods. It was found that while some machine learning methods can outperform traditional ones in terms of performance metrics, they often require significantly more time. For instance, DQN took between one and two hours to achieve performance comparable to dispatching rules, which only require seconds.

The third question focused on the scalability of machine learning methods with increasing job shop sizes. The research assessed neural network performance with varying job shop sizes and found that performance was influenced by input size and hyperparameters. By fixing input sizes, the performance was more transferable, and agents trained on specific problem sizes were able to solve other job shops within those limits.

The fourth question examined the generalization capabilities of machine learning models to new, unseen job shop scenarios. Agents trained on multiple problems showed improved generalization, though still not as effective as dispatching rules in dynamic environments. Generalization was better with increased problem diversity but remained limited.

The final question explored the application of these methods to real-world problems. For companies with minimal AI experience, traditional methods may be more practical due to their simplicity and effectiveness. For companies with moderate AI expertise, DQN could be deployed using transfer learning, particularly for smaller job shops with fewer than 20 inputs. For companies with extensive AI resources, exploring methods like MARL or PPO could yield better performance and efficiency.

In summary, while DQN shows promise, its practical application in real-world job shops is currently limited by its computational demands and generalization issues. As AI technology evolves, its application in job shop scheduling may become more viable.

10.1 Recommendations

Based on the research findings, it is advisable not to deploy machine learning solutions in real-world environments at this stage, given that traditional methods offer comparable performance with significantly lower time and resource requirements. For future research, it is crucial to gather real-world data to better understand the practical application of machine learning in job shops. Investigating how job shops actually schedule jobs, manage operations, and handle due dates could provide valuable insights. A larger and more representative dataset would allow for longer training periods, potentially improving agent performance. Increasing the experience buffer could also enhance performance but will require additional computational resources, which has not been thoroughly explored in this study.

The research also revealed that hyperparameter tuning and training are time-consuming. To address this, increasing computational resources, such as using multiple laptops, PCs, or GPUs, would accelerate hyperparameter tuning and training processes. This would provide a more comprehensive understanding of machine learning applications in job shops. Lastly, exploring other advanced methods, such as MARL and PPO, is recommended. These methods have demonstrated better performance in static job shop scheduling scenarios according to the literature. If these methods can improve generalization and problem-specific performance, they could make machine learning more feasible for real-world applications.

Concluding, how can machine learning be deployed in real-world environments? This can be done

in multiple ways as mentioned in this section before. However, at this point, it is recommended to continue using traditional methods such as dispatching rules due to their ability to perform in a comparable manner while solving problems in a matter of seconds. As technology develops and further knowledge is gained on deploying AI, the performance and practicality of machine learning methods for job shop scheduling are likely to improve.

References

[1] Chris J. Maddison David Silver, Aja Huang et al. Mastering the game of go with deep neural networks and tree search. *Nature 2016 529:7587*, 529:484–489, 1 2016.

[2] Christopher M. Bishop. Pattern recognition and machine learning. *Pattern Recognition and Machine Learning*, 12 2006.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[4] Adam E. Duerr Silas Bergen, Manuela M. Huso et al. A review of supervised learning methods for classifying animal behavioural states from environmental features. *Methods in Ecology and Evolution*, 14:189–202, 1 2023.

[5] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9910:69 – 84, 1 2016.

[6] Babak Ehteshami Bejnordi Geert Litjens, Thijs Kooi et al. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60 – 88, 12 2017.

[7] Wojciech M. Czarnecki Oriol Vinyals, Igor Babuschkin et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575:350 – 354, 11 2019.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[9] Jiangliang Jin and Yunjian Xu. Optimal differentiated threshold characterization for multi-task stochastic deadline scheduling with queuing. *Automatica*, 163:111545, 5 2024.

[10] Rugang Tang, Xin Ning, Zheng Wang, Jiaqi Fan, and Shichao Ma. Dynamic scheduling for multi-level air defense with contingency situations based on human-intelligence collaboration. *Engineering Applications of Artificial Intelligence*, 132:107893, 6 2024.

[11] Yu Hung Chang, Chien Hung Liu, and Shingchern D. You. Scheduling for the flexible job-shop problem with a dynamic number of machines using deep reinforcement learning. *Information (Switzerland)*, 15:82, 2 2024.

[12] Xin Hao, Phee Lep Yeoh, Changyang She, Branka Vucetic, and Yonghui Li. Secure deep reinforcement learning for dynamic resource allocation in wireless mec networks. *IEEE Transactions on Communications*, 72:1414 – 1427, 3 2024.

[13] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *https://doi.org/10.1287/mnsc.34.3.391*, 34:391–401, 3 1988.

[14] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109:137–141, 8 1998.

[15] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. *Prentice-Hall, Englewood Cliffs, 225-251. Scientific Research Publishing*, 1963.

[16] Lawrence S. Resouce constrained project scheduling : an experimental investigation of heuristic scheduling techniques (supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.

[17] David Applegate and William Cook. A computational study of the job-shop scheduling problem. *https://doi.org/10.1287/ijoc.3.2.149*, 3:149–156, 5 1991.

[18] Robert H. Storer, S. David Wu, and Renzo Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *https://doi.org/10.1287/mnsc.38.10.1495*, 38:1495–1509, 10 1992.

[19] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1 1993.

[20] Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop problems. *Parallel Problem Solving from Nature*, 2:283–292, 01 1992.

[21] Michael L. Pinedo. Scheduling: Theory, algorithms, and systems, sixth edition. *Scheduling: Theory, Algorithms, and Systems, Sixth Edition*, pages 1–698, 1 2022.

[22] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *https://doi.org/10.1287/opre.14.4.699*, 14:699–719, 8 1966.

[23] Christian Artigues and Dominique Feillet. A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals of Operations Research*, 159:135 – 159, 3 2008.

[24] Mitsuru Kuroda and Zeng Wang. Fuzzy job shop scheduling. *International Journal of Production Economics*, 44:45 – 51, 6 1996.

[25] Peter Brucker, Bernd Jurisch, and Bernd Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107 – 127, 3 1994.

[26] Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183:643 – 657, 12 2007.

[27] George Nemhauser and Laurence Wolsey. Integer and combinatorial optimization. *Integer and Combinatorial Optimization*, pages 1–766, 1 2014.

[28] Dinh Nguyen Pham and Andreas Klinkert. Surgical case scheduling as a generalized job shop scheduling problem. *European Journal of Operational Research*, 185:1011 – 1025, 3 2008.

[29] Cemal Özgüven, Lale Özbakir, and Yasemin Yavuz. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34:1539 – 1548, 6 2010.

[30] Leilei Meng, Chaoyong Zhang, Yaping Ren, Biao Zhang, and Chang Lv. Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. *Computers and Industrial Engineering*, 142:106347, 4 2020.

[31] Leilei Meng, Chaoyong Zhang, Xinyu Shao, and Yaping Ren. Milp models for energy-aware flexible job shop scheduling problem. *Journal of Cleaner Production*, 210:710 – 723, 2 2019.

[32] Luca Maria Gambardella, Andrea E. Rizzoli, and Marco Zaffalon. Simulation and planning of an intermodal container terminal. *Simulation*, 71:107 – 116, 8 1998.

[33] Kim. Marriott and Peter J. Stuckey. Programming with constraints : an introduction. page 467, 1998.

[34] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1520:417 – 431, 1 1998.

[35] J. Christopher Beck, T. K. Feng, and Jean Paul Watson. Combining constraint programming and local search for job-shop scheduling. *https://doi.org/10.1287/ijoc.1100.0388*, 23:1–14, 5 2010.

[36] Yuan Sun, Su Nguyen, Dhananjay Thiruvady, Xiaodong Li, Andreas T. Ernst, and Uwe Aickelin. Enhancing constraint programming via supervised learning for job shop scheduling. *Knowledge-Based Systems*, 293:111698, 6 2024.

[37] John H. Blackstone, Don T. Phillips, and Gary L. Hogg. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, 20:27–45, 1982.

[38] Su Nguyen, Mengjie Zhang, Mark Johnston, and Kay Chen Tan. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 17:621–639, 2013.

[39] M. Thenarasu, K. Rameshkumar, S. P. Anbuudayasankar, G. Arjunbarath, and P. Ashok. Development and selection of hybrid dispatching rule for dynamic job shop scheduling using multi-criteria decision making analysis (mcdma). *Volume 14, Issue 2, Pages 487 - 504*, 14:487–504, 2020.

[40] Ulrich Dorndorf and Erwin Pesch. Evolution based learning in a job shop scheduling environment. *Computers and Operations Research*, 22:25 – 40, 1 1995.

[41] Shu Luo. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Volume 91*, 91, 1 2020.

[42] David E. Goldberg and John H Holland. Genetic algorithms in search, optimization, and machine learning david e. goldberg the university of alabama t. *Machine Learning*, 3:95–99, 1979.

[43] F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers and Operations Research*, 35:3202 – 3212, 10 2008.

[44] Guohui Zhang, Liang Gao, and Yang Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38:3563 – 3573, 4 2011.

[45] Christian Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spektrum*, 17:87 – 92, 6 1995.

[46] Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms - i. representation. *Computers and Industrial Engineering*, 30:983 – 997, 9 1996.

[47] José Fernando Gonçalves, Jorge José De Magalhães Mendes, and Maurício G.C. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77 – 95, 11 2005.

[48] Peter J. M. van Laarhoven and Emile H. L. Aarts. Simulated annealing: Theory and applications. *Simulated Annealing: Theory and Applications*, 1987.

[49] Hirofumi Matsuo, Chang Juck SUH, and Robert S. Sullivan. A controlled search simulated annealing method for the single machine weighted tardiness problem. *Annals of Operations Research*, 21:85–108, 12 1989.

[50] Peter J.M. Van Laarhoven, Emile H.L. Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113 – 125, 1990.

[51] M. Kolonko. Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, 113:123 – 136, 2 1997.

[52] R. K. Suresh and K. M. Mohanasundaram. Pareto archived simulated annealing for job shop scheduling with multiple objectives. *International Journal of Advanced Manufacturing Technology*, 29:184 – 196, 5 2005.

[53] Fred Glover and Manuel Laguna. Tabu search. *Tabu Search*, 1997.

[54] Mauro Dell'Amico and Marco Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231 – 252, 9 1993.

[55] Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, 15:205 – 215, 12 1994.

[56] Eugeniusz Nowicki and Czesław Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145 – 159, 4 2005.

[57] Marco Dorigo and Thomas Stützle. Ant colony optimization. 2004.

[58] J. Heinonen and F. Pettersson. Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. *Applied Mathematics and Computation*, 187:989 – 998, 4 2007.

[59] Yang Jiang and Qiulei Ding. An improved ant colony optimization for job-shop scheduling problem. *ICIC Express Letters, Part B: Applications*, 8:1465 – 1471, 11 2017.

[60] Nadia Lachtar and Imen Driss. Application of ant colony optimization for job shop scheduling in the pharmaceutical industry. *Journal Europeen des Systemes Automatises*, 56:713 – 723, 1 2023.

[61] Maurice Clerc. Particle swarm optimization. *Particle Swarm Optimization*, 1 2010.

[62] D. Y. Sha and Cheng Yu Hsu. A hybrid particle swarm optimization for job shop scheduling problem. *Computers and Industrial Engineering*, 51:791 – 808, 12 2006.

[63] D. Y. Sha and Hsing Hung Lin. A multi-objective pso for job-shop scheduling problems. *Expert Systems with Applications*, 37:1065 – 1070, 3 2010.

[64] Shi Jinn Horng Tsung Lieh Lin and Tzong Wann Kao et al. An efficient job-shop scheduling algorithm based on particle swarm optimization. *Expert Systems with Applications*, 37:2629 – 2636, 3 2010.

[65] Thomas H. Cormen, Charles Eric Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms, 4 edition. *The MIT Press Cambridge, Massachusetts London, England*, page 1291, 2022.

[66] Guanlong Deng, Qingtang Su, Zhiwang Zhang, Huixia Liu, Shuning Zhang, and Tianhua Jiang. A population-based iterated greedy algorithm for no-wait job shop scheduling with total flow time criterion. *Engineering Applications of Artificial Intelligence*, 88:103369, 2 2020.

[67] Mingming Xu, Shuning Zhang, and Guanlong Deng. No-wait job shop scheduling using a population-based iterated greedy algorithm. *Algorithms*, 14:145, 1 2021.

[68] Yi Zhang, Haihua Zhu, Dunbing Tang, Tong Zhou, and Yong Gui. Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, 78:102412, 12 2022.

[69] Renke Liu, Rajesh Piplani, and Carlos Toro. A deep multi-agent reinforcement learning approach to solve dynamic job shop scheduling problem. *Computers and Operations Research*, 159:106294, 11 2023.

[70] Ali Fırat İnal, Çağrı Sel, Adnan Aktepe, Ahmet Kürşad Türker, and Süleyman Ersöz. A multi-agent reinforcement learning approach to the dynamic job shop scheduling problem. *Sustainability (Switzerland)*, 15:8262, 5 2023.

[71] Abebaw Degu Workneh and Maha Gmira. Deep q network method for dynamic job shop scheduling problem. *Lecture Notes in Networks and Systems*, 771:137 – 155, 1 2023.

[72] Yejian Zhao, Yanhong Wang, Yuanyuan Tan, Jun Zhang, and Hongxia Yu. Dynamic jobshop scheduling algorithm based on deep q network. *IEEE Access*, 9:122995 – 123011, 1 2021.

[73] Liyuan Song, Yuanyuan Li, and Jiacheng Xu. Dynamic job-shop scheduling based on transformer and deep reinforcement learning. *Processes*, 11:3434, 12 2023.

[74] Libing Wang, Xin Hu, Yin Wang, Sujie Xu, Shijun Ma, Kexin Yang, Zhijun Liu, and Weidong Wang. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Computer Networks*, 190:107969, 5 2021.

[75] Xinquan Wu, Xuefeng Yan, Donghai Guan, and Mingqiang Wei. A deep reinforcement learning model for dynamic job-shop scheduling problem with uncertain processing time. *Engineering Applications of Artificial Intelligence*, 131:107790, 5 2024.

[76] Ziqing Wang and Wenzhu Liao. Smart scheduling of dynamic job shop based on discrete event simulation and deep reinforcement learning. *Journal of Intelligent Manufacturing*, 2023.

[77] Yiming Gu, Ming Chen, and Liang Wang. A self-learning discrete salp swarm algorithm based on deep reinforcement learning for dynamic job shop scheduling problem. *Applied Intelligence*, 53:18925 – 18958, 8 2023.

[78] Xinquan Wu and Xuefeng Yan. A spatial pyramid pooling-based deep reinforcement learning model for dynamic job-shop scheduling problem. *Computers Operations Research*, 160:106401, 12 2023.

[79] Zhong Yang, Li Bi, and Xiaogang Jiao. Combining reinforcement learning algorithms with graph neural networks to solve dynamic job shop scheduling problems. *Processes*, 11:1571, 5 2023.

[80] Chupeng Su, Cong Zhang, Dan Xia, Baoan Han, Chuang Wang, Gang Chen, and Longhan Xie. Evolution strategies-based optimized graph reinforcement learning for solving dynamic job shop scheduling problem. *Applied Soft Computing*, 145:110596, 9 2023.

[81] Zhenyu Liu, Haoyang Mao, Guodong Sa, Hui Liu, and Jianrong Tan. Dynamic job-shop scheduling using graph reinforcement learning with auxiliary strategy. *Journal of Manufacturing Systems*, 73:1 – 18, 4 2024.

[82] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[83] Ryan Lowe and WU et al. Multi-agent actor-critic for mixed cooperative-competitive environments. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[84] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[85] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[86] Sepp Hochreiter and JÃ¼rgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[87] Create custom reinforcement learning environment template - matlab rlcreateenvtemplate - mathworks benelux. https://nl.mathworks.com/help/reinforcement-learning/ref/rlcreateenvtemplate.html. Accessed: 30-06-2024.

[88] Options for dqn agent - matlab - mathworks benelux. https://nl.mathworks.com/help/reinforcement-learning/ref/rldqnagentoptions.html. Accessed: 02-05-2024.

A Usage of AI tools

As per the document "Use of AI in Education at the University of Twente" by the University Twente, the following statement about the usage of AI tools is made: During the preparation of this work the author used ChatGPT and DeepL in order to improve readability and language of the work. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.

B Further explained concepts of deep- and reinforcement learning

In this Appendix section, some concepts of deep learning and reinforcement learning are further elaborated in. This is done to ensure that more information on certain concepts is available.

B.1 Activation functions

There are a couple of common activation functions. First, Sigmoid is a function that squashes the input to a value between 0 and 1, making it suitable for binary classification tasks, where outputs need to be between [0, 1]. The function is shown in Eq. 9, where x represents the weighted sum given in Eq. **??**. It introduces non-linearity and smooth gradients but suffers from the vanishing gradient problem for extreme values of x. This vanishing gradient problem could hinder the learning process of the neural network.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

Secondly, Hyperbolic Tangent (tanh) transforms the input x to a value between -1 and 1, centered at 0, depicted in Eq. 10. It has similarity to the sigmoid function, while providing stronger gradients making it more effective in training deep neural networks.

$$\text{Tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{10}$$

Next, Rectified Linear Unit, or ReLU, sets all negative values to zero, while leaving positive values unchanged, as shown in Eq. 11. ReLU is computationally efficient and helps counter the vanishing gradient problem, which leads to faster convergence during training.

$$\text{ReLU}(x) = \max(0, x) \tag{11}$$

To counter for a common issue with ReLU, called "dying ReLU", various other ReLU functions have been developed, such as Leaky ReLU. Leaky ReLU allows a small gradient for negative values of the input, as shown in Eq. 12, where the parameter α controls the slope of the negative part. This addresses the "dying ReLU" problem, which occurs when a neuron always outputs a negative value, making it effectively a "dead" neuron.

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{12}$$

Finally, softmax is often used. The softmax function is commonly used in the output layer of neural networks for multi-class classification tasks. It converts raw scores into probabilities, as shown in Eq. 13, which ensures that the sum of the output probabilities adds up to 1, which makes it suitable for multi-class classification.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{13}$$

B.2 Common loss functions

Common loss functions include Mean Squared Error (MSE) used for regression tasks and Cross-Entropy Loss for classification tasks. MSE measures the average squared between predicted and actual values, given in Eq. 14 where n is the number of samples, y_i is the actual target value and $\hat{y}_i$ is the predicted value.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{14}$$

The Cross-Entropy Loss measures the difference between predicted probability distribution and the actual distribution, the Binary Cross-Entropy Loss equation is given in Eq. 15, where n is the number

of samples, y_i is the actual binary label (0 or 1) and $\hat{y}_i$ is the predicted probability for the positive class. In some cases, custom loss functions are used, tailored to specific objectives.

$$\text{Binary Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{15}$$

B.3 Optimization algorithms

In this subsection optimization algorithms are elaborated on.

SDG updates the model parameters in the opposite direction of the gradient of the loss function, in relation to the parameters. SDG is computationally efficient, although it may suffer from slow convergence and oscillations in the loss landscape. The mathematical equation is given in Eq. 16, where θ_t is the parameter vector at time step t, η is the learning rate and $\nabla J(\theta_t)$ is the gradient of the loss function with respect to the parameters.

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t) \tag{16}$$

Adam combines the benefits of adaptive learning rates and momentum to accelerate convergence and handle sparse gradients effectively. At the start of the training process, Adam initializes the parameters (θ), the time step ($t = 0$) and the first ($m = 0$) and second ($v = 0$) moment variables, setting the stage for the subsequent updates and computations during the optimization process. For each time step t in training, the first moment estimation (m) and second moment estimation (v) are computed. The first moment estimation is calculated by Eq. 17 where β_1 is the exponential decay rate for the first moment estimate (m_{t+1}) and $\nabla J(\theta_t)$ is the gradient of the loss function in relation with the parameters at time step t. The second moment estimate (v_{t+1}) is given in Eq. 18, where β_2 is the exponential decay rate for the second moment estimate and $(\nabla J(\theta_t))^2$ represents element-wise squaring of the gradient. Thus, the first moment estimate represent the exponentially decaying average of past gradients, including the momentum into the optimization process. The second moment estimate computes the exponentially decaying average of the squared gradients, providing information about the variance of the gradients.

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla J(\theta_t) \tag{17}$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2)(\nabla J(\theta_t))^2 \tag{18}$$

After computing these values, Adam applies a bias correction to both the first and second moment estimates, to adjust for the initialization bias at the start of training. These help in making the moment estimates more accurate and reliable as the training progresses. $\hat{m}_{t+1}$ represents the first moment with bias correction and $\hat{v}_{t+1}$ the second moment with bias correction. Eq. 19 shows the first moment correction and Eq. 20 the second moment correction.

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^{t+1}} \tag{19}$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^{t+1}} \tag{20}$$

Finally, the parameters are updated, shown in Eq. 21. Here, η is the learning rate and ϵ represents a small constant to prevent division by zero. The bias-corrected first and second moment estimates are used by Adam, to update the model parameters (θ) at each time step. The update rule adjusts the parameters based on the adaptive learning rate, scaled by the ratio of the first moment estimate to the square root of the second estimate. The learning rate controls the size of the parameter updates, ensuring that the optimization process converges effectively.

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon} \tag{21}$$

RMSprop (Root Mean Square Propagation) is an optimization algorithm widely used in the training of deep neural networks. RMSprop is designed to adapt the learning rate for each parameter individually, based on the magnitude of the recent gradients, improving efficiency and stability of

the training process. RMSprop computes the exponentially weighted moving average of the squared gradients at each time step, given in Eq. 22, where r is the accumulated squared gradient, ρ is the decay rate, $\nabla\theta$ represents the gradient of the parameters (the same as $\nabla J(\theta_t$ of Eq. 17) and $\odot$ denotes element-wise multiplication.

$$r = \rho r + (1 - \rho)\nabla\theta \odot \nabla\theta \tag{22}$$

Next, the velocity update is calculated. The velocity update combines the gradient of parameters with the normalized accumulated squared gradient to adjust the step size of the parameter updates during optimization. This is given in Eq. 23, where v is the velocity update, α is the global learning rate, and ϵ is a small constant to prevent division by zero. The term $\sqrt{r} + \epsilon$ normalizes the gradient by the root mean square of the accumulated squared gradients.

$$v = \frac{\alpha}{\sqrt{r} + \epsilon}\nabla\theta \tag{23}$$

Finally, the algorithm updates the parameters with the velocity update, shown by Eq. 24.

$$\theta = \theta - v \tag{24}$$

B.4 Regularization techniques

In this appendix, we provide a detailed explanation of several regularization techniques used in neural networks. These techniques are designed to improve the generalization ability of the model by adding constraints or modifications during training.

L1 Regularization

L1 Regularization, also known as Lasso (Least Absolute Shrinkage and Selection Operator) regularization, encourages sparsity in the weight matrix. It adds a penalty proportional to the absolute value of the weights to the loss function. The regularization term is given by:

$$\text{L1 Regularization} = \lambda \sum_i |w_i| \tag{25}$$

where λ is the regularization parameter that controls the strength of the penalty, and w_i represents each weight in the model. L1 regularization tends to drive some weights to exactly zero, which can simplify the model and help with feature selection.

L2 Regularization

L2 Regularization, also known as Ridge regularization, adds a penalty proportional to the square of the weights. This technique discourages large weights by including a term in the loss function that is proportional to the sum of the squared weights:

$$\text{L2 Regularization} = \lambda \sum_i w_i^2 \tag{26}$$

where λ is the regularization parameter, and w_i represents each weight in the network. L2 regularization helps in smoothing the learned function and prevents overfitting by shrinking the weights towards zero, though not exactly zero as in L1 regularization.

Dropout

Dropout is a regularization method that helps prevent overfitting by randomly dropping a fraction p of the neurons during each training iteration. This means that during each forward pass, a proportion of neurons are ignored (i.e., their activations are set to zero). The dropout rate p is typically set between 0.2 and 0.5. The dropout mechanism is applied as follows:

$$\text{Dropout Rate} = p \tag{27}$$

During training, each neuron is kept with a probability of $1 - p$ and dropped with probability p. During testing, dropout is not applied, and the weights are scaled by the dropout rate to maintain the expected activation values.

Batch Normalization

Batch Normalization normalizes the inputs to each layer to have zero mean and unit variance. This technique accelerates training and improves model stability. The normalized output of a layer is given by:

$$\text{BN}(x) = \gamma \frac{x - \mu}{\sigma} + \beta \tag{28}$$

where x is the input to the layer, μ is the mean of the batch, σ is the standard deviation of the batch, and γ and β are learnable parameters that allow the network to scale and shift the normalized output. This normalization helps reduce internal covariate shift and makes the network less sensitive to the initial weights.

By incorporating these regularization techniques, neural networks become more robust and better at generalizing to new, unseen data, thus improving their performance and reducing overfitting.

B.5 Algorithms in reinforcement learning

Different algorithms are deployed for RL to learn optimal policies or value functions. Dynamic programming (DP) uses a pre-defined model of the environment to identify the optimal policy through iterative methods such as value iteration or policy iteration. DP works well with small discrete state and action spaces but is computationally expensive and requires a model of the environment.

Monte Carlo methods use value functions to learn from sample episodes without needing a model of the environment. The estimated value functions are based on average returns observed from multiple episodes. Monte Carlo methods are model-free and suitable for episodic tasks. However, they may require a large number of episodes to converge.

Temporal-Difference (TD) learning is a model-free method that updates value estimates based on observed transitions. TD combines ideas from DP and Monte Carlo methods by using bootstrapping—updating the value estimate for a state based on an estimate of the value of future state(s). This makes TD suitable for real-time learning. An example of TD is Q-learning, which learns the action-value (Q-value) directly from experience. The Q-values are iteratively updated based on observed transitions. Q-learning learns a value function using a policy different from the one being improved. The Q-learning update equation is a form of the Bellman equation, as given in Eq. 29. Here, $Q(s, a)$ represents the action-value function, R_{t+1} is the immediate reward received after taking action a in state s, γ is the discount factor, α is the learning rate, s' represents the next state after taking action a and $\max_{a'} Q(s', a')$ represents the maximum Q-value among all possible actions in state s'.

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{29}$$

C Comparison of heuristic methods

In Table 43 a comparison is made between different heuristics methods, in terms of found makespan (C) in time units and computational time (t) in seconds per instance. The size of an instance is in jobs x machines.

Paper		[47]		[45]		[48]		[60]	
Method		GA		GA		SA		ACO	
Instance	**size**	C	t	C	t	C	t	C	t
ft06	6x6	55	13			55	52	55	3,25
ft10	10x10	930	292	936	135	1006	5945	930	14,5
ft20	20x5	1165	204	1181	147	1319	6841	1165	25
La01	10x5	666	37			666	6	666	1,45
La02	10x5	655	51			655	24		
La03	10x5	597	39			606	129		
La04	10x5	590	42			590	121		
La05	10x5	593	32			593	5		
La06	15x5	926	99			926	16	926	1,1
La07	15x5	890	86			890	66		
La08	15x5	863	99			863	16		
La09	15x5	951	94			951	13		
La10	15x5	958	91			958	14		
La11	20x5	1222	197			1222	32		
La12	20x5	1039	201			1039	34	1043	2,24
La13	20x5	1150	189			1150	32		
La14	20x5	1292	187			1292	27		
La15	20x5	1207	187			1207	34		
La16	10x10	945	232			956	686	947	2,4
La17	10x10	784	216			784	112		
La18	10x10	848	219			861	112		
La19	10x10	842	235			848	830		
La20	10x10	902	235			902	667		
La21	15x10	1046	602			1063	1991		
La22	15x10	927	629			938	2163	927	5,98
La23	15x10	1032	594			1032	275		
La24	15x10	935	578			952	2098		
La25	15x10	977	609			992	2133		
La26	20x10	1218	1388	1232	492	1218	4342	1218	1045
La27	20x10	1235	1251	1269	502	1269	4535		
La28	20x10	1216	1267	1256	495	1224	4354		
La29	20x10	1157	1350	1233	501	1203	581		
La30	20x10	1355	1260	1355	499	1355	3956		
La31	30x10	1784	3745			1784	1517		
La32	30x10	1850	3741			1850	1752		
La33	30x10	1719	3637			1719	1880		
La34	30x10	1721	3615			1721	1886		
La35	30x10	1888	3716			1888	434		
La36	15x15	1268	1826	1297	573	1293	5346		
La37	15x15	1397	1860	1447	578	1433	5287		
La38	15x15	1196	1859	1251	570	1215	5480		
La39	15x15	1233	1869	1251	567	1248	5766		
La40	15x15	1222	2185	1252	555	1234	5573		

Table 43: Comparison of heuristic methods on instances

D Generating job shop scheduling problems

In this chapter, the method of generating job shop scheduling problems is described. First, static job shop scheduling problem generation is explained, based on Taillard [19]. Next, the adjustment of these problems is explained to create dynamic job shop scheduling problems based on the usage of static job shop problems.

D.1 Generating static problems

Generating static job shop scheduling problems can be done with a simple algorithm, given in Algorithm 3. Here, N is the number of problems to create, J the total number of jobs, M the total number of machines, t_l the lowest possible processing time, t_u the highest possible processing time and S the seed number. The processing time is determined with a uniform distribution, as per Taillard [19]. Algorithm 3 uses inputs that describe the job shop in terms of number of jobs, machines and the range for the processing times for the operations. With this job shop data, the algorithm creates two distinct matrices, one storing information on the machine order to go through for each job, and the other one to store the corresponding processing times, together becoming the set of operations. Next, the algorithm runs through each job, using a random permutation to create an order of machines, after which the algorithm adds a processing time per machine. Afterwards, these matrices are combined to create a job shop scheduling problem similar to Figure 4 in Chapter 3.1. From line 15 onward, the processing time for each job as well as each machine are calculated, to determine the lowest possible makespan for the created job shop scheduling problem, also given in Eq. 30. When the algorithm is completed, using a seed (usually 0) for the random number generator (RNG) to ensure the problems are reproducible, the algorithm has created N number of job shop scheduling problems.

Algorithm 3 Data Generation for Generalization Testing

1: **Inputs:** (N, J, M, t_l, t_u, S)
2: **Set** random number generator seed to S
3: $Instances \leftarrow []$
4: **for** $n = 1$ **to** N **do**
5: **Initialize matrices**
6: $MachineOrder[J, M], ProcessingTimes[J, M] \leftarrow$ empty matrices
7: **for** $j = 1$ **to** J **do**
8: $MachineOrder[j] \leftarrow$ random permutation of M
9: **for** $m = 1$ **to** M **do**
10: $ProcessingTimes[j, m] \leftarrow$ random processing time between t_l and t_u
11: **end for**
12: **end for**
13: **Combine machine order and processing times into Matrix**
14: $Matrix \leftarrow$ combine($MachineOrder$, $ProcessingTimes$)
15: **Calculate job and machine processing times**
16: $jobProcessingTimes[J], machineProcessingTimes[M] \leftarrow [], []$
17: **for** $j = 1$ **to** J **do**
18: $jobProcessingTimes[j] \leftarrow$ sum of processing times for job j in $Matrix$
19: **end for**
20: **for** $m = 1$ **to** M **do**
21: $ProcessingTimesOnMachine \leftarrow$ find all occurrences of machine m in $Matrix$
22: $machineProcessingTimes[m] \leftarrow$ sum of $ProcessingTimesOnMachine$
23: **end for**
24: **Compute lower bound (LB)**
25: $LB \leftarrow \max(\max(jobProcessingTimes), \max(machineProcessingTimes))$
26: **Store Matrix and LB for current instance**
27: $Instances$.append(($Matrix$, LB))
28: **end for**
29: **return** $Instances$

To calculate the lower bound of the makespan found per problem, Eq. 30 is given. Here, $d[i,j]$ is the processing time for operation i of job j, n is the number of jobs, m is the number of machines, $\sum_{i=1}^{m} d[i,j]$ is the total processing time required by job j and $\sum_{j=1}^{n} d[i,j]$ is the total processing time required on machine i.

$$\text{LB} = \max\left(\max_{j}\left(\sum_{i=1}^{m} d[i,j]\right), \max_{i}\left(\sum_{j=1}^{n} d[i,j]\right)\right) \tag{30}$$

With Algorithm 3 problems can be created and defined to train and test the agents on, with the ability to compare the performance with the help of Eq. 30. Lastly, Taillard uses Tabu Search to find solutions and assess the complexity of the generated problems. However, for this study no Tabu search will be used after generating these problems.

D.2 Creating dynamic problems from static JSSP

To transition from static JSSP to dynamic JSSP, arrival times, due dates and priority values are defined. After these are defined, pseudo-code is given to describe the generation process in Algorithm 4. Here, no lower bound will be calculated and no minimal slack and tardiness are determined.

D.2.1 Generating arrival times

In dynamic job shop scheduling, arrival times are generated using an arrival rate λ, representing the average number of jobs arriving per time unit. An arrival rate $\lambda = 0.3$ indicates a probability of 30 percent for a job arriving at each time unit. Based on the size and capacity of the job shop, λ should be adjusted to prevent overloading the system and ensure solvable job shop scheduling problems.

D.2.2 Assigning priority values

The priority values are defined as weights. The weights are given in a distribution of 1 to 3, where the distribution changes dependent on the definition of the job shop. A weight of 1 means a high priority, while a weight of 3 means a low priority. These are thus used to adjust the due dates. These could also be used for actions defined for choosing jobs with the highest priority.

D.2.3 Defining due dates

Due dates defined in job shop scheduling can be done with the use of the arrival time of the job and the total processing time for the job to finish production. Pinedo [21] uses disjunctive graphs and with that, taking the longest route for each job, subtracting the total processing time and adding a value called the slack time. The slack time is a variable used to give the job more slack, which is before the due date that the job is finished. Pinedo uses a slack time of 24 with small job shops. This value is very dependent on both the processing times in the job shop as well as the size of the job shop. Having a high slack time in comparison to the processing times results in having no tardiness, while having a relatively low slack time results in not being able to process any jobs without tardiness. As in this research no disjunctive graph is used, the method to determine the due dates is based on using slack time, with an equations for the shortest route a job can take as well as its priority.

The priority values, or weights, are used to determine a different due date per priority. The weight is determined with a range of 1 to 3. Here, 1 is the highest priority and 3 the lowest. Hence, for the dynamic job shop scheduling problems, the due date is determined by Eq. 31. Here, d_j is the due date of the job j, n is the number of operations, i the value for the operation,t_a the arrival time, t_s the slack time and w_j the weight. The weight is divided by 2 to accommodate for the priority, increasing the due date for low priority and decreasing the due date for high priority.

$$d_j = t_a + \sum_{i=1}^{n} p_{i,j} + t_s \frac{w_j}{2} \tag{31}$$

D.2.4 Algorithm for generating dynamic problems

In Algorithm 4 the method on how dynamic job shop problems are generated is given. Here, the inputs are the static job shop scheduling problem, for example the problem ft06, where for each job an arrival time, due date and weight is added, in that order, to the static job shop problem to make it a dynamic job shop scheduling problem. Again, RNG is used to ensure the problems can be reproduced.

Algorithm 4 Transform Static Job Shop Scheduling to Dynamic Job Shop Scheduling

Inputs: $(StaticData, ArrivalRate, SlackTime, S)$
Set random number generator seed S
$DynamicData \leftarrow$ empty matrix
for each $Instance$ **in** $StaticData$ **do**
  $ArrivalTime \leftarrow 0$
  for each Job **in** $Instance$ **do**
    if Job is the first job **then**
      $ArrivalTimeJob \leftarrow ArrivalTime$
    else
      $ArrivalTimeFound \leftarrow$ **false**
      while not $ArrivalTimeFound$ **do**
        $RandomNumber \leftarrow$ generate random number between 0 and 1
        if $RandomNumber \leq ArrivalRate$ **then**
          $ArrivalTimeJob \leftarrow ArrivalTime$
          $ArrivalTimeFound \leftarrow$ **true**
        end if
        Increment $ArrivalTime$ by 1
      end while
    end if
    $WeightFound \leftarrow$ **false**
    while not $WeightFound$ **do**
      $RandomNumber \leftarrow$ generate random number between 0 and 1
      $ChangeOfNextWeight \leftarrow$ find change of next weight
      if $RandomNumber \leq ChangeOfNextWeight$ **then**
        Determine and add weight for the Job
        $WeightFound \leftarrow$ **true**
      end if
    end while
    Calculate due date based on Equation 31
    $DueDate \leftarrow$ calculate due date using $SlackTime$ and other parameters
    Update Job with computed arrival time, weight, and due date
    Job.update($ArrivalTimeJob$, $Weight$, $DueDate$)
  end for
  Save transformed dynamic job shop problem instance
  $DynamicData$.append($Instance$)
end for
return $DynamicData$

In this chapter detailed methodologies for generating both static and dynamic job shop scheduling problems have been proposed.The algorithms for creating these problems have been discussed and highlighted the considerations necessary for ensuring consistency between static and dynamic setups.

E Requirements and considerations for machine learning method choice

E.1 Requirements for the model

A set of requirements is created, which are used to determine what the needs are and what methods can be applied to meet these requirements. The set of requirements are:

1. The technique has the capability to address and solve a diverse range of job shop scheduling problems, ensuring flexibility and adaptability across various scenarios within a defined range of machines, jobs, and processing times, representing general data of a single job shop.
2. The model should be applicable to both static and dynamic job shop environments, separately.
3. The technique should be relatively straightforward to deploy within custom environments, minimizing the need for specialized expertise.
4. The computational time required to solve the scheduling problems should be minimized to ensure timely solutions in real-world manufacturing contexts. This is crucial for maintaining operational efficiency and responsiveness in dynamic environments.
5. The method should possess the capability to learn (near-)optimal scheduling policies without requiring predefined solutions. This includes the ability to adapt and improve performance through interactions with the scheduling environment, leveraging reinforcement learning principles to refine decision-making over time.

Given these requirements, the goal is to create a model that is able to solve multiple job shop scheduling problems, even without explicitly being trained on a certain problem, within a reasonable time frame. Hence, the created model needs to be able to understand patterns from data, instead of plainly learning steps that lead to a good solution for a specific problem without being a complex and time-consuming method.

E.2 Possible RL and DL methods for the model

Different RL and DL methods are considered for the model, varying in complexity, generalization capabilities, and computational resources required. The complexity of these methods is influenced by the sophistication of their algorithms, learning challenges, data requirements, and the balance of trade-offs (e.g., exploration vs. exploitation in RL, depth vs. computational cost in DL). Generalization capabilities depend on the algorithms, network architectures, and regularization techniques. Computational resource needs are dictated by model complexity, data volume, and training efficiency, often requiring advanced hardware. Based on these characteristics, the methods will be judged.

Proximal Policy Optimization is a policy gradient method for reinforcement learning. It simplifies the process of training stable policies by performing multiple updates on a single set of data, maintaining a balance between exploration and exploitation. The complexity of PPO is high due to the intricate optimization steps involved. This complexity allows PPO to generalize well across various tasks, as it can learn robust policies. However, these benefits come with moderate to high computational demands, owing to the iterative nature of policy updates and the need for extensive data processing.

Multi-Agent Reinforcement Learning extends traditional RL to environments with multiple interacting agents. Each agent learns to optimize its own policy while coordinating with others, which significantly increases the overall complexity. The coordination and synchronization of multiple agents require sophisticated algorithms and substantial computational resources, leading to very high computational time. MARL excels in generalization, as it can adapt to a wide range of tasks involving multiple agents, making it suitable for complex, dynamic environments.

Graph Neural Networks are designed to process graph-structured data, making them highly effective in applications where relationships between entities are crucial. The complexity of GNNs is

high because they involve operations on nodes and edges of graphs, requiring advanced computational techniques. GNNs exhibit excellent generalization capabilities due to their ability to capture the dependencies and structures within data. However, this comes at the cost of moderate to high computational time, as processing graphs can be resource-intensive.

Convolutional Neural Networks are widely used for image and video processing tasks. They utilize convolutional layers to detect patterns and features in data, making them moderately complex compared to other DL methods. CNNs offer moderate generalization, performing well in visual recognition tasks but less so in others. The computational time for CNNs is moderate, as the convolution operations are relatively efficient, but deeper networks can increase computational requirements.

RNNs and LSTMs are specialized for sequential data, such as time series or natural language processing. They maintain internal states that capture temporal dependencies, which increases their complexity. Both RNNs and LSTMs provide good generalization in tasks involving sequences but require high computational resources due to their sequential nature and the need to process long sequences of data.

Deep Q-learning Neural Network combines Q-learning with deep neural networks to approximate value functions. This approach balances the simplicity of Q-learning with the powerful function approximation capabilities of neural networks, resulting in moderate complexity. DQN generalizes well across different environments by learning policies that can be transferred to similar tasks. The computational time for DQN is moderate, as it involves training neural networks, but it is more efficient than methods like PPO or MARL.

Method	**Complexity**	**Generalization**	**Computational Time**
PPO [82]	High	Good	Moderate to High
MARL [83]	Very High	Excellent	Very High
GNN [84]	High	Excellent	Moderate to High
CNN [85]	Moderate	Moderate	Moderate
RNN & LSTM [86]	High	Good	High
DQN [40]	Moderate	Good	Moderate

Table 44: Comparison of different possible DL and RL methods

Based on the findings shown in Table 44, CNN and DQN appear to be the best in terms of complexity and computational time, which are required to be as low as possible. However, CNN performs worse in tasks other than visual recognition tasks. Hence, DQN is chosen as the method for the model.

F Activation functions for DQN

To choose the activation layer, the initialization of the fully connected layers needs to be understood. The weights of the layers are initialized by a Gaussian distribution, given in Eq. 32, where x is the variable, μ is the mean, σ is the standard deviation, exp denotes the exponential function and π is the constant pi. At initialization, the mean μ is zero. The standard deviation σ can be arbitrary, chosen based on empirical evidence or network requirements.

$$\mathrm{f(x)} = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{32}$$

In Table 45 a list of possible activation functions that are well suited for the DQN to solve JSSP is given. This is used to base the choice of activation layer on.

Activation Function	**Properties**
Sigmoid	Output range: $(0, 1)$ Smooth gradient Suitable for binary classification Suffers from vanishing gradient problem
tanh	Output range: $(-1, 1)$ Stronger gradients than sigmoid Effective in training deep neural networks
ReLU	Output range: $[0, \infty)$ Computationally efficient Addresses vanishing gradient problem
Softmax	Converts raw scores into probabilities Suitable for multi-class classification Ensures sum of output probabilities adds up to 1

Table 45: Common Activation Functions and Their Properties

In the context of the JSSP using DQN, the choice of activation function is best to be ReLU. ReLU shows the highest computational efficiency, compared to tanh and sigmoid. Hence, it is suitable for training neural networks efficiently, crucial for both RL dealing with complex networks as well as for the goal to balance the computational time and performance for the model.

ReLU helps address the vanishing gradient problem by avoiding saturation in the positive range, occurring in sigmoid and tanh. Therefore, faster convergence is possible during training, beneficial for the stability and effectiveness of the DQN. Finally, ReLU sets all negative values to zero, which leads to sparse activations. The sparsity can help in learning more robust and efficient representations, especially for high-dimensional input spaces, encountered in JSSP. Leaky ReLU is also a good alternative, addressing the "dying ReLU" problem by allowing small gradients for negative inputs. However, traditional ReLU performs well in many cases and is simpler to implement. Hence, ReLU is best fitting for the specific problem and method.

F.1 Solving Dying ReLU problem

The "dying ReLU" problem in neural networks occurs when a significant number of ReLU neurons become inactive, by constantly outputting a value of zero during training. As ReLU outputs a value between zero for negative values, this occurs when the weighted input to a ReLU neuron is negative. This leads to zero gradients which "kills" the neurons, no longer contributing to the learning process. This can degrade the performance of the network, especially in deep architectures.

To accommodate for the dying ReLU problem, the weight initialization given in Eq. 32 can be adjusted. Some methods are using different weight initialization techniques, being:

- **Zero Initialization:** Setting all weights to zero can lead to symmetry breaking issues where all neurons in a layer behave identically. This can hinder the learning process as neurons fail to learn unique features. Therefore, zero initialization is generally avoided.

- **Random Initialization:** Initializing weights randomly from a small Gaussian or uniform distribution helps break symmetry and introduces diversity in the network. Common approaches include Xavier/Glorot initialization and He initialization.
- **Xavier/Glorot Initialization:** This method initializes weights from a Gaussian distribution with zero mean and variance calculated based on the number of input and output units of the layer. It helps in keeping the activations and gradients within a reasonable range during training.
- **He Initialization:** Similar to Xavier initialization, He initialization sets the variance of the Gaussian distribution based on the number of input units. It is commonly used with activation functions like ReLU to address the dying ReLU problem.
- **LeCun Initialization:** This method initializes weights using a Gaussian distribution with a mean of zero and a variance that scales with the number of input units. It is particularly effective for networks with sigmoid or hyperbolic tangent activation functions.
- Orthogonal Initialization: Initializing weight matrices with orthogonal matrices helps in ensuring that the activations and gradients do not shrink or explode during training. This technique can be particularly beneficial for deep networks.

. For the DQN implementation, He initialization is a suitable choice to solve the dying ReLU problem. This method is designed to keep the variance of the activations consistent across layers, thereby addressing the dying ReLU problem.

He initialization sets the initial weights of a layer using a Gaussian distribution with zero mean and a variance of $\frac{2}{n_{\text{in}}}$, where n_{in} is the number of input units in the layer. This is mathematically represented in Eq. 33.

$$W \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right) \tag{33}$$

Using He initialization, we ensure that the weights are set to appropriate values, preventing neurons from becoming inactive and maintaining effective training of the neural network.

G Steps to Deploy the DQN Method in MATLAB

To deploy a DQN agent in MATLAB, these steps are taken:

1. **Define Observation and Action Spaces:** Specify the dimensions of the observation and action spaces based on your environment.
2. **Initialize the Environment:** Set up the simulation environment for the agent.
3. **Define Neural Network Architecture:**
   - Set the number of neurons and layers.
   - **Input Layer:** Matches the observation space.
   - **Hidden Layers:** As defined in the architecture.
   - **Output Layer:** Represents the action space, providing Q-values.
4. **Initialize the Network:**
   - Construct the network using the defined layers.
   - Convert the network with the `dlnetwork` function for agent use.
5. **Create the DQN Agent:**
   - Use the `rlDQNAgent` function to set up the agent with the neural network, observation, and action spaces.
   - Configure agent options such as exploration strategy, learning rate, and discount factor.
6. **Train the Agent:** Use the `train` function to train the agent within the defined environment.
7. **Save and Test the Agent:** Save the trained agent and test its performance.

H Defining the environment

In this Appendix, definitions are given to the general outline of an environment. Next, variables are defined to create new reward structures, tested in a dynamic environment. Finally, an inefficiency in the scheduling of the environment is addressed.

H.1 General coding structure of an environment

A general outline for creating a working environment is presented in Algorithm 5.

Algorithm 5 General environment pseudo-code in RL

```
Initialize environment parameters
Define state representation
Define reward function
Define action representation
Function step(action):
    perform_action(action)
    next_state ← get_next_state()
    reward ← calculate_reward()
    done ← check_termination_condition()
    return next_state, reward, done

Function perform_action(action):
    Execute the action in the environment

Function get_next_state():
    Obtain the next state of the environment after the action is performed

Function calculate_reward():
    Calculate the reward based on the current state and next state

Function check_termination_condition():
    Check if the termination condition of an episode is met
```

This pseudo-code can be implemented in combination with the reinforcement learning toolbox of MATLAB to create a custom environment using the rlCreateEnvTemplate function [87]. As the action and state space and the reward signal have already been defined, the remaining components to define are the reset function and the step function to determine interactions between the state and action space.

H.2 Creating reward structures for dynamic job shops

Each reward structure leverages a different signal, thus creating a different incentive for the agent to learn. First, definitions are given to introduced variables. Hereafter reward structures for just-in-time as well as maximizing slack are proposed.

H.2.1 Definitions used for the rewards

To define rewards, a couple of definitions are introduced for the dynamic environment. In adapting to the dynamic environment, the problem is approached as a multi-objective task. Two primary objectives are distinguished: Just-In-Time (JIT), which aims to minimize makespan and tardiness while maximizing slack, and another approach that focuses on slack minimization.

Tardiness and slack are defined in Chapter 3.4.2 and are expressed by Eq. 2 ad Eq. 3 respectively.

To provide the agent with a learning incentive for each step taken, the expected completion time (ECT) is defined. ECT considers the utilization rates of machines $M_{i,k}$, processing times $P_{j,k}$, and the earliest starting time t_j for the next operation, depicted in Eq. 34.

$$ECT_j = \sum_k P_{j,k} \cdot \frac{1}{\mu_{i_k}} + t_j \tag{34}$$

From ECT_j, an expected tardiness ET_j, given by Eq. 36, and expected slack ES_j, given by Eq. 35, can be derived.

$$ES_j = \max(0, d_j - ECT_j) \tag{35}$$

$$ET_j = \max(0, ECT_j - d_j) \tag{36}$$

H.2.2 Just-in-time rewards

Specific reward structures are formulated for JIT training, aimed at minimizing both tardiness and slack to optimize scheduling efficiency. Four distinct reward formulas are proposed, each designed to incentivize reductions in slack and tardiness deviations. Eq. 37 penalizes deviations in both expected slack (ES) and expected tardiness (ET) when a job is not finished. Upon job completion, rewards are based on the actual differences in slack (S) and tardiness (T).

$$r_t = \begin{cases} -\Delta\text{ES} - \Delta\text{ET} & \text{if job not finished} \\ -(S_j - ES_j) - (T_j - ET_j) & \text{if job finished after action} \end{cases} \tag{37}$$

Eq. 38 is similar to Eq. 37, but penalizes only the actual slack and tardiness when a job is finished, without considering expected values.

$$r_t = \begin{cases} -\Delta\text{ES} - \Delta\text{ET} & \text{if job not finished} \\ -S_j - T_j & \text{if job finished after action} \end{cases} \tag{38}$$

Eq. 39 includes the utilization rate (Δu_{jk}) as an additional incentive to reduce slack and tardiness deviations when a job is not finished. Upon job completion, rewards consider both deviations and utilization rate effects.

$$r_t = \begin{cases} -\Delta\text{ES} - \Delta\text{ET} + (\Delta u_{j_k}) & \text{if job not finished} \\ -(S_j - ES_j) - (T_j - ET_j) + (\Delta u_{j_k}) & \text{if job finished after action} \end{cases} \tag{39}$$

Eq. 40 is similar to Eq. 39 but penalizes only the actual slack and tardiness, along with the utilization rate effect, when a job is finished.

$$r_t = \begin{cases} -\Delta\text{ES} - \Delta\text{ET} + (\Delta u_{j_k}) & \text{if job not finished} \\ -S_j - T_j + (\Delta u_{j_k}) & \text{if job finished after action} \end{cases} \tag{40}$$

These reward structures will be evaluated in a dynamic job shop scheduling environment to determine their effectiveness in achieving JIT objectives.

H.2.3 Rewards for maximizing slack

To maximize slack in job shop scheduling, reward structures are formulated similarly to those for JIT but with a focus on rewarding slack increases rather than penalizing deviations.

Eq. 41 gives rewards based on the differences in expected slack (ES) and expected tardiness (ET) when a job is not finished. Upon job completion, rewards are based on the actual differences in slack (S) and tardiness (T).

$$r_t = \begin{cases} \Delta\text{ES} - \Delta\text{ET} & \text{if job not finished} \\ (S_j - ES_j) - (T_j - ET_j) & \text{if job finished after action} \end{cases} \tag{41}$$

Eq. 42 is similar to Eq. 41 but rewards only the actual slack and tardiness when a job is finished, without considering expected values.

$$r_t = \begin{cases} \Delta\text{ES} - \Delta\text{ET} & \text{if job not finished} \\ S_j - T_j & \text{if job finished after action} \end{cases} \tag{42}$$

Eq. 43 includes the utilization rate (Δu_{jk}) as an additional incentive to increase slack when a job is not finished. Upon job completion, rewards consider both slack and tardiness deviations along with utilization rate effects.

$$r_t = \begin{cases} \Delta\text{ES} - \Delta\text{ET} + (\Delta u_{j_k}) & \text{if job not finished} \\ (S_j - ES_j) + (T_j - ET_j) + (\Delta u_{j_k}) & \text{if job finished after action} \end{cases} \tag{43}$$

Eq. 44 is similar to Eq. 43 but rewards only the actual slack and tardiness, along with the utilization rate effect, when a job is finished.

$$r_t = \begin{cases} \Delta\text{ES} - \Delta\text{ET} + (\Delta u_{j_k}) & \text{if job not finished} \\ S_j - T_j + (\Delta u_{j_k}) & \text{if job finished after action} \end{cases} \tag{44}$$

Each of these reward structures will undergo testing in a dynamic job shop scheduling environment to evaluate their effectiveness and suitability.

The machine learning technique, Deep Q-Network, and the environment, defined as a Markov Decision Process, have been discussed. In the next chapter, DQN will be implemented in MATLAB to explore this environment further, with additional details on both the DQN and environment.

H.3 Initializing the environment: reset function

The reset function is used to reset the whole environment. This is done to initialize and re-initialize the environment for the start of each episode. Important parts of the initialization, and thus for the reset function, are:

- Determine the number of jobs and machines in the job shop.
- Based on the number of jobs and machines, pre-allocate a matrix for scheduling jobs.
- Based on the job shop problem data given, create the initial current state.
- Initialize other variables needed to plan jobs and their operations.

As these have been defined, the basics of the reset function are defined.

H.4 Addressing scheduling inefficiency for static job shop scheduling

Based on tests done on the dispatching rules, it is found that they do not work very well in the created environment. When comparing the outcomes of the dispatching rules on the problem ft06 found in our environment in comparison with the outcome found in research by Liu [81], the performance differs. This is shown in Table 46.

dispatching rule	SPT	LPT	FIFO	LIFO
found makespan [81]	84	73	65	70
Makespan environment	109	129	152	170

Table 46: Makespan comparison of Liu [81] and the created environment on ft06

When analyzing the Gantt chart depicted in Figure 37 together with the problem ft06, given in Chapter 3.1 in Table 4, it is found that there is a possibility of improving upon the Algorithm 1, given in Chapter H.1. The problem is found in the fact that, looking at this specific problem, job 3, needing to start at machine 3, could be planned earlier. However, currently these idle times in between jobs is not taken into account when planning jobs. Hence, the environment needs to be adjusted.

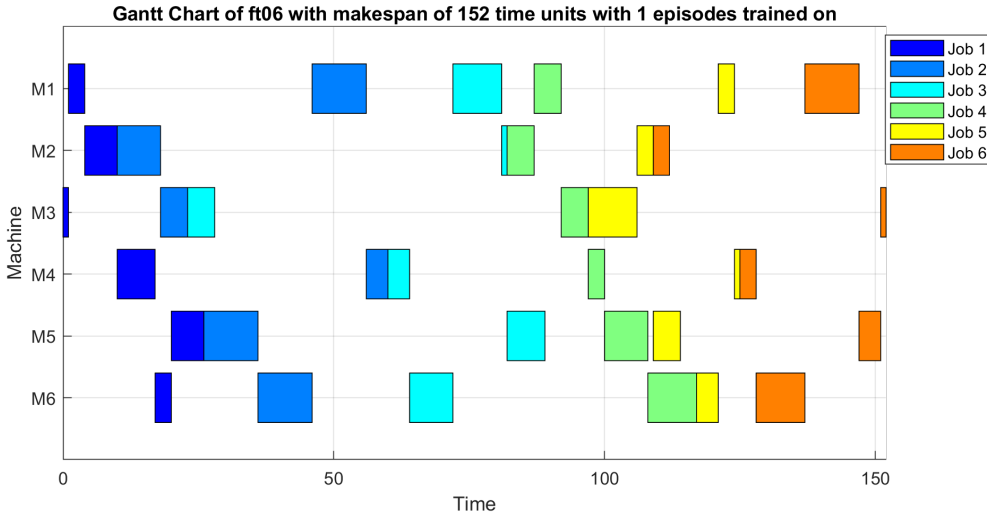


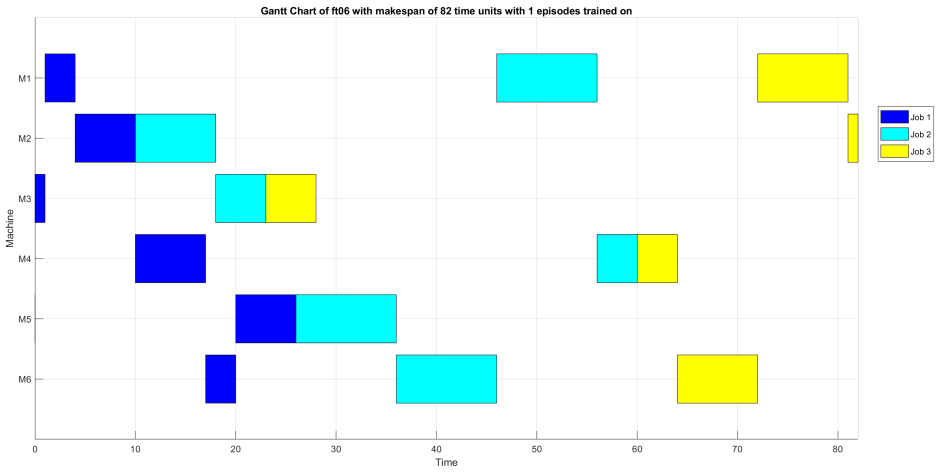
Figure 37: Gantt chart of FIFO on ft06

Algorithm 6 Updated workings of a job shop

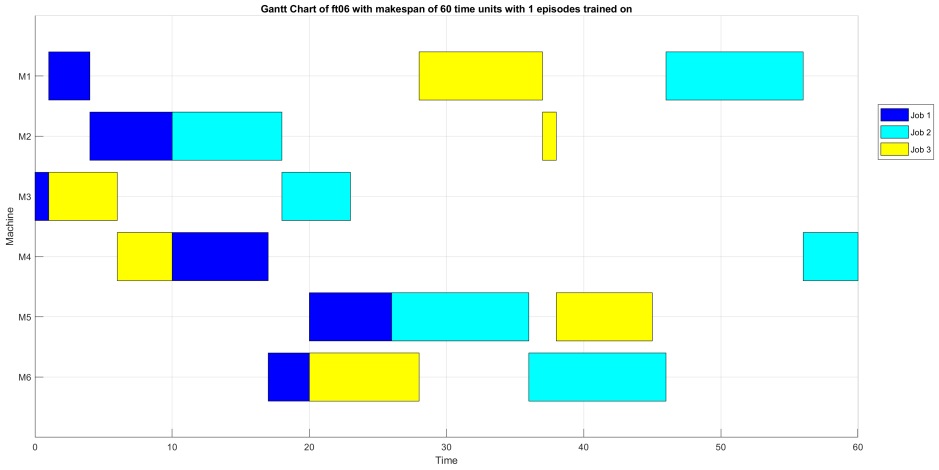
Input: General job shop problem
Output: Scheduled jobs of job shop problem
while not all jobs have been processed **do**
  Choose an action
  Based on the action, determine the job and its operation to schedule and which machine to schedule on
  Determine if there are gaps in the planning for the machine
  if there are gaps in the planning for the machine **then**
    Determine where the gaps in the planning are
    Determine how big the gaps in the planning are
    Determine if the processing time of the operation fits in the gap(s)
    if processing time fits in one of the gaps **then**
      Find the earliest possible gap to plan the job in
      Schedule the job in the gap
    else
      Determine the start and end time:
      if starting time of the job $\geq$ starting time machine **then**
        Starting time of operation = starting time of the job
      else
        Starting time of operation = starting time machine
      end if
      End time = starting time + processing time of operation
      Schedule the job
    end if
  end if
end while

With this updated Algorithm 6, in Figure 38 the change in planning is depicted, where in subfigure 38a the jobs are always planned after jobs that are already planned, subfigure 38b shows the improvement of Algorithm 6 where the jobs are planned in the earliest possible time available in the schedule.

With this change it is found that the found makespan per dispatching rule improves, given in Table 47, while maintaining the correct order of machines the jobs need to planned on, with the updated Gantt chart for the FIFO dispatching rule given in Figure 39.



(a) Situation after planning job 3 for Alg. 1 with makespan of 83



(b) Situation after planning job 3 for Alg. 6 with makespan of 60

Figure 38: Change of planning algorithms for the job shop scheduling

dispatching rule	SPT	LPT	FIFO	LIFO
found makespan [81]	84	73	65	70
Makespan environment	109	129	152	170
Makespan environment updated	83	79	71	86

Table 47: Makespan comparison of Liu [81] and the updated environment on ft06

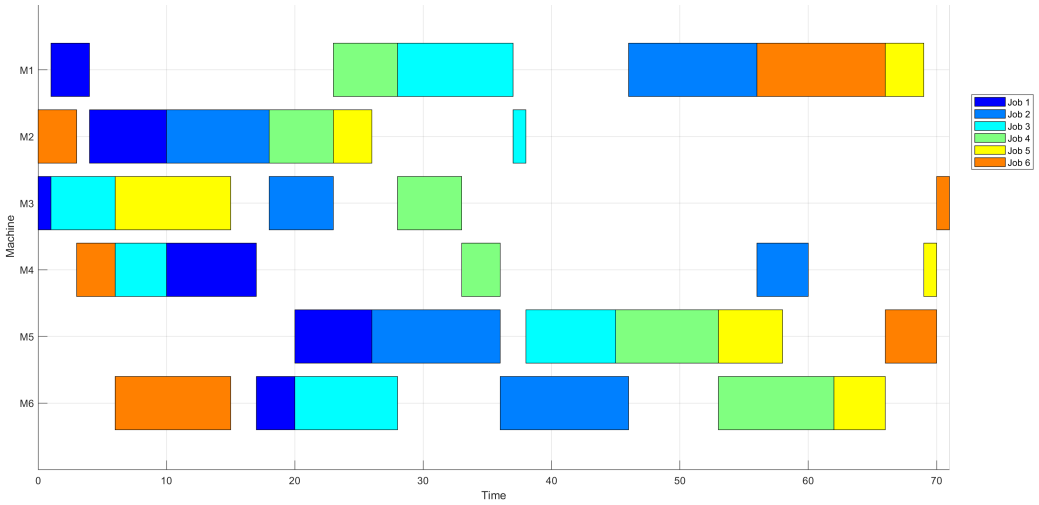
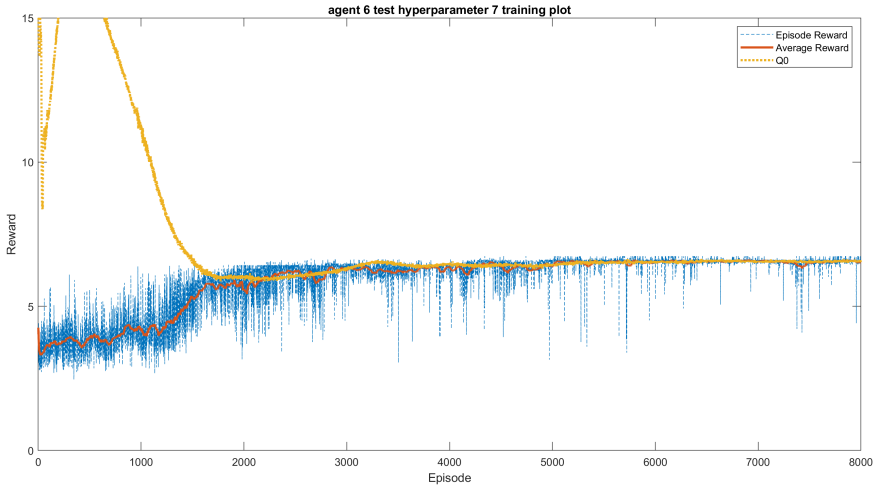
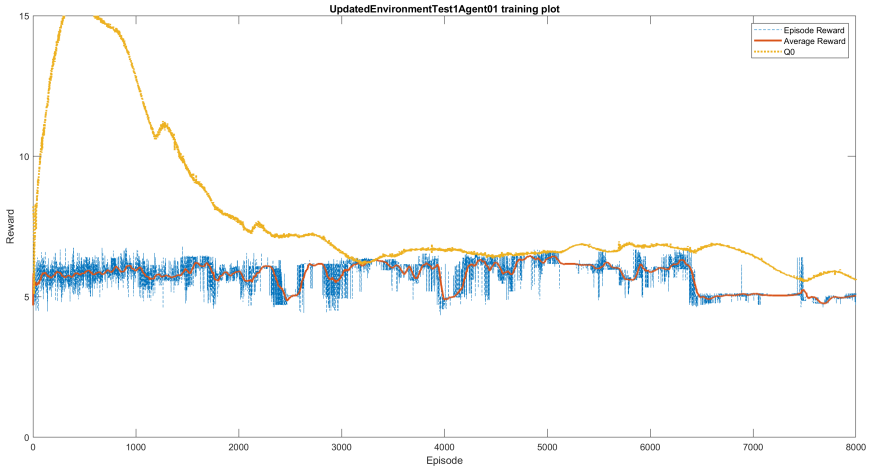


Figure 39: Gantt chart using FIFO with updated environment

When comparing the training with the same hyperparameters but change in scheduling, thus actions having different influences, it is found that the hyperparameters do not work for this change, shown in Figure 40. It is found that the training behavior is less stable, which is due to the difference in how the actions work, thus different rewards signals will be gained which influences the learning. hence, for the remainder of the experiments on static job shop scheduling problems, the environment will not be changed. However, for the dynamic job shop scheduling, this change will be made as the hyperparameters will need to be changed as well to accommodate for the dynamic changes in the environment.



(a) Training graph with old scheduling



(b) Training graph with new scheduling

Figure 40: Change of training graph with same hyperparameters for different scheduling

I Results of first hyperparameter tuning: static job shop scheduling

In this appendix, the steps taken for hyperparameter tuning are described. To determine the best hyperparameters, the learning rate, discount factor, ϵ, ϵ decay, number of neurons, number of hidden layers, number of episodes and a reward scaling factor are changed. The reward scaling factor is introduced to see if a different size of reward signal changes the performance of learning for the model.

Each iteration is described and finally the performance of the chosen hyperparameters is given. The used hyperparameters and variables are also given in Chapter 8.1.2 in Table **??** and Table 15.

I.1 First iteration

For the first iteration, the set of hyperparameters is given in Table 48. Here, a wide range of hyperparameters is taken to assess which individual as well as combinations of hyperparameters give better results in terms of performance and learning behavior.

It.	**LR**	**DF**	ϵ	ϵ_d	**N**	**HL**	**RS**	**E**
1	1e-1, 1e-4, 1e-7	1e-3, 0.5, 1	1e-3, 0.5, 1	1e-2, 1e-6, 1e-10	64, 128, 256	1, 2, 3, 4, 5	1, 10, 100	4000

Table 48: Hyperparameter settings for first iteration

In Figure 41 the first comparison of agents is shown, looking at found makespan, gained reward and average utilization rate of the machines. As can be seen, the performance of the agent greatly differs from one another.

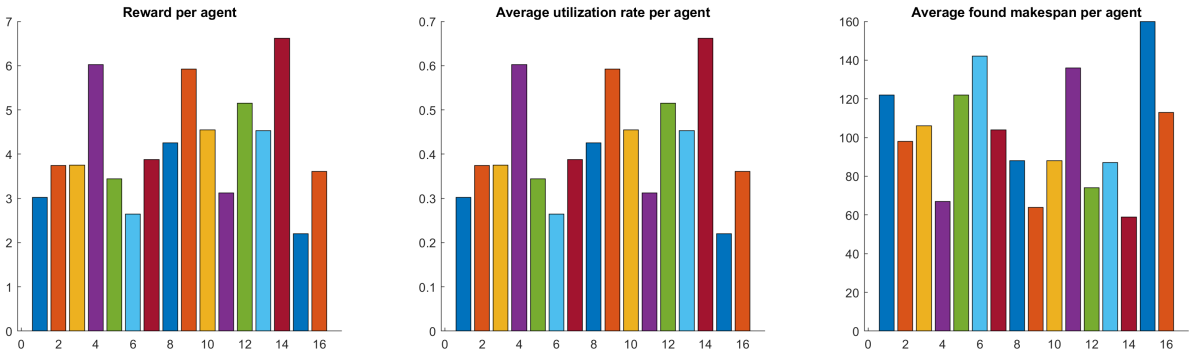
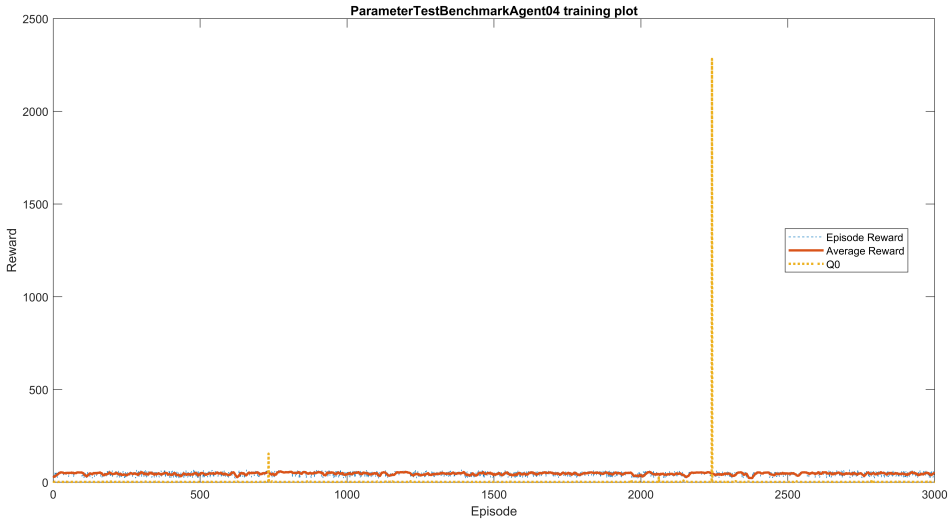
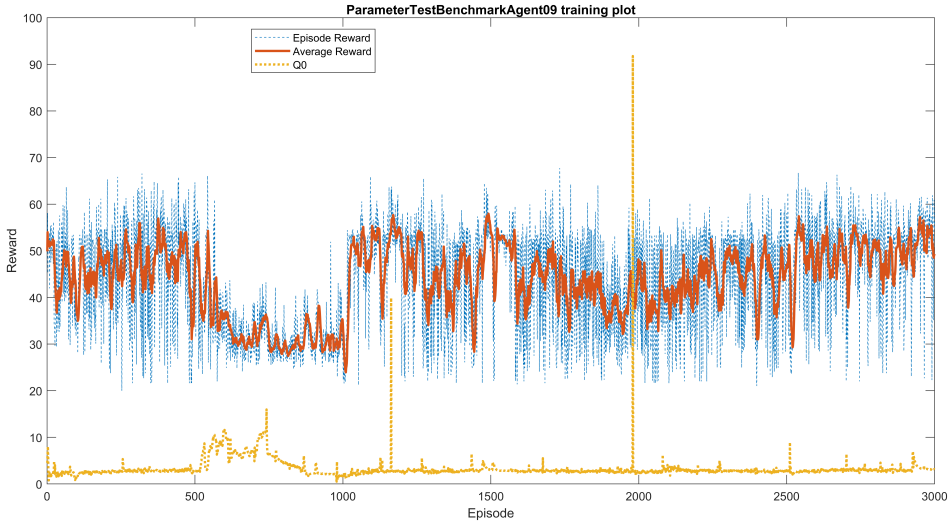


Figure 41: First comparison of created agents

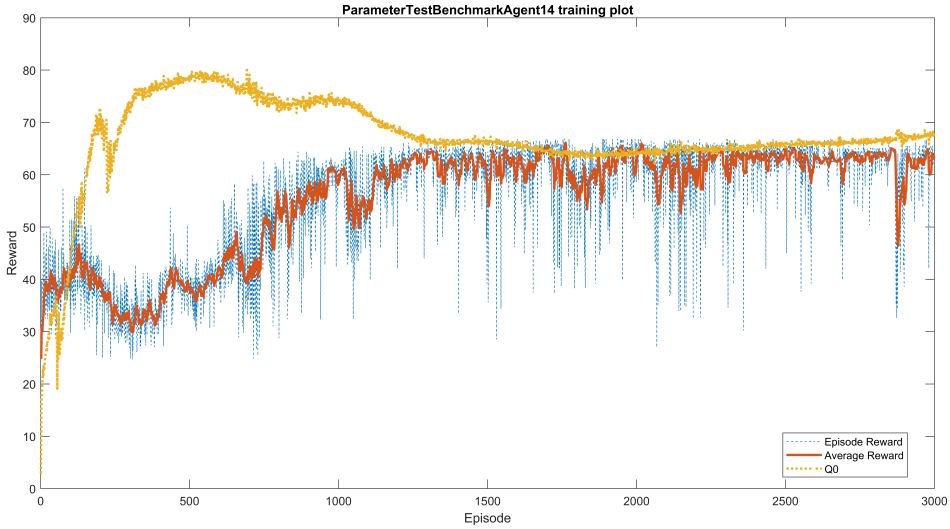
Looking at the performances, the agents 4, 9 and 14 perform best. Hence, the behavior is compared in Figure 42. It can be seen that agent 4's Q0-values are badly estimated with a big overshoot at around 2200 episodes, while agent 9 shows no sign of learning from it's experiences. Lastly, agent 14, while unstable, shows improvement in terms of the reward and the Q0-value estimates become somewhat equal to the gained reward, which shows that the agent learns.



(a) Learning behavior agent 4



(b) Learning behavior agent 9



(c) Learning behavior agent 14

Figure 42: Comparison of learning behavior of agent 4 (a), 9 (b) and 14 (c)

Now looking at their different hyperparameters, given in Table 49, agent 14 should be focused on here. The agent shows the most promising results in both performance as well as training behavior. Hence, the chosen hyperparameters for the next test should be more focused towards the hyperparameters of agent 14.

Agent	Discount factor	Learning rate	ϵ
4	0.0001	0.1	1
9	0.5	0.1	0.001
14	1	1e-4	0.001

Table 49: Data of best agents of the hyperparameter tuning

I.2 Second iteration

For the second iteration, specific data for the agents is tracked, given in Table 50, where ϵ_d represents the decay of epsilon and CT depicts the computational time in seconds.

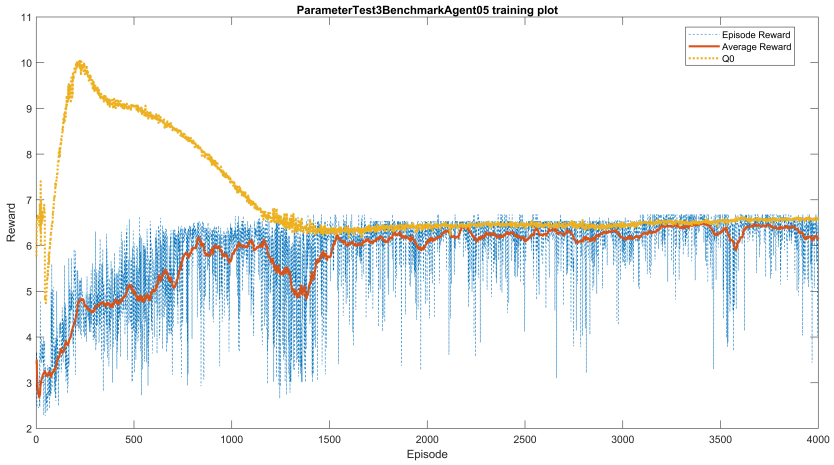
Agent	LR	DF	ϵ	ϵ_d	Neurons	Hidden layers	CT (s)
1	1e-4	1e-3	0.5	1e-6	64	2	990.7478
2	1e-5	0.5	0.5	1e-10	128	4	1505.1
3	1e-3	1e-3	1e-3	1e-6	64	4	1223
4	1e-3	1e-3	1e-3	1e-6	128	5	1461.8
5	1e-5	1	1e-3	0.01	256	4	3267.7
6	1e-4	1	1e-3	1e-10	64	5	1396.1
7	1e-4	1e-3	0.5	1e-6	256	3	1714
8	1e-3	1e-3	1e-3	1e-6	64	4	1164.6
9	1e-4	1e-3	1e-3	1e-6	256	2	1461.4
10	1e-4	0.5	1e-3	1e-6	128	2	1170
11	1e-4	0.5	1e-3	1e-10	128	4	1329.1
12	1e-3	1e-3	1	0.01	256	5	2805.9
13	1e-3	0.5	0.5	1e-10	64	4	1138.3
14	1e-5	1e-3	0.5	1e-10	128	3	1249.2
15	1e-3	1e-3	1	1e-6	128	2	1135.8
16	1e-3	1e-3	1e-3	1e-10	256	3	2113.8
17	1e-4	1e-3	0.5	1e-6	256	3	1570.1
18	1e-3	1	1e-3	1e-6	128	2	1820.7
19	1e-5	0.5	1	1e-10	256	5	1765.6
20	1e-4	1	0.5	1e-6	128	2	1464
21	1e-3	1e-3	1	0.01	64	5	1626.8
22	1e-3	0.5	1	1e-10	64	4	1154.4
23	1e-4	1e-3	1e-3	0.01	64	2	1034.2
24	1e-3	1e-3	0.5	1e-6	256	4	2300
25	1e-3	1e-3	1e-3	1e-6	256	2	1581.3
26	1e-4	1	0.5	1e-10	128	4	1435.4
27	1e-4	1e-3	0.5	1e-6	128	3	1283.5
28	1e-5	0.5	0.5	0.01	64	2	1024.5
29	1e-3	0.5	1	0.01	64	2	1061.3
30	1e-4	0.5	0.5	0.01	256	5	4061.6

Table 50: Values of agents in second hyperparameter tuning test

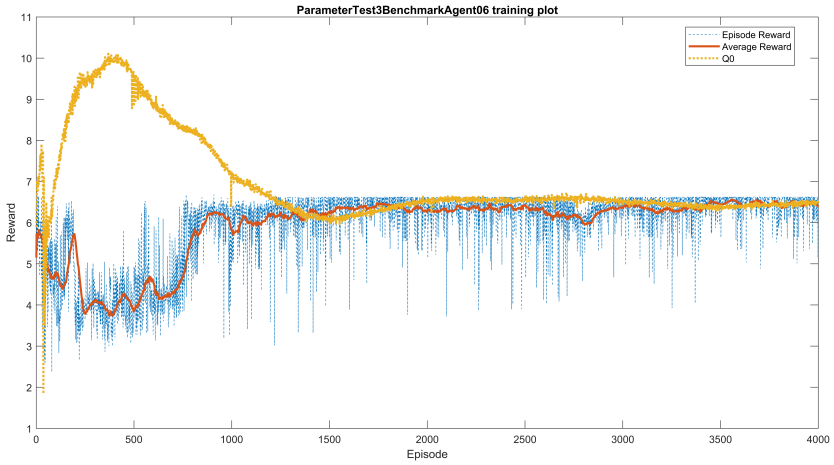
From these agents, the results of testing is given in Figure 43. Based on this comparison it is found that they have a lot of difference in performance, while also some agents have a high computational time (in seconds) in comparison to the other agents.

Figure 43: Comparison of performance of agents in test 2

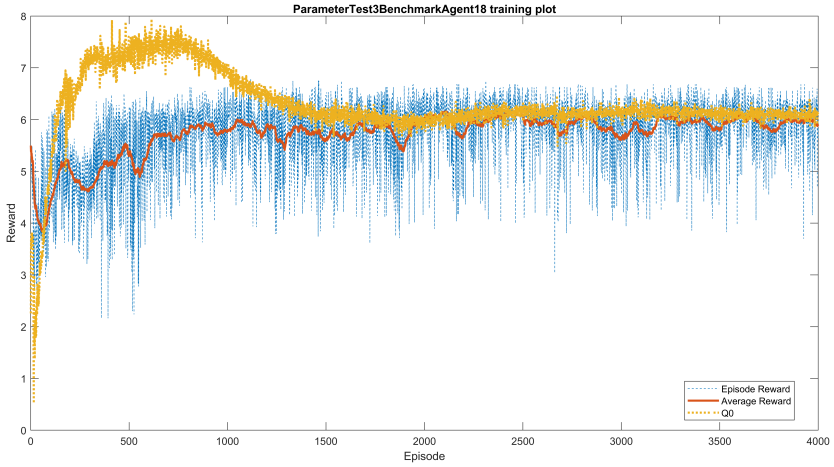
From this figure, the agents 5, 6, 18, 26 and 29 are chosen as being the best performers, finding a makespan of 59, 60, 70, 72 and 67 respectively, with a reward of 6.61, 6.59, 6.22, 5.47 and 5.73 respectively. All of these agents are trained within an hour, which is seen as a reasonable training time. In Figure 44 and 45, the training behavior of these tests are compared. The scales of the graphs are not equal. However, the behavior is not influenced by this scale, as the agents 5, 6 and 18 show wanted behavior in increasing their found reward while also finding overlap in the Q0-values with the found average reward, hence showing that it has learned. Hence, the next iteration will focus more towards the hyperparameters of the agents 5, 6 and 18.



(a) Training graph of agent 5 from test 2

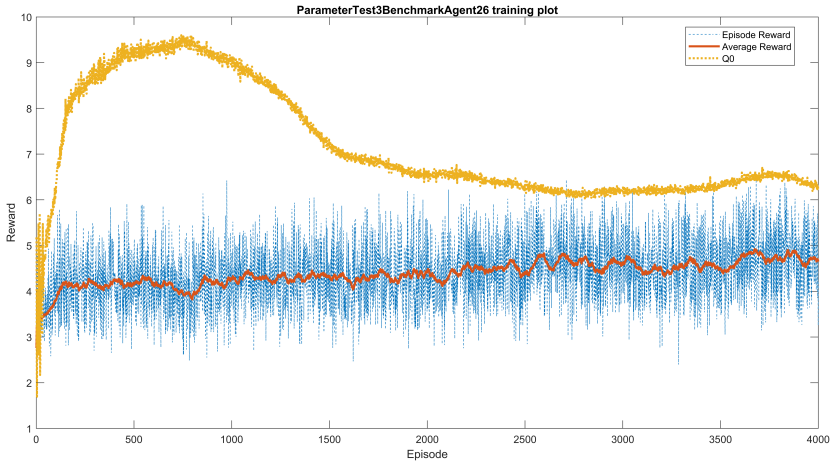


(b) Training graph of agent 6 from test 2

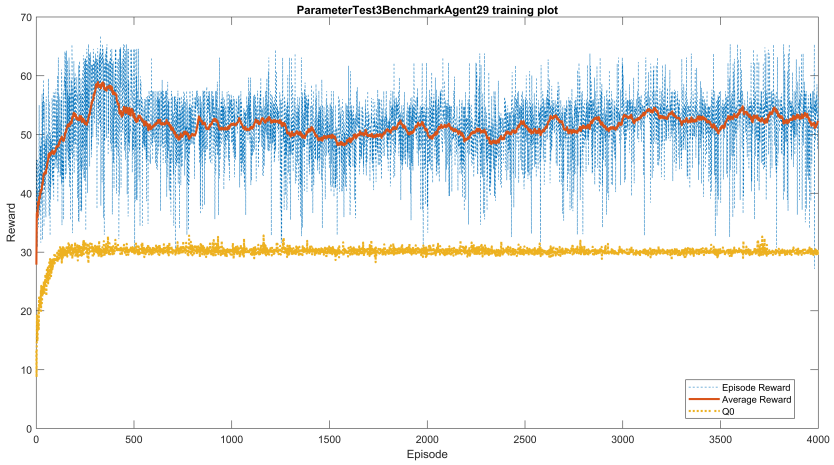


(c) Training graph of agent 18 from test 2

Figure 44: Comparison of training behavior of agents 5, 6, and 18 from test 2



(a) Training graph of agent 26 from test 2



(b) Training graph of agent 29 from test 2

Figure 45: Comparison of training behavior of agents 26 and 29 from test 2

I.3 Third iteration

For this test, the comparison is given in Figure 46.

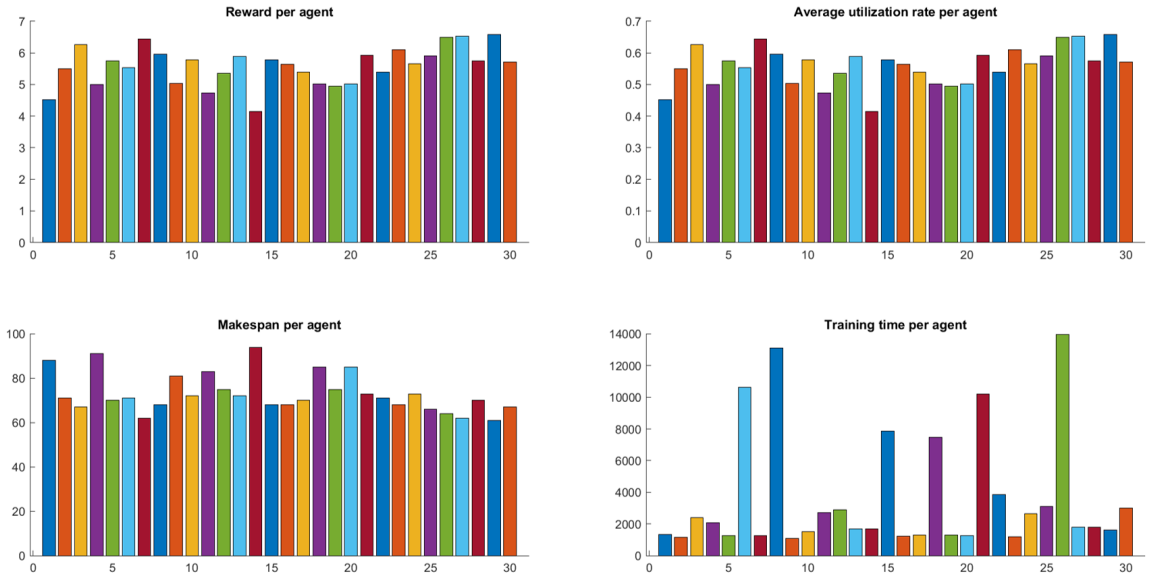


Figure 46: Comparison of performance of agents in third iteration

From these results it was found that overall the agents performed better, but no agent actually performed better than the latter test. From these results the focus in on the agents 7, 26, 27 and 29, which had a makespan lower than 65. Based on those founding the agents with a higher discount factor perform better, with a lower epsilon value, with the agent hyperparameters given in Table 51. Also, 512 neurons greatly increase the computational time. Hence, the choices for the next iterations are based on these findings.

Agent	LR	DF	ϵ	ϵ_d	Neurons	Hidden Layers	CT (s)
1	1E-4	0.5	0.25	1E-6	128	4	1330.94
2	1E-3	0.75	0.1	1E-6	64	4	1169.93
3	1E-3	1	1E-5	1E-10	256	4	2384.79
4	1E-4	0.75	1E-5	1E-10	128	6	2065.79
5	1E-5	0.75	0.1	1E-10	64	6	1256.10
6	1E-5	0.5	0.1	1E-2	512	4	10636.01
7	1E-3	1	0.1	1E-10	64	5	1259.75
8	1E-5	1	0.25	1E-6	512	6	13120.60
9	1E-5	0.5	0.5	1E-10	64	4	1088.11
10	1E-4	1	0.25	1E-6	128	5	1501.83
11	1E-3	0.5	0.25	1E-6	256	5	2711.98
12	1E-5	0.5	0.25	1E-6	256	4	2909.71
13	1E-3	1	0.25	1E-10	128	5	1674.79
14	1E-3	0.5	0.1	1E-6	128	5	1674.89
15	1E-3	1	0.25	1E-6	512	6	7866.81
16	1E-5	0.75	1E-5	1E-10	64	5	1240.58
17	1E-4	0.5	1E-5	1E-10	64	6	1287.47
18	1E-4	0.5	0.1	1E-10	512	4	7478.97
19	1E-5	0.5	0.1	1E-6	64	6	1289.92
20	1E-3	0.5	0.1	1E-2	64	5	1270.89
21	1E-4	0.75	0.1	1E-6	512	5	10198.45
22	1E-4	0.5	0.1	1E-2	256	5	3859.49
23	1E-3	1	0.25	1E-2	64	4	1177.79
24	1E-3	0.5	0.25	1E-2	256	5	2649.49
25	1E-3	0.75	1E-5	1E-10	256	6	3119.70
26	1E-5	1	0.1	1E-6	512	6	13971.34
27	1E-5	1	1E-5	1E-6	128	6	1786.05
28	1E-4	0.75	0.5	1E-2	128	5	1805.29
29	1E-3	1	0.1	1E-2	128	5	1635.33
30	1E-5	0.5	0.1	1E-10	256	4	2993.36

Table 51: Values of agents in third hyperparameter tuning iteration

I.4 Fourth iteration

For the fourth iteration, the comparison of agents is given in Figure 47. From this comparison it is concluded that agents 9 and 10 perform best, finding a reward above 6.6 and a makespan of 61.

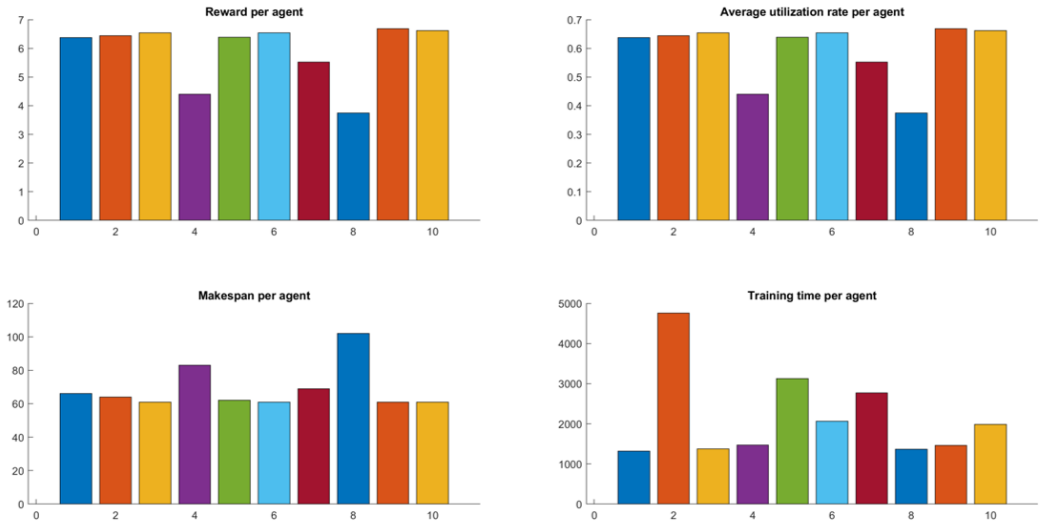
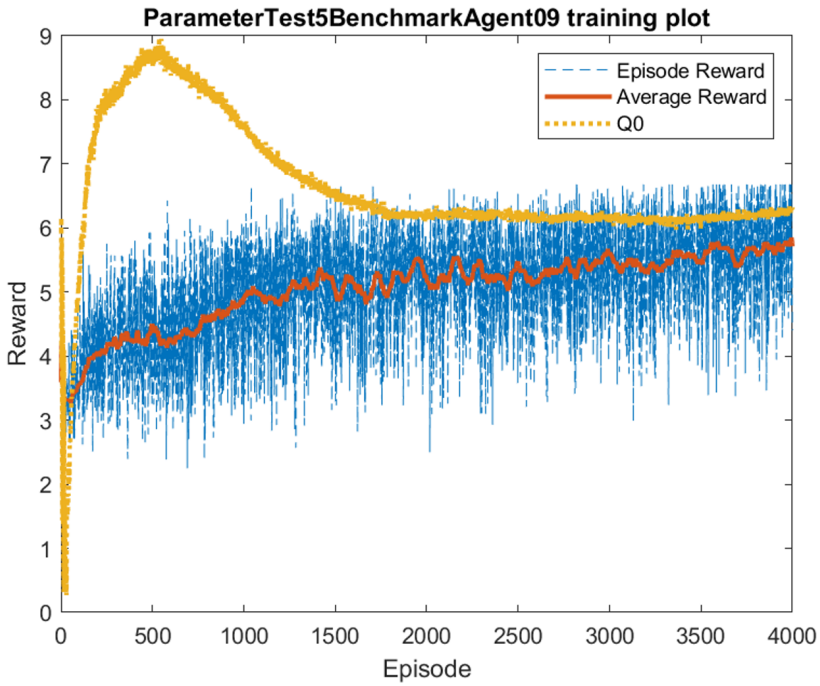
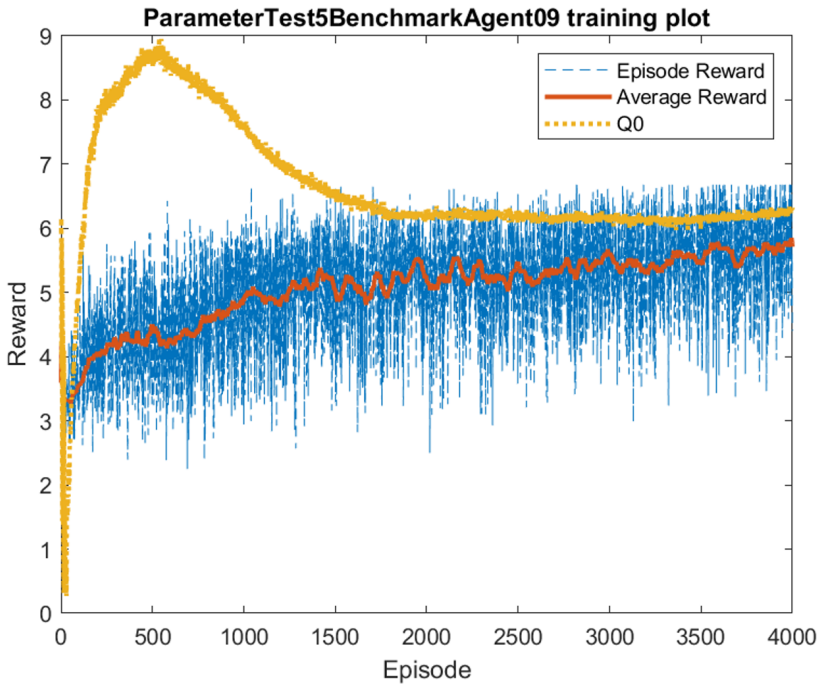


Figure 47: Comparison of performance of agents fourth iteration

In Figure 48, the training graph of these agents is given. It is found that for the 4000 episodes, there still might be some learning to be done after those 4000 episodes, based on the two graphs. Hence, after these tests, the episodes done is scaled up to 8000. In Table 52 the parameters for the agents in this test have been given.



(a) Training graph of agent 9 from test 4



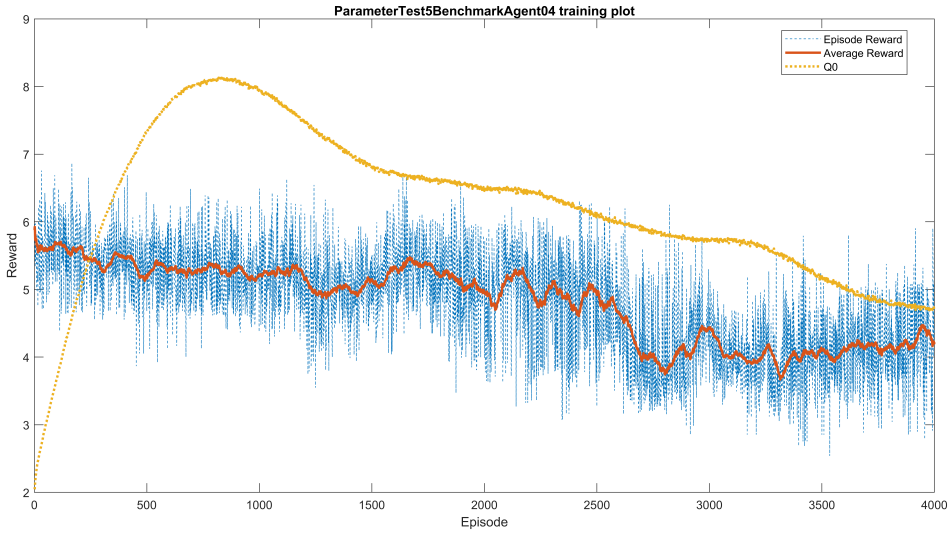
(b) Training graph of agent 10 from test 4

Figure 48: Comparison of training behavior of 2 best performing agents

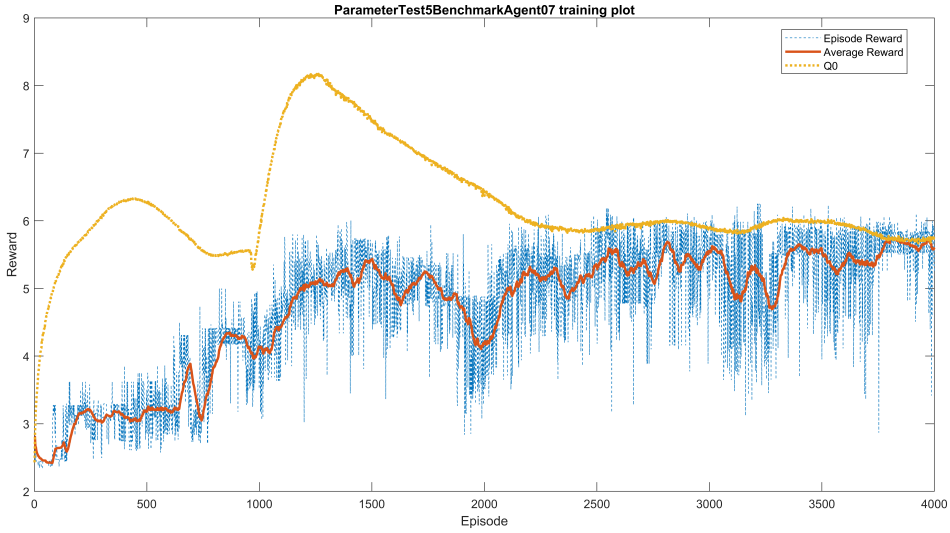
Agent	LR	DF	ϵ	ϵ_d	Neurons	Hidden Layers	CT (s)
1	1E-4	1	0.2	1E-6	64	4	1316.77
2	1E-4	1	0.2	1E-10	256	6	4759.11
3	1E-4	1	1E-5	1E-6	64	4	1371.21
4	1E-6	1	1E-5	1E-2	64	6	1470.96
5	1E-3	1	1E-5	1E-2	256	5	3124.45
6	1E-3	1	1E-5	1E-6	128	6	2066.07
7	1E-6	1	1E-5	1E-10	256	4	2766.70
8	1E-6	1	0.2	1E-10	64	6	1365.82
9	1E-4	1	0.2	1E-6	128	4	1462.81
10	1E-4	1	1E-5	1E-10	128	5	1987.91

Table 52: Values of agents in fourth hyperparameter tuning test

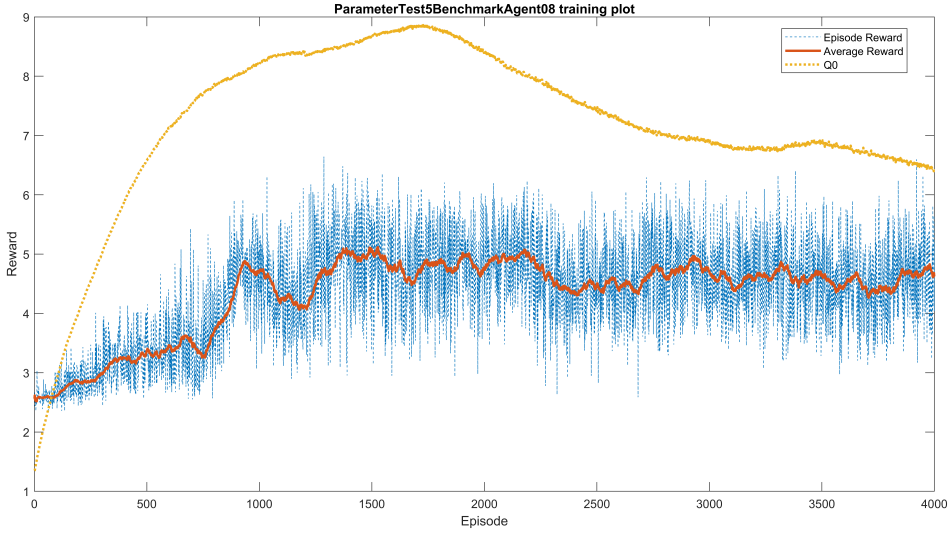
As from these values in combination with the given training graphs no real conclusions can be made yet, in Figure 49. Here, the training graph of agents 4, 7 and 8 are given. When looking at their learning rate, these are the only agents with a learning rate of 1E-6. Hence, it is concluded that the learning rate 1E-6 is not beneficial for the learning behavior of the agents, as these agents show no actual learning and all are unstable. Hence, the learning rate of 1e-6 is removed from the range for the next iteration.



(a) Training graph of agent 4 from test 4



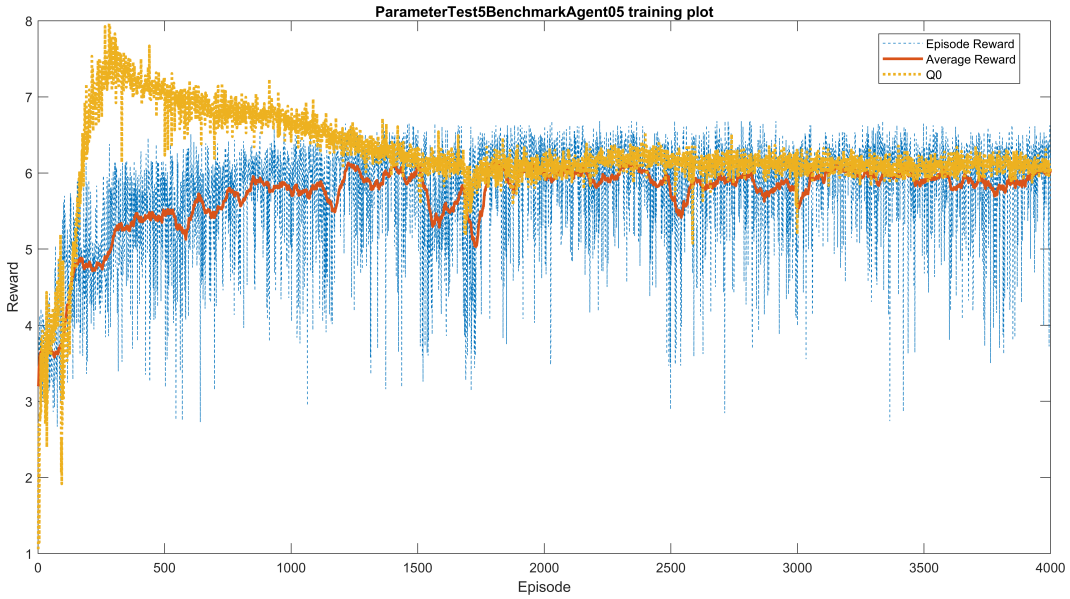
(b) Training graph of agent 7 from test 4



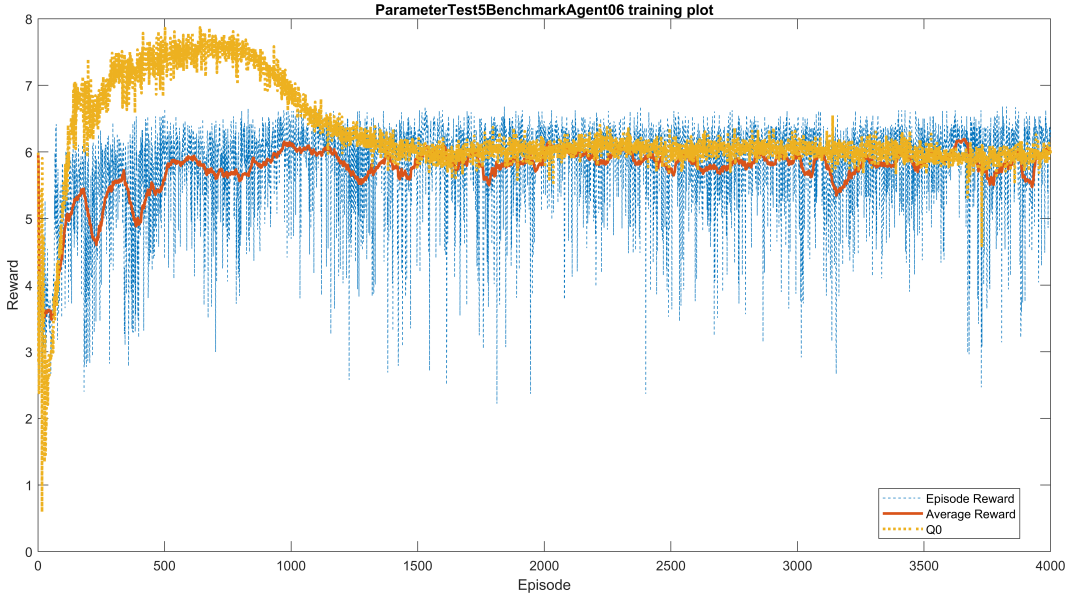
(c) Training graph of agent 8 from test 4

Figure 49: Comparison of training behavior of bad performers

Again the learning behavior is checked, for agent 5 and 6 in Figure 50. Here, the agents have a learning rate of 1e-3, and it is concluded that this learning rate is to big as the agent seems to heavily change the neural network thus they do seem to learn an optimal policy, while not consistently gaining the same rewards and the Q0 also being unstable. Hence, learning rate of 1e-3 is also removed for the next iteration.



(a) Training graph of agent 5 from test 4



(b) Training graph of agent 6 from test 4

Figure 50: Comparison of training behavior of bad performers

I.5 Fifth iteration

In this final test, 15 agents were trained. In Figure 51 these agents have been compared on the performance metrics.

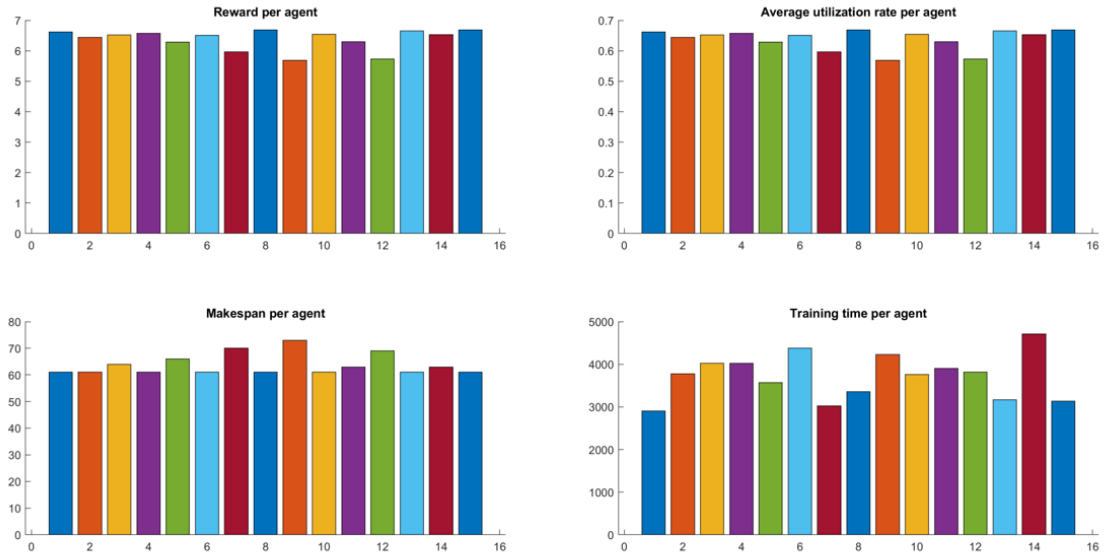
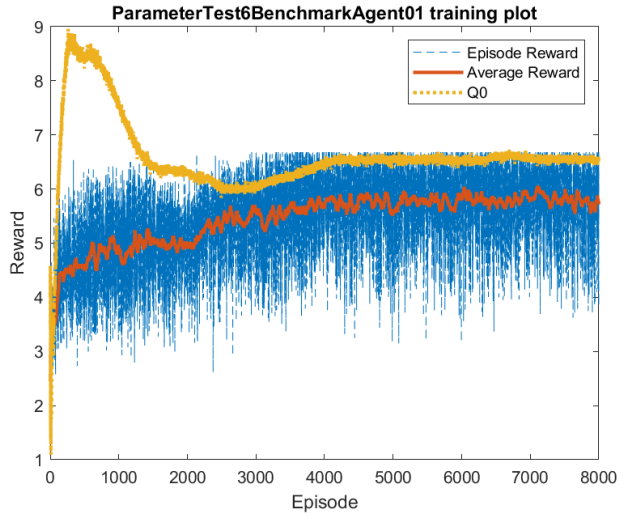
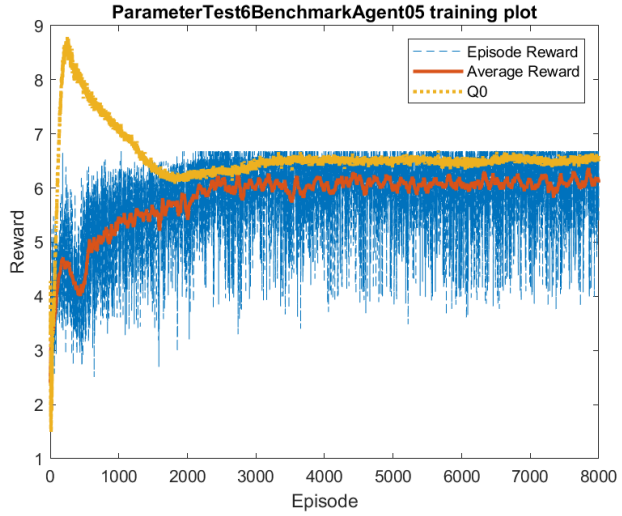


Figure 51: Comparison of agents on performance metrics of test 5

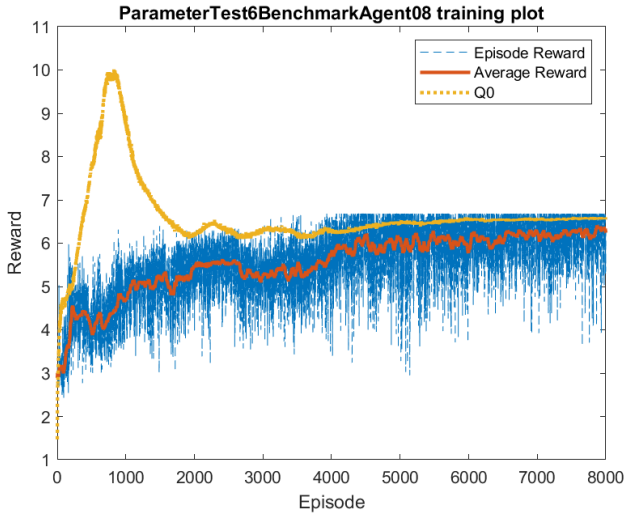
From these agents in Figure 51, agents 1, 5, 8, 10, 13 and 15 are the ones with the best performances. In Figure 52 and 53 the learning behavior is depicted in their training graphs.
From these it is found that agent 13 shows the best learning behavior, as the Q0-values are about equal to the average received reward, showing that it has learned an optimal policy. Hence, the values of agent 13 are used for the future experiments, given in Table 53.



(a) Training graph of agent 1 from test 5

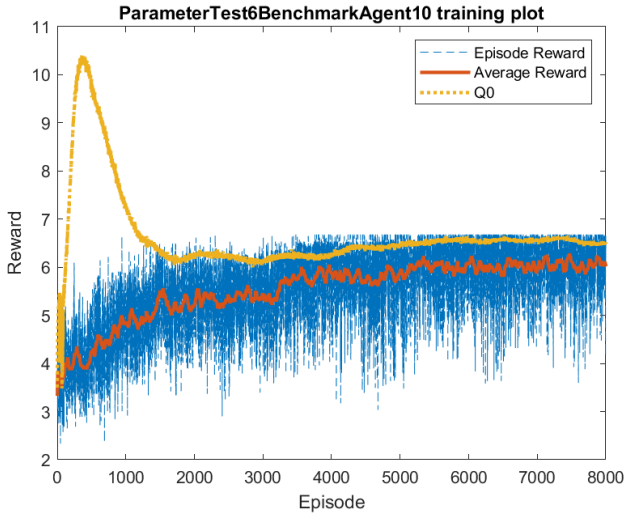


(b) Training graph of agent 5 from test 5

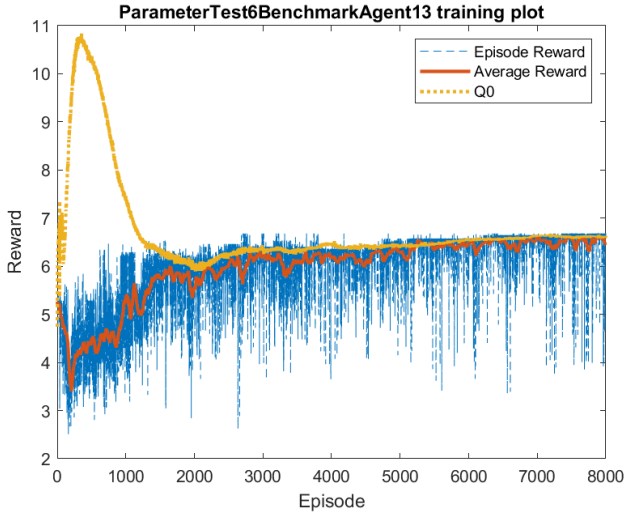


(c) Training graph of agent 8 from test 5

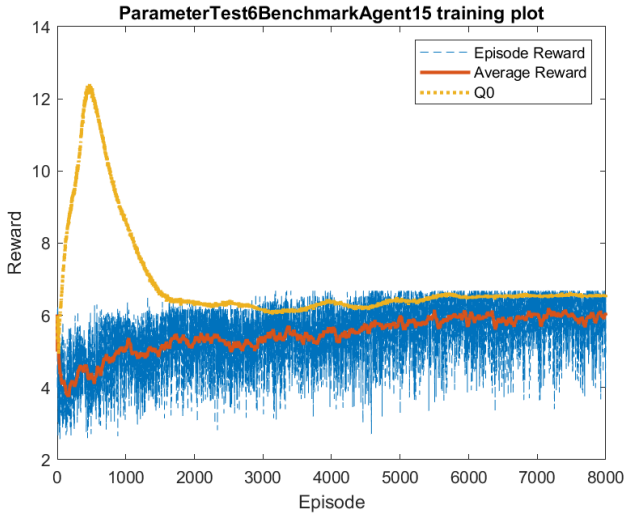
Figure 52: Comparison of training behavior of best performing agents



(a) Training graph of agent 10 from test 5



(b) Training graph of agent 13 from test 5



(c) Training graph of agent 15 from test 5

Figure 53: Comparison of training behavior of best performing agents

Agent	LR	DF	ϵ	ϵ_d	Neurons	Hidden Layers	CT (s)
1	1E-4	1	0.2	1E-6	128	4	2906.60
2	1E-4	1	0.01	1E-10	128	5	3775.23
3	1E-5	1	1E-01	1E-6	128	5	4021.75
4	1E-4	1	1E-02	1E-6	128	5	4017.49
5	1E-4	1	1E-01	1E-6	128	5	3565.92
6	1E-4	1	3E-01	1E-2	128	6	4378.41
7	1E-4	1	3E-01	1E-10	128	5	3021.26
8	1E-5	1	0.1	1E-6	128	5	3359.47
9	1E-5	1	0.2	1E-10	128	6	4228.14
10	1E-5	1	1.00E-01	1E-10	128	5	3757.96
11	1E-5	1	0.3	1E-2	128	5	3900.31
12	1E-5	1	0.3	1E-10	128	5	3817.84
13	1E-5	1	0.01	1E-2	128	4	3168.59
14	1E-5	1	0.01	1E-10	128	6	4712.36
15	1E-5	1	0.1	1E-10	128	4	3132.68

Table 53: Values of agents in fifth hyperparameter tuning test

To compare the chosen agent, in Figure 54 the Gantt chart of the results of agent 13 are given while in Table 54 the statistical results of the agent are given. When comparing to the benchmark agent, it is found that the learning behavior is improved, as the agent shows that the average gained rewards becomes more over time as well as statistically showing that it can find a result more consistent. Hence it is concluded that the found hyperparameters improve upon the benchmark agent.

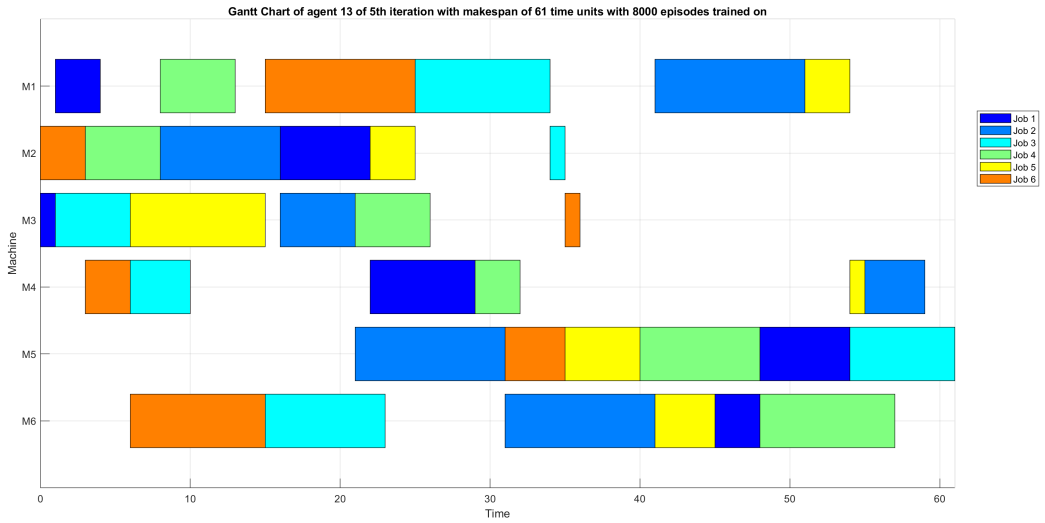


Figure 54: Gantt chart of chosen agent

	R	C	U_{avg}
UB	6.65	61	0.665
LB	6.65	61	0.665
mean	6.65	61	0.665
var	3.22E-30	0	1.26E-32
StD	1.79E-15	0	1.12E-16

Table 54: Statistical values of agent 13 from first iteration

J Results of training per standardized problem

With the defined values of the hyperparameters from the previous hyperparameter tuning, the proposed method was tested, where the inputs for the neural networks are equal to the size of the job shop problem, being $M + N$, where M is the number of machines and N the number of jobs. In Figure 55 together with Table 55, it is shown that the results of these tests show that when increasing the number of jobs and machines, the computational time greatly increases with them.

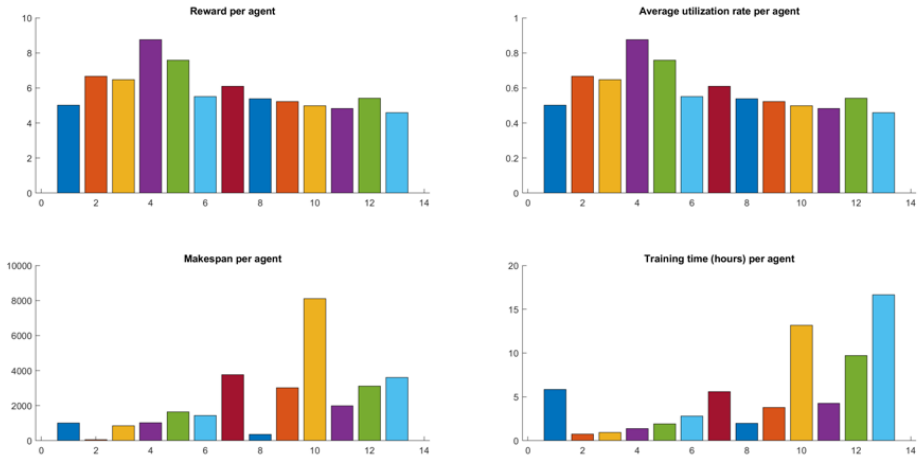


Figure 55: Comparison of agents on benchmark tests

agent	instance	reward	makespan	machines	jobs
1	abz8	5.014	1010	15	20
2	ft06	6.661	61	6	6
3	la04	6.473	853	5	10
4	la09	8.761	1021	5	15
5	la15	7.582	1636	5	20
6	la25	5.511	1427	10	15
7	la35	6.099	3756	10	30
8	svw01	5.224	3013	10	20
9	svw15	4.978	8104	10	50
10	ta01	4.827	1990	15	15
11	ta31	5.418	3122	15	30
12	ta41	4.595	3603	20	30

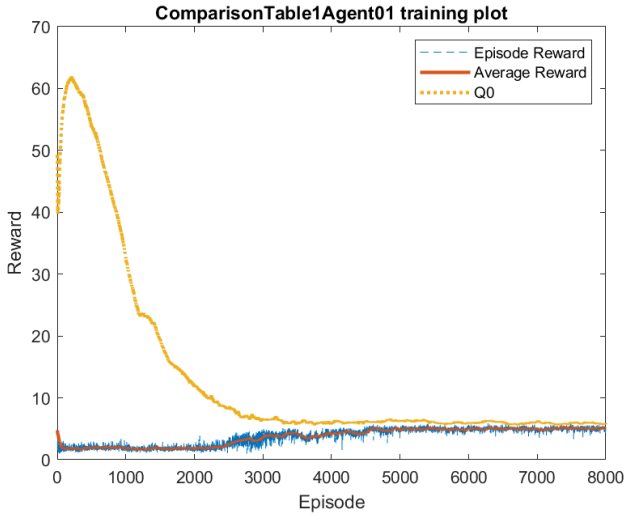
Table 55: Agents test on benchmark problems

Comparing these results with the results found by Liu [81] depicted in Table 56, it is found the bigger the problem becomes, the less effective the created agents are, comparing to both the dispatching rules as well as the DRL methods proposed in terms of makespan.

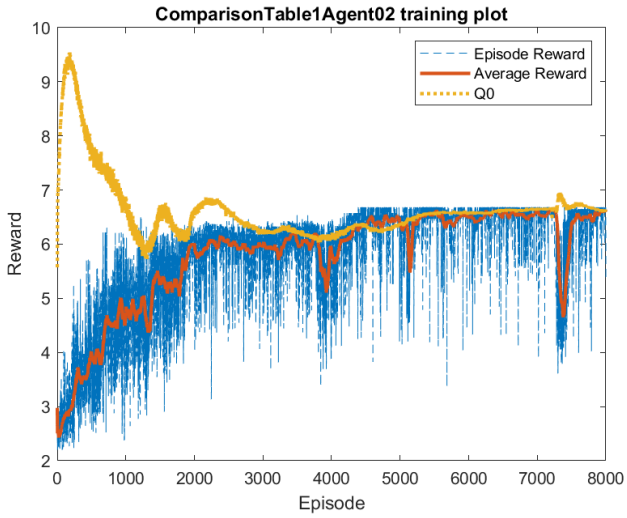
		Dispatching rules									DRL methods			Ours
Instance	Size	SPT	LPT	FIFO	LIFO	SWT	LWT	MWKR	LWKR	GA	D3QPN	L2D	P3OR	DQN
ft06	6 × 6	84	73	65	70	83	62	59	68	55	59	64	57	61
orb07	10 × 10	504	520	502	500	512	487	482	519	426	438	470	415	
la04	10 × 5	711	832	758	741	864	712	706	885	617	635	736	624	853
la09	15 × 5	1045	1183	997	1073	1135	1012	973	1149	954	978	1015	952	1021
la15	20 × 5	1339	1612	1282	1345	1587	1312	1258	1598	1128	1241	1295	1235	1636
la25	15 × 10	1297	1374	1283	1352	1471	1336	1172	1425	1160	1153	1204	1148	1427
la35	30 × 10	2133	2324	2004	2215	2368	2274	1962	2287	2019	1994	2085	1927	3756
abz8	20 × 15	929	949	879	938	957	936	810	992	744	778	861	736	1010
yn1	20 × 20	1196	1115	1123	1177	1214	1163	1045	1205	926	1053	1121	997	
swv01	20 × 10	1737	2145	1889	2123	2005	1923	1971	1838	1732	1712	1845	1645	3013
swv15	50 × 10	3501	4404	3603	3573	4133	4026	4905	3919	3422	3431	3516	3328	8104
ta01	15 × 15	1462	1701	1830	1627	1712	1523	1438	1737	1457	1405	1521	1412	1990
ta31	30 × 15	2335	2417	2436	2417	2754	2218	2143	2962	2237	2116	2231	2044	3122
ta41	30 × 20	2499	2925	2973	2760	2814	2609	2538	2976	2739	2475	2613	2387	3603
ta51	50 × 15	3856	3880	3717	3391	3702	3624	3567	3596	3250	3151	3224	3018	
ta61	50 × 20	3606	3989	4046	3870	3827	3568	3376	4073	3658	3365	3441	3256	
ta71	100 × 20	6232	7038	6704	6767	6735	6524	5938	6993	6524	5938	6993	5624	

Table 56: Table comparing proposed DQN with other methods from Liu [81]

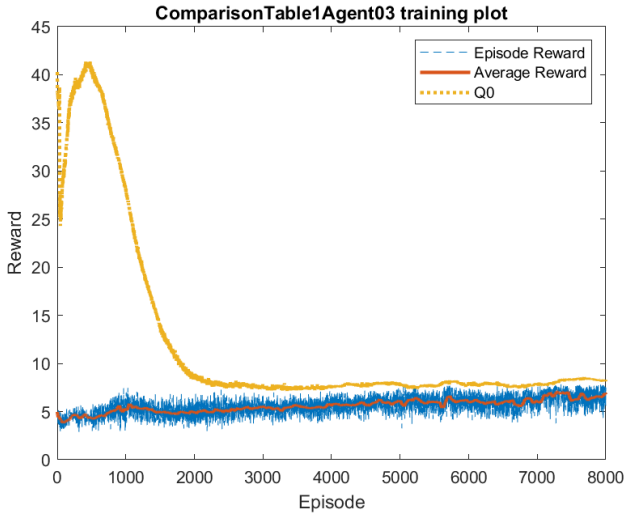
Looking at each training graph in Figures 56, 57, 58 and 59, it is also seen that the bigger the problem becomes, the less stable the training is. The Q0-value is highly overestimated when the problem becomes bigger. This is probably due to the fact that the change of the architecture of the neural network while maintaining the same hyperparameters. These hyperparameters do not accommodate for the change in input size, which makes them less compatible for these changes in architecture. It is important to note that for the figures in Figure **??** are not equally scaled to show each agents behavior individually.



(a) Agent 1: abz8 (15 jobs, 20 machines)

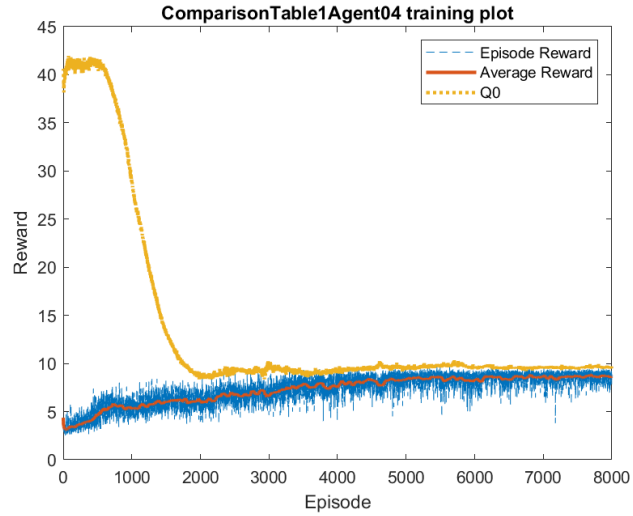


(b) Agent 2: ft06 (6 jobs, 6 machines)

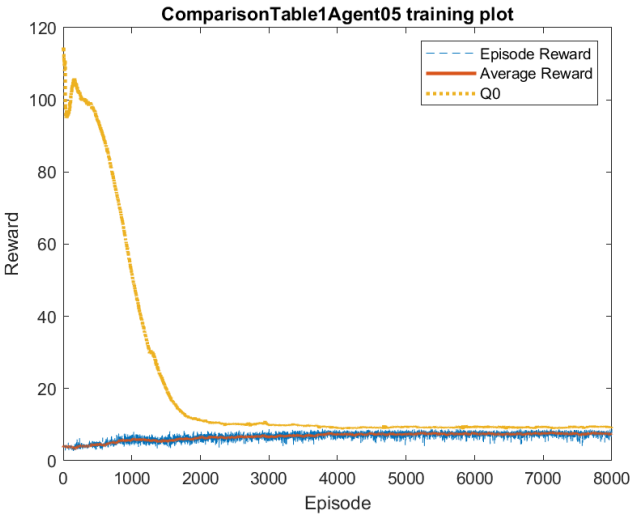


(c) Agent 3: la04 (5 jobs, 10 machines)

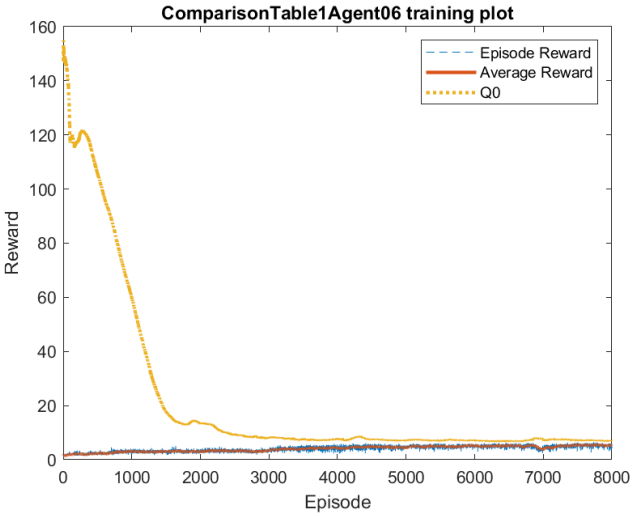
Figure 56: Comparison of training graphs of the benchmark trained agents



(a) Agent 4: la09 (5 jobs, 15 machines)

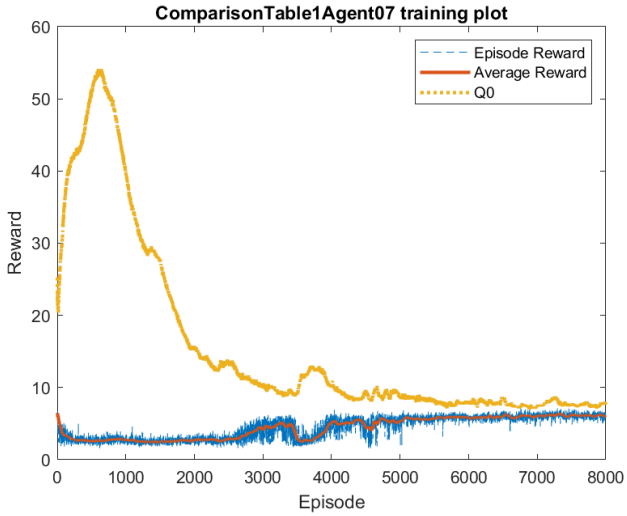


(b) Agent 5: la15 (5 jobs, 20 machines)

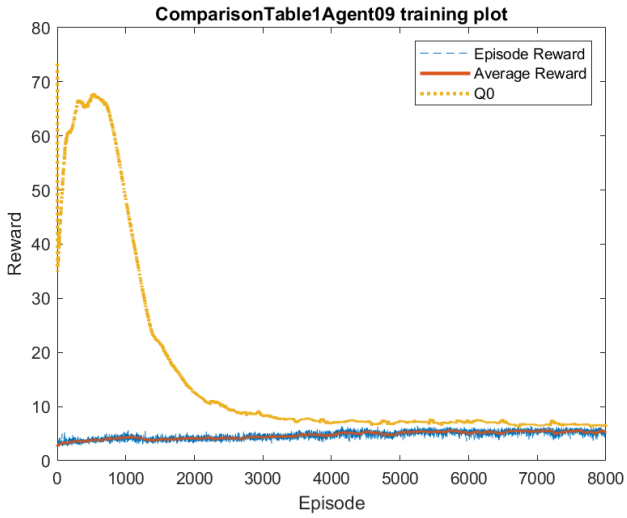


(c) Agent 6: la25 (10 jobs, 15 machines)

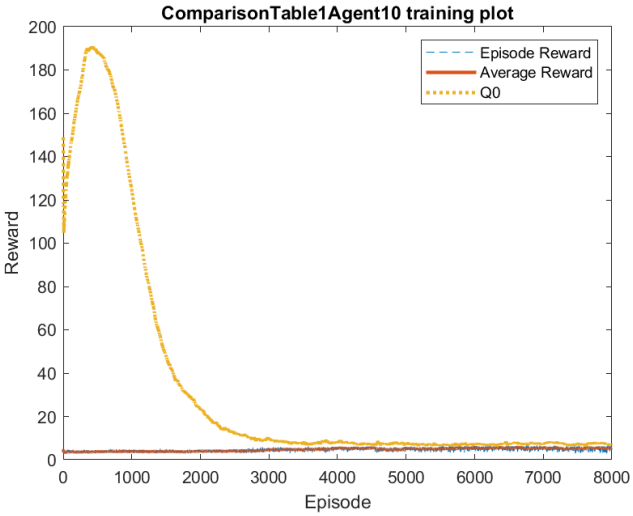
Figure 57: Comparison of training graphs of the benchmark trained agents



(a) Agent 7: la35 (15 jobs, 20 machines)

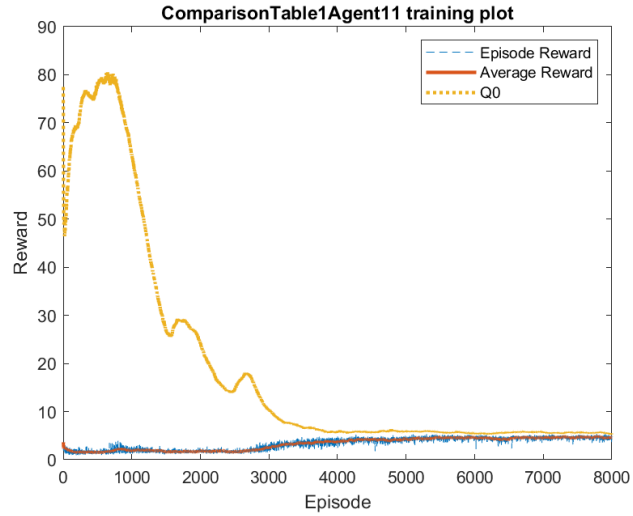


(b) Agent 8: svw01 (10 jobs, 20 machines)

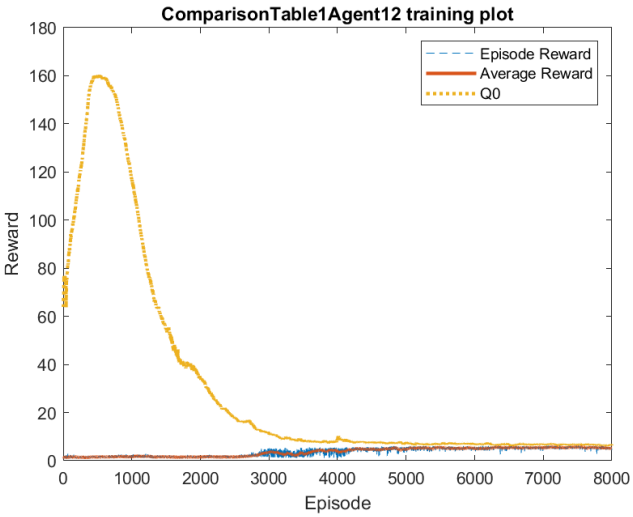


(c) Agent 9: swv15 (10 jobs, 50 machines)

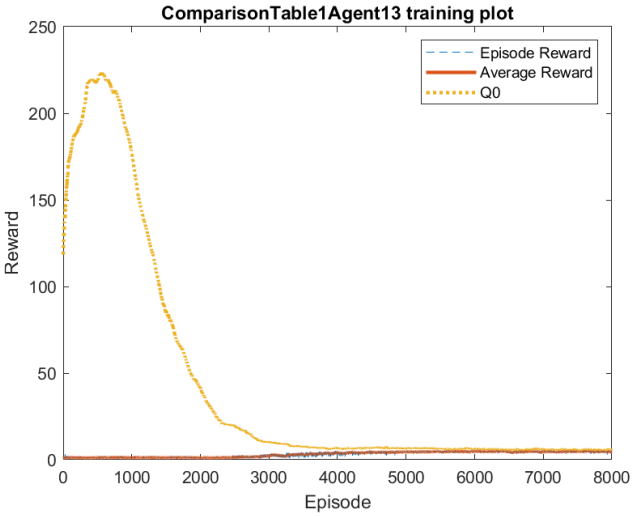
Figure 58: Comparison of training graphs of the benchmark trained agents



(a) Agent 10: ta01 (15 jobs, 15 machines))



(b) Agent 11: ta31 (15 jobs, 30 machines)



(c) Agent 12: ta41 (20 jobs, 30 machines)

Figure 59: Comparison of training graphs of the benchmark trained agents

K Results of second hyperparameter tuning: static job shop scheduling

To ensure the creation of the best performing DQN agent, random hyperparameter tuning is used again. The learning rate, discount factor, epsilon, epsilon decay and minimum epsilon for each trained agent are varied randomly. The reward scaling has been set to 10 for these tests. Additionally, the neural network's architecture is modified by adjusting the number of hidden layers and neurons corresponding to those hidden layers.

Based on earlier research, it can be concluded that a lot of the learning process of the agents can be assessed early in the training sequence. With this knowledge, initial testing is done with a wide range of variables with 500 episodes, still using problem ft06, to assess the effectiveness of the combinations early on in the learning process, while maintaining low computational time.

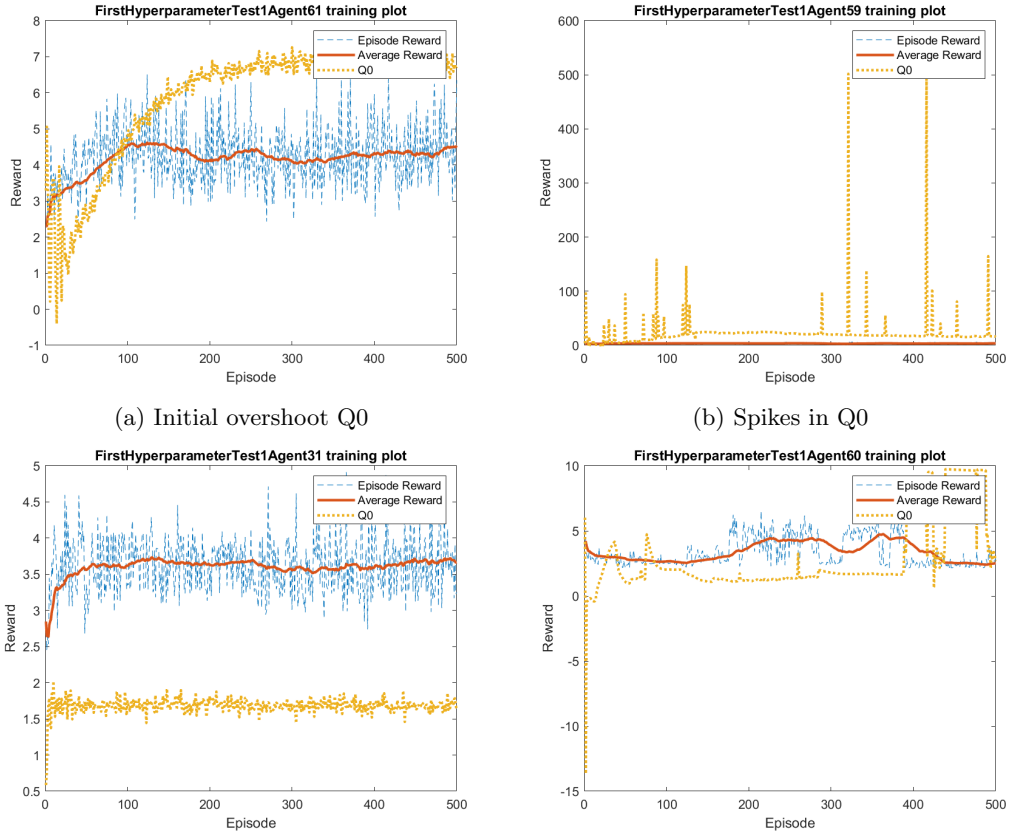
K.1 First iteration: initialization testing

The agents for the first test are trained on 500 episodes, assessing their training graphs to determine which hyperparameters show wanted learning behavior. In Table 57 the range of the hyperparameters are given for this test.

LR	DF	ϵ	ϵ_m	ϵ_d	Neurons	Hidden layers
1e-1 1e-4 1e-7	0.1, 0.5, 0.99	1e-7 1e-4 1	1e-13 1e-7 0.01	1e-10 1e-6 1e-2	64 512 2048	1 4 7

Table 57: Range of values for first initialization hyperparameter test

75 different agents have been trained, and the learning behavior is categorized in 4 different categories, based on the training graphs. These categories are initial overshoot, spikes, almost no change in Q0 (flat) and finally others, depicting behavior which can not really be categorized. In Figure 60 the different categories are depicted in different figures, while in Table 58 the different agents per group are given, and in Table 59 the specific settings per agent are given.



(a) Initial overshoot Q0

(b) Spikes in Q0

(c) no change in Q0

(d) Others category

Figure 60: Representation of different categories

Category	Agents
Overshoot	8 16 18 21 22 26 35 43 44 46 61 73 74 75
spikes	3 9 11 14 15 17 27 38 40 42 45 47 48 53 55 56 58 59 62 66 69
Flat	2 4 7 10 19 23 30 31 32 33 36 37 39 51 70
others	1 5 6 12 13 20 24 25 28 29 34 41 49 50 52 54 57 60 63 64 65 67 68 71 72

Table 58: Overview of different categories and agents

From the parameters of the different categories, the spikes category is found to all have a learning rate of 0.1, which shows that the heavy changes of weights and biases results in unstable behavior in terms of Q0 estimations. The overshoot category shows consistency in the learning rate all being 1e-4, while also having a discount factor of 0.5 or 0.99, and also most of them having a epsilon of 1. From earlier results, it is found that this overshoot will most likely result in wanted learning behavior. Hence, the hyperparameters need to be tuned more into the direction of these agents hyperparameters.

Finally, the computational time is assessed in Figure 61, where it shows that the higher number of neurons show longer computational times, as expected, thus the reduction of layers and neurons is needed for the next hyperparameter tuning.

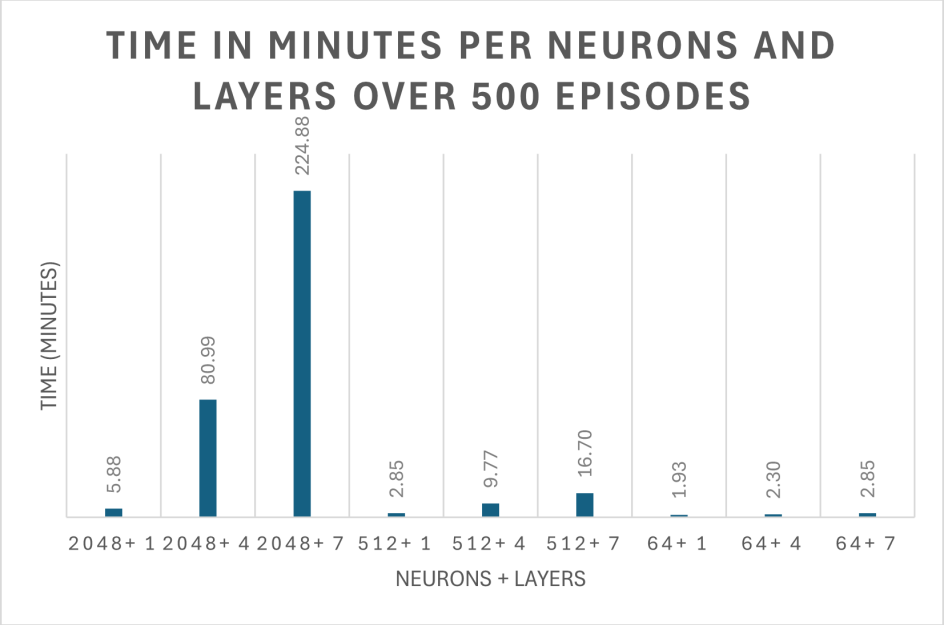


Figure 61: Comparison of computational time for different combinations of layers and neurons

Agent	**LR**	**DF**	ϵ	ϵ_d	ϵ_m	**Neurons**	**HL**	**CT**
1	1.0E-07	9.9E-01	1.0E-07	1.0E-10	1.0E-13	6.4E+01	4.0E+00	1.4E+02
2	1.0E-07	1.0E-04	1.0E-04	1.0E-06	1.0E-01	5.1E+02	7.0E+00	7.3E+02
3	1.0E-01	1.0E-04	1.0E+00	1.0E-10	1.0E-13	2.0E+03	7.0E+00	8.5E+03
4	1.0E-04	1.0E-04	1.0E-04	1.0E-06	1.0E-07	2.0E+03	1.0E+00	3.7E+02
5	1.0E-07	5.0E-01	1.0E-04	1.0E-02	1.0E-01	2.0E+03	1.0E+00	2.7E+02
6	1.0E-07	1.0E-04	1.0E+00	1.0E-10	1.0E-13	6.4E+01	4.0E+00	1.2E+02
7	1.0E-07	9.9E-01	1.0E+00	1.0E-06	1.0E-07	6.4E+01	1.0E+00	1.1E+02
8	1.0E-04	5.0E-01	1.0E+00	1.0E-06	1.0E-07	6.4E+01	7.0E+00	1.6E+02
9	1.0E-01	1.0E-04	1.0E+00	1.0E-10	1.0E-07	2.0E+03	4.0E+00	4.0E+03
10	1.0E-07	9.9E-01	1.0E-04	1.0E-10	1.0E-07	5.1E+02	1.0E+00	1.4E+02
11	1.0E-01	5.0E-01	1.0E-04	1.0E-02	1.0E-13	5.1E+02	7.0E+00	9.5E+02
12	1.0E-04	5.0E-01	1.0E-07	1.0E-02	1.0E-13	6.4E+01	7.0E+00	2.0E+02
13	1.0E-07	1.0E-04	1.0E-04	1.0E-06	1.0E-07	5.1E+02	4.0E+00	4.8E+02
14	1.0E-01	5.0E-01	1.0E-07	1.0E-06	1.0E-07	5.1E+02	1.0E+00	2.0E+02
15	1.0E-01	1.0E-04	1.0E-04	1.0E-02	1.0E-07	5.1E+02	7.0E+00	8.3E+02
16	1.0E-04	5.0E-01	1.0E+00	1.0E-10	1.0E-07	5.1E+02	7.0E+00	1.4E+03
17	1.0E-01	1.0E-04	1.0E+00	1.0E-10	1.0E-13	5.1E+02	7.0E+00	9.2E+02
18	1.0E-04	9.9E-01	1.0E+00	1.0E-02	1.0E-01	2.0E+03	4.0E+00	6.6E+03
19	1.0E-04	1.0E-04	1.0E-04	1.0E-06	1.0E-01	6.4E+01	4.0E+00	1.4E+02
20	1.0E-04	5.0E-01	1.0E-04	1.0E-02	1.0E-13	6.4E+01	1.0E+00	1.2E+02
21	1.0E-04	5.0E-01	1.0E-04	1.0E-02	1.0E-07	2.0E+03	7.0E+00	1.8E+04
22	1.0E-04	5.0E-01	1.0E+00	1.0E-06	1.0E-13	2.0E+03	7.0E+00	1.5E+04
23	1.0E-07	9.9E-01	1.0E-04	1.0E-10	1.0E-01	6.4E+01	1.0E+00	1.1E+02
24	1.0E-04	5.0E-01	1.0E-07	1.0E-10	1.0E-07	5.1E+02	1.0E+00	1.9E+02
25	1.0E-04	1.0E-04	1.0E+00	1.0E-10	1.0E-07	2.0E+03	1.0E+00	3.9E+02
26	1.0E-04	9.9E-01	1.0E-07	1.0E-06	1.0E-13	2.0E+03	7.0E+00	1.6E+04
27	1.0E-01	1.0E-04	1.0E-07	1.0E-02	1.0E-07	2.0E+03	4.0E+00	4.4E+03
28	1.0E-04	5.0E-01	1.0E-04	1.0E-10	1.0E-01	6.4E+01	7.0E+00	1.7E+02
29	1.0E-07	9.9E-01	1.0E-07	1.0E-06	1.0E-01	5.1E+02	4.0E+00	4.4E+02
30	1.0E-07	1.0E-04	1.0E+00	1.0E-10	1.0E-07	2.0E+03	4.0E+00	3.4E+03
31	1.0E-04	1.0E-04	1.0E-07	1.0E-06	1.0E-13	2.0E+03	7.0E+00	1.7E+04
32	1.0E-04	1.0E-04	1.0E-07	1.0E-10	1.0E-07	5.1E+02	4.0E+00	8.6E+02

33	1.0E-04	1.0E-04	1.0E-07	1.0E-02	1.0E-01	2.0E+03	1.0E+00	3.7E+02
34	1.0E-04	5.0E-01	1.0E-04	1.0E-06	1.0E-07	6.4E+01	1.0E+00	1.2E+02
35	1.0E-04	9.9E-01	1.0E+00	1.0E-06	1.0E-07	2.0E+03	1.0E+00	3.5E+02
36	1.0E-04	1.0E-04	1.0E-04	1.0E-10	1.0E-13	2.0E+03	7.0E+00	1.7E+04
37	1.0E-07	1.0E-04	1.0E+00	1.0E-06	1.0E-01	2.0E+03	4.0E+00	3.4E+03
38	1.0E-01	9.9E-01	1.0E-04	1.0E-06	1.0E-07	6.4E+01	7.0E+00	1.6E+02
39	1.0E-04	5.0E-01	1.0E-07	1.0E-06	1.0E-01	2.0E+03	1.0E+00	3.5E+02
40	1.0E-01	5.0E-01	1.0E-04	1.0E-06	1.0E-01	6.4E+01	7.0E+00	1.7E+02
41	1.0E-07	1.0E-04	1.0E-07	1.0E-10	1.0E-07	6.4E+01	4.0E+00	1.4E+02
42	1.0E-01	1.0E-04	1.0E-04	1.0E-02	1.0E-01	6.4E+01	7.0E+00	1.6E+02
43	1.0E-04	9.9E-01	1.0E-04	1.0E-10	1.0E-13	2.0E+03	7.0E+00	1.6E+04
44	1.0E-04	9.9E-01	1.0E-07	1.0E-10	1.0E-13	6.4E+01	4.0E+00	1.5E+02
45	1.0E-01	1.0E-04	1.0E-07	1.0E-02	1.0E-01	5.1E+02	4.0E+00	5.6E+02
46	1.0E-04	1.0E-04	1.0E+00	1.0E-06	1.0E-01	6.4E+01	7.0E+00	1.6E+02
47	1.0E-01	1.0E-04	1.0E-07	1.0E-10	1.0E-01	2.0E+03	4.0E+00	5.1E+03
48	1.0E-01	1.0E-04	1.0E-07	1.0E-02	1.0E-01	2.0E+03	7.0E+00	7.0E+03
49	1.0E-07	1.0E-04	1.0E-04	1.0E-06	1.0E-01	6.4E+01	4.0E+00	1.3E+02
50	1.0E-07	9.9E-01	1.0E-04	1.0E-10	1.0E-01	6.4E+01	7.0E+00	1.5E+02
51	1.0E-04	1.0E-04	1.0E-07	1.0E-10	1.0E-13	6.4E+01	1.0E+00	1.2E+02
52	1.0E-07	5.0E-01	1.0E-04	1.0E-06	1.0E-13	6.4E+01	1.0E+00	1.1E+02
53	1.0E-01	9.9E-01	1.0E-04	1.0E-10	1.0E-13	5.1E+02	1.0E+00	1.8E+02
54	1.0E-07	5.0E-01	1.0E-04	1.0E-10	1.0E-07	6.4E+01	1.0E+00	1.1E+02
55	1.0E-01	9.9E-01	1.0E+00	1.0E-06	1.0E-13	5.1E+02	1.0E+00	1.8E+02
56	1.0E-01	9.9E-01	1.0E-07	1.0E-06	1.0E-07	2.0E+03	7.0E+00	8.0E+03
57	1.0E-04	5.0E-01	1.0E+00	1.0E-06	1.0E-07	6.4E+01	4.0E+00	1.4E+02
58	1.0E-01	1.0E-04	1.0E-07	1.0E-02	1.0E-01	2.0E+03	1.0E+00	3.8E+02
59	1.0E-01	9.9E-01	1.0E-07	1.0E-06	1.0E-01	2.0E+03	1.0E+00	3.7E+02
60	1.0E-01	9.9E-01	1.0E+00	1.0E-02	1.0E-01	6.4E+01	4.0E+00	1.4E+02
61	1.0E-04	9.9E-01	1.0E-07	1.0E-06	1.0E-01	2.0E+03	1.0E+00	3.8E+02
62	1.0E-01	9.9E-01	1.0E-04	1.0E-02	1.0E-01	6.4E+01	7.0E+00	1.9E+02
63	1.0E-07	5.0E-01	1.0E+00	1.0E-10	1.0E-07	2.0E+03	1.0E+00	2.9E+02
64	1.0E-04	1.0E-04	1.0E+00	1.0E-10	1.0E-01	6.4E+01	1.0E+00	1.3E+02
65	1.0E-07	9.9E-01	1.0E-07	1.0E-02	1.0E-01	2.0E+03	4.0E+00	3.5E+03
66	1.0E-01	5.0E-01	1.0E-04	1.0E-10	1.0E-07	2.0E+03	7.0E+00	1.1E+04
67	1.0E-04	9.9E-01	1.0E+00	1.0E-10	1.0E-07	6.4E+01	1.0E+00	1.1E+02
68	1.0E-07	1.0E-04	1.0E-04	1.0E-02	1.0E-01	5.1E+02	1.0E+00	1.4E+02
69	1.0E-01	5.0E-01	1.0E-07	1.0E-02	1.0E-07	6.4E+01	7.0E+00	1.7E+02
70	1.0E-04	1.0E-04	1.0E-07	1.0E-10	1.0E-07	5.1E+02	1.0E+00	1.7E+02
71	1.0E-01	9.9E-01	1.0E-07	1.0E-10	1.0E-13	6.4E+01	4.0E+00	1.4E+02
72	1.0E-07	1.0E-04	1.0E-07	1.0E-02	1.0E-07	5.1E+02	7.0E+00	7.4E+02
73	1.0E-04	9.9E-01	1.0E-07	1.0E-10	1.0E-13	2.0E+03	4.0E+00	8.4E+03
74	1.0E-04	5.0E-01	1.0E+00	1.0E-06	1.0E-01	5.1E+02	7.0E+00	1.5E+03
75	1.0E-04	5.0E-01	1.0E+00	1.0E-02	1.0E-07	6.4E+01	7.0E+00	1.8E+02

Table 59: All settings for the created agents of test 1

K.2 Second iteration

A new set of variables will be set based on the previous results, shown in Table 60. The goal will be to assess the data based on 100 created agents, over 1000 episodes. The increase in episodes is due to the benchmark agent showing a change in behavior around 750 episodes as well as the importance of assessing how the initial overshoot changing after 500 episodes.

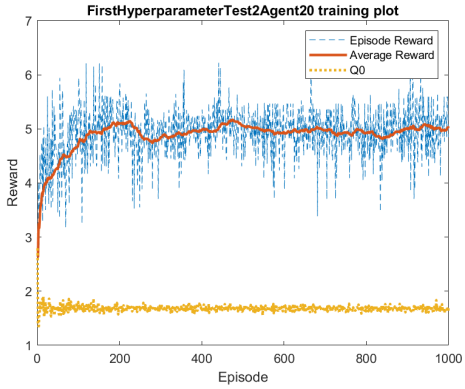
LR	DF	ϵ	ϵ_m	ϵ_d	Neurons	Hidden layers
1e-3 1e-4 1e5	1e-4 0.5 0.99	1e-7 1e-4 1	1e-13 1e-7 0.01	1e-10 1e-6 1e-2	64 128 256	1 to 5

Table 60: Possible values second initialization hyperparameter test

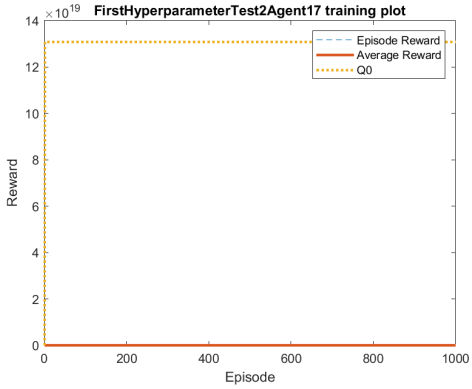
Due to a checking error, the highest learning rate became 1e5 instead of 1e-5. However, the results are still useful. From these 100 agents, 7 different categories of learning trajectories have been determined. These are a low and flat Q0, high and flat Q0, initial overshoot, no Q0, overshoot below gained rewards, spikes and undershoot (or instantly decreasing Q0). The different categories are shown in Figure 62 and the categorization of the agents is given in Table 61.

Category	**Agents**
Low & flat Q0	10 13 14 16 20 24 27 30 35 36 39 41 46 50 55 56 69 77 79 81 90 98
High & flat Q0	5 11 17 21 32 42 49 65 71 73 83 88
Overshoot	4 6 8 12 15 22 33 34 52 57 61 63 64 76 78 80 82 85 87 94 95 96
No Q0 found	40 48 51 62 84 86 92
Overshoot (2) below reward	2 25 29 59 67 68 70 93 97 99 100
Spikes	1 3 5 7 18 19 26 28 37 43 44 45 53 54 60 66 72 74 91
Undershoot/instant decrease	9 23 31 38 47 58 89

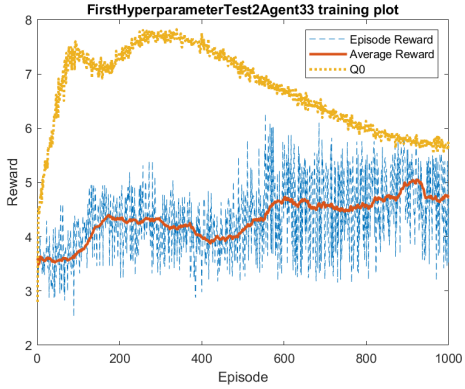
Table 61: Overview of different categories and agents in test 2



(a) Low & flat Q0



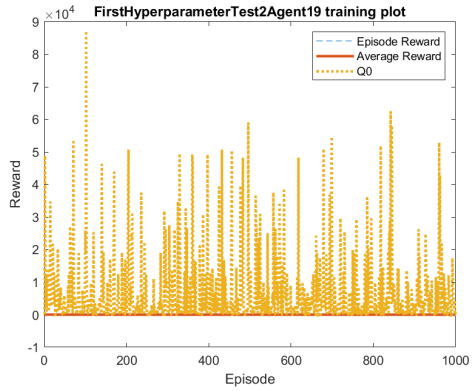
(b) High & flat Q0



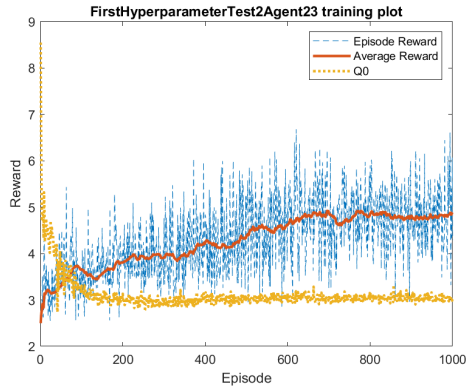
(c) Overshoot

(d) No Q0 found

(e) Overshoot 2



(f) Spikes



(g) Undershoot

Figure 62: Representation of different categories in test 2

Agent	**LR**	**DF**	ϵ	ϵ_d	ϵ_m	**Neurons**	**HL**	**CT**
1	1.0E+05	0.99	1.0E-07	1.0E-10	1.0E-13	64	3	268.89
2	1.0E-03	0.5	1	1.0E-02	1.0E-07	128	5	438.67
3	1.0E+05	1.0E-04	1.0E-04	1.0E-10	1.0E-07	256	5	448.88
4	1.0E-04	0.99	1	1.0E-10	1.0E-07	64	2	247.36
5	1.0E+05	0.5	1	1.0E-10	1.0E-13	256	2	338.28
6	1.0E-03	0.99	1.0E-04	1.0E-06	1.0E-13	256	4	619.20
7	1.0E+05	0.5	1.0E-04	1.0E-06	1.0E-13	64	1	271.71
8	1.0E-04	0.99	1	1.0E-06	1.0E-01	64	4	360.90
9	1.0E-03	0.5	1.0E-07	1.0E-06	1.0E-07	64	2	277.50
10	1.0E-03	1.0E-04	1.0E-07	1.0E-06	1.0E-07	128	4	395.32
11	1.0E+05	1.0E-04	1.0E-07	1.0E-06	1.0E-07	64	2	243.87
12	1.0E-03	0.99	1.0E-07	1.0E-10	1.0E-01	64	3	275.13
13	1.0E-04	0.5	1	1.0E-02	1.0E-01	64	3	276.71
14	1.0E-03	1.0E-04	1.0E-04	1.0E-06	1.0E-01	128	3	342.92
15	1.0E-03	0.99	1	1.0E-02	1.0E-01	256	2	399.20
16	1.0E-03	1.0E-04	1	1.0E-10	1.0E-07	128	1	250.94
17	1.0E+05	1.0E-04	1	1.0E-06	1.0E-13	128	2	251.86
18	1.0E+05	0.5	1.0E-04	1.0E-06	1.0E-13	256	1	259.80
19	1.0E+05	0.99	1.0E-04	1.0E-02	1.0E-13	256	3	357.24
20	1.0E-04	1.0E-04	1.0E-04	1.0E-10	1.0E-07	256	5	889.58
21	1.0E+05	0.99	1.0E-04	1.0E-10	1.0E-13	256	2	311.78
22	1.0E-03	0.99	1.0E-04	1.0E-02	1.0E-13	128	5	429.66
23	1.0E-04	0.5	1.0E-04	1.0E-10	1.0E-01	128	2	304.99
24	1.0E-04	1.0E-04	1.0E-04	1.0E-10	1.0E-07	128	2	314.39
25	1.0E-03	0.5	1	1.0E-02	1.0E-07	128	3	345.27
26	1.0E+05	0.99	1.0E-04	1.0E-06	1.0E-13	256	5	446.13
27	1.0E-04	1.0E-04	1	1.0E-02	1.0E-01	64	1	230.63
28	1.0E+05	1.0E-04	1.0E-04	1.0E-06	1.0E-13	64	3	256.04
29	1.0E-04	0.5	1.0E+00	1.0E-10	1.0E-07	64	4	280.12
30	1.0E-03	0.5	1.0E-04	1.0E-06	1.0E-01	128	1	262.55
31	1.0E-03	0.5	1.0E-04	1.0E-02	1.0E-07	64	3	277.88
32	1.0E+05	0.5	1.0E-04	1.0E-10	1.0E-13	128	2	265.72
33	1.0E-04	0.99	1.0E-07	1.0E-06	1.0E-13	256	5	918.14
34	1.0E-04	0.99	1.0E-07	1.0E-06	1.0E-13	256	2	478.71
35	1.0E-04	1.0E-04	1.0E-04	1.0E-10	1.0E-13	128	3	388.71
36	1.0E-04	1.0E-04	1.0E-04	1.0E-02	1.0E-13	256	3	594.42
37	1.0E+05	0.5	1.0E-07	1.0E-10	1.0E-13	64	3	256.06
38	1.0E-04	0.5	1.0E-04	1.0E-10	1.0E-13	128	4	422.87
39	1.0E-03	1.0E-04	1	1.0E-02	1.0E-01	128	3	336.96
40	1.0E+05	0.5	1	1.0E-06	1.0E-01	128	4	295.60
41	1.0E-04	0.5	1.0E-07	1.0E-06	1.0E-13	64	5	320.94
42	1.0E+05	0.5	1.0E-07	1.0E-06	1.0E-01	64	1	228.68
43	1.0E+05	1.0E-04	1.0E-04	1.0E-02	1.0E-07	256	4	389.49
44	1.0E+05	0.5	1.0E-07	1.0E-02	1.0E-01	128	5	330.92
45	1.0E+05	0.5	1	1.0E-02	1.0E-07	128	5	327.84
46	1.0E-04	1.0E-04	1.0E-07	1.0E-06	1.0E-07	64	3	279.91
47	1.0E-04	0.5	1.0E-07	1.0E-06	1.0E-07	128	3	362.59
48	1.0E+05	0.99	1	1.0E-06	1.0E-13	256	4	374.45
49	1.0E+05	1.0E-04	1.0E-04	1.0E-02	1.0E-07	256	1	255.21
50	1.0E-03	0.5	1.0E-07	1.0E-10	1.0E-07	256	4	569.80
51	1.0E+05	0.99	1.0E+00	1.0E-06	1.0E-07	256	5	415.46
52	1.0E-04	0.99	1.0E-07	1.0E-10	1.0E-07	64	1	241.66
53	1.0E+05	0.99	1.0E-07	1.0E-10	1.0E-07	256	5	431.50
54	1.0E+05	0.99	1.0E-04	1.0E-10	1.0E-13	256	5	440.89
55	1.0E-03	1.0E-04	1.0E-07	1.0E-10	1.0E-13	64	5	324.31

56	1.0E-03	0.5	1.0E-04	1.0E-06	1.0E-07	128	5	431.41
57	1.0E-03	0.99	1	1.0E-10	1.0E-13	256	5	642.59
58	1.0E-03	0.5	1.0E-07	1.0E-10	1.0E-13	128	3	344.66
59	1.0E-03	0.5	1	1.0E-10	1.0E-07	128	3	322.63
60	1.0E+05	0.5	1.0E-07	1.0E-10	1.0E-07	128	3	284.52
61	1.0E-04	0.99	1	1.0E-02	1.0E-13	128	2	303.47
62	1.0E+05	0.99	1	1.0E-06	1.0E-07	128	3	274.47
63	1.0E-03	0.99	1	1.0E-02	1.0E-13	64	2	255.34
64	1.0E-03	0.99	1	1.0E-10	1.0E-13	64	4	283.37
65	1.0E+05	0.5	1.0E-04	1.0E-06	1.0E-01	64	1	224.47
66	1.0E+05	0.99	1	1.0E-10	1.0E-07	128	1	229.44
67	1.0E-03	0.5	1	1.0E-10	1.0E-01	64	5	296.20
68	1.0E-03	0.5	1.0E-04	1.0E-02	1.0E-01	128	1	265.39
69	1.0E-03	1.0E-04	1.0E-07	1.0E-02	1.0E-07	256	4	598.76
70	1.0E-03	0.5	1.0E-07	1.0E-02	1.0E-07	128	3	389.15
71	1.0E+05	1.0E-04	1.0E-04	1.0E-06	1.0E-07	64	2	284.83
72	1.0E+05	0.5	1.0E-04	1.0E-02	1.0E-07	64	4	320.17
73	1.0E+05	0.99	1.0E-04	1.0E-10	1.0E-07	128	2	306.29
74	1.0E+05	1.0E-04	1.0E-04	1.0E-02	1.0E-01	64	5	326.76
75	1.0E+05	0.99	1.0E-07	1.0E-02	1.0E-01	128	3	287.73
76	1.0E-03	0.99	1.0E-07	1.0E-10	1.0E-07	128	1	268.00
77	1.0E-04	1.0E-04	1.0E-04	1.0E-02	1.0E-07	64	5	333.20
78	1.0E-03	0.99	1.0E-04	1.0E-06	1.0E-13	64	2	260.62
79	1.0E-03	1.0E-04	1.0E-07	1.0E-10	1.0E-01	256	5	687.39
80	1.0E-03	0.99	1	1.0E-02	1.0E-01	128	3	336.08
81	1.0E-04	1.0E-04	1.0E-07	1.0E-06	1.0E-07	256	1	293.95
82	1.0E-03	0.99	1.0E-07	1.0E-10	1.0E-13	128	5	431.61
83	1.0E+05	0.99	1.0E-04	1.0E-06	1.0E-07	256	2	304.53
84	1.0E+05	1.0E-04	1	1.0E-02	1.0E-01	128	4	305.69
85	1.0E-03	0.99	1.0E-04	1.0E-06	1.0E-13	64	5	320.58
86	1.0E+05	0.99	1	1.0E-10	1.0E-07	128	5	313.68
87	1.0E-03	0.99	1.0E-07	1.0E-02	1.0E-01	64	2	254.19
88	1.0E+05	0.99	1	1.0E-02	1.0E-13	256	2	304.71
89	1.0E-04	0.5	1.0E-07	1.0E-02	1.0E-07	128	4	414.69
90	1.0E-03	0.5	1.0E-04	1.0E-10	1.0E-13	256	3	482.77
91	1.0E+05	0.99	1.0E-04	1.0E-02	1.0E-01	256	4	388.31
92	1.0E+05	0.5	1	1.0E-06	1.0E-01	64	3	245.65
93	1.0E-04	0.5	1	1.0E-10	1.0E-01	128	3	335.43
94	1.0E-04	0.99	1.0E-04	1.0E-10	1.0E-13	256	2	483.30
95	1.0E-04	0.99	1.0E-04	1.0E-02	1.0E-01	64	2	254.03
96	1.0E-03	0.99	1.0E-07	1.0E-10	1.0E-01	64	5	315.10
97	1.0E-04	1.0E-04	1	1.0E-02	1.0E-13	128	2	312.03
98	1.0E-03	1.0E-04	1.0E-04	1.0E-10	1.0E-07	256	3	492.36
99	1.0E-04	1.0E-04	1.0E-07	1.0E-06	1.0E-01	256	1	288.41
100	1.0E-03	0.5	1.0E-07	1.0E-10	1.0E-13	64	4	301.22

Table 62: All settings for the agents created in test 2

From the figures it can be concluded that the overshoot figures overall show a greater reward gained in comparison to the other categories, ensuring that the focus should be on the overshoot figures. From these values and the figures, it is determined that low and flat Q0 as well as the overshoot below rewards category agents contains no discount factor of 0.99. The high and flat Q0 as well as the no Q0 categories all have the accidental learning rate of 1e5. The undershoot category all have a discount factor of 0.5. Finally, the overshoot agents all have a discount factor of 0.99. Hence, it is determined that, with the knowledge of focusing on overshoot still, the discount factor should be higher. No other conclusions are made from these tests.

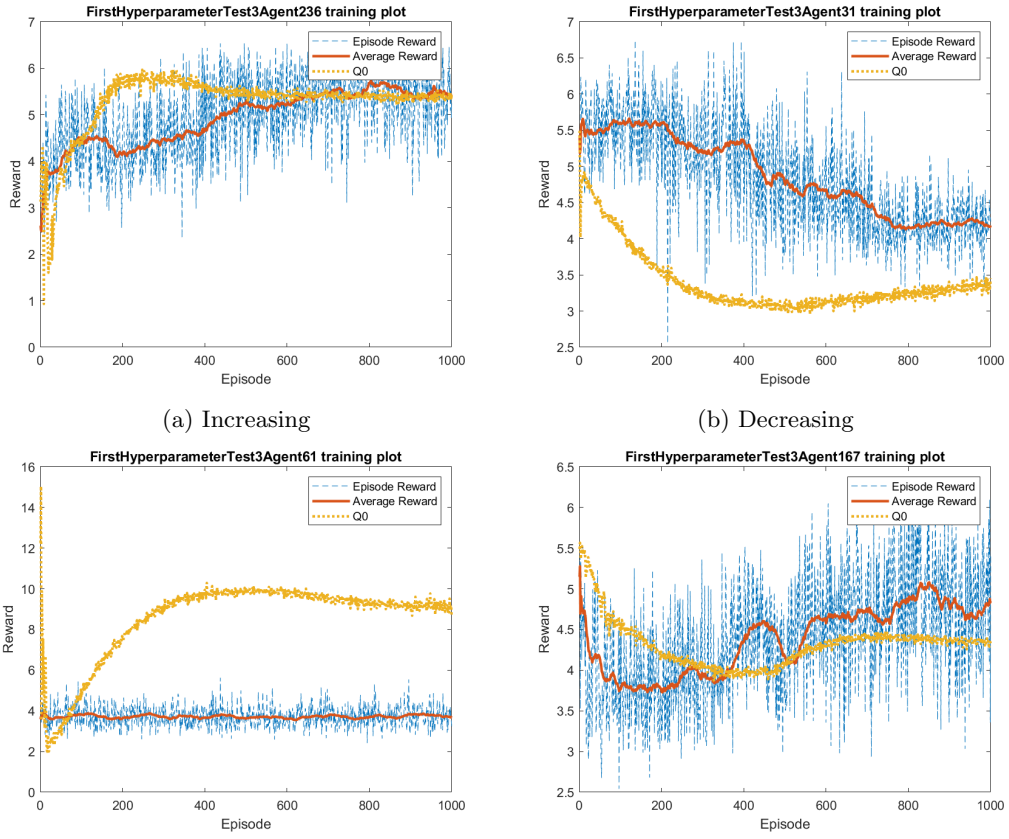
K.3 Third iteration

From the previous test, new settings have been defined in Table 63. Due to the shorter time these tests take and the useful results, the number of agents will also be increased to 250 agents.

LR	DF	ϵ	ϵ_m	ϵ_d	Neurons	Hidden layers
1e-3 1e-4 1e-5	0.75, 0.875, 0.99	1e-7 1e-4 1	1e-13 1e-7 0.01	1e-10 1e-6 1e-2	64 128 256	1 to 5

Table 63: Range of values for second initialization hyperparameter test

In Table 66, the settings for these 250 agents are given. Based on the resulting graphs from the training, being stable learning processes, in combination with the goal get a reward as high as possible, the focus is on categorizing the behavior based on the average received rewards. Hence, categories created are an increasing average reward, decreasing average reward and no increase, or flat, average reward. Next to these three, graphs with higher fluctuations are put into a separate category, called the others category. These are described in Figure 63.



(a) Increasing

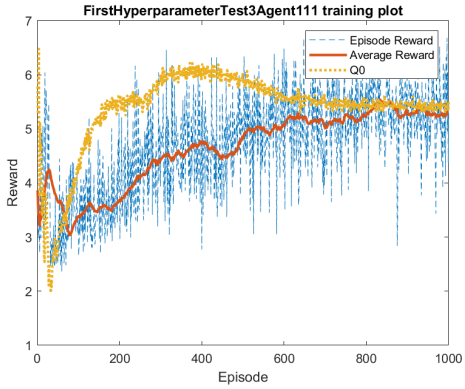
(b) Decreasing

(c) Flat

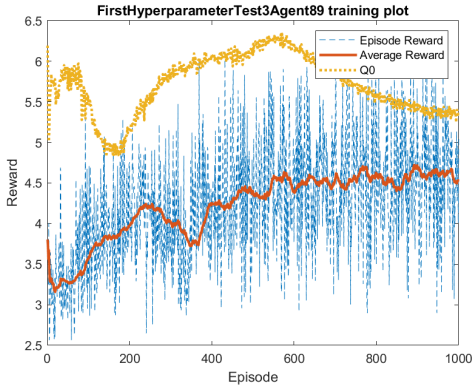
(d) Others

Figure 63: Representation of different categories in test 3

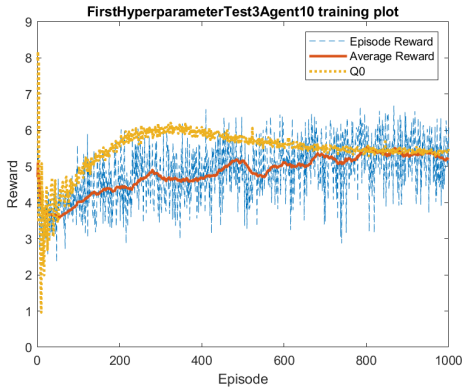
Also, due to the high number of graphs that have an increase in average reward, subcategories are created. The first subcategory is a the volatility in in the gained reward, where the wanted behavior is having a high volatility, i.e. exploration, at the start of the learning process, while being lower at the end. Secondly, a subcategory is defined to be based on the stabilized learning, based on Q0 and the average received reward. The differences are shown in Figure 64.



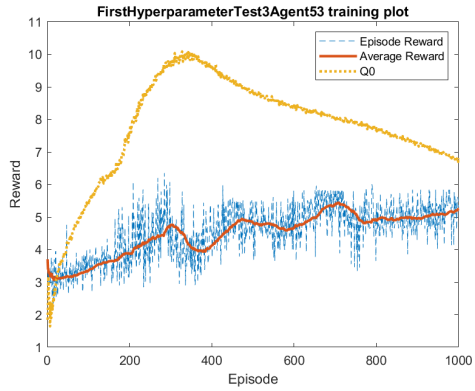
(a) high & stable



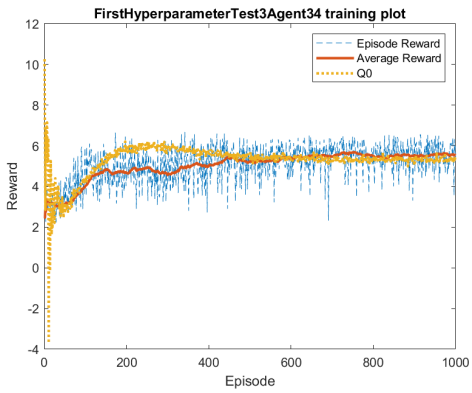
(b) High & unstable



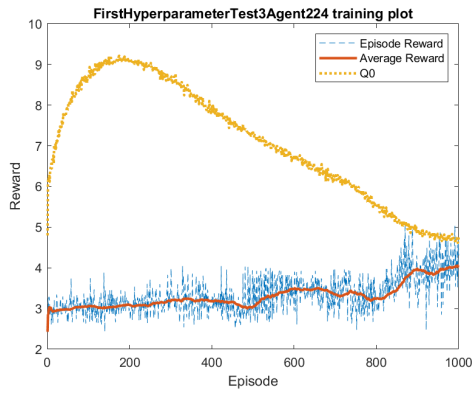
(c) Medium & stable



(d) Medium & unstable



(e) Low & stable



(f) Low & unstable

Figure 64: Representation of subcategories for increasing reward category

Category	Agents
Increasing reward	1 7 9 10 12 14 18 20 24 25 27 28 33 34 37 38 40 41 42 44 45 46 48 51 52 53 57 58 59 62 65 67 68 69 72 73 75 76 80 81 83 84 88 89 90 91 93 94 95 97 101 102 103 105 107 110 111 115 116 117 118 121 122 125 126 128 130 131 133 134 135 139 140 142 145 149 150 151 152 154 155 156 157 158 161 162 163 164 165 166 169 170 172 173 174 179 180 181 182 184 187 190 192 194 197 200 204 205 209 210 212 215 218 221 223 224 226 227 228 229 230 231 232 233 235 236 241 242 243 245 247 250
Decreasing reward	23 31 79 123 136
Flat	2 3 6 8 11 13 15 17 19 21 26 29 30 32 36 39 49 50 54 55 61 66 70 71 74 78 85 86 87 92 96 98 100 104 106 109 113 114 127 129 137 138 141 144 146 148 159 160 168 171 177 178 183 185 188 189 191 193 199 201 202 203 207 208 211 213 214 216 220 222 225 237 238 239 244 246 248 249
Others	4 5 16 22 35 43 47 56 60 63 77 82 99 108 112 119 120 124 132 143 147 153 167 175 176 186 195 196 198 206 217 219 234 240

Table 64: Overview of different categories and agents in test 3

The agents for the subcategories are defined in Table 65.

Category	Agents
High & stable	12 14 25 33 37 72 73 75 88 111 115 133 155 163 184 215 236 241 250
High & unstable	28 40 64 89 101 131 145 161 192 197 245
Medium & stable	9 10 27 38 45 51 52 62 65 67 68 69 76 81 83 84 91 93 94 102 105 107 116 118 126 134 135 150 151 154 156 157 162 170 172 173 180 187 194 204 209 223 226 230 232 233 235 242
Medium & unstable	7 20 24 42 44 46 53 57 58 80 95 103 110 117 122 125 128 149 164 165 166 169 179 182 205 212 218 221 228 229 231
Low & stable	18 34 41 48 59 97 121 130 174 181 200 247
Low & unstable	1 90 139 140 142 152 158 190 210 224 227 243

Table 65: Overview of different categories and agents in test 3

It is concluded from this test that the agents with a single hidden layer do not perform up to standard, as they show unwanted behavior such as slow increase in reward as well as receiving low rewards. Also, due to the fact of keeping exploration high at the early stages, the epsilon should be increased, should always be higher than the minimum epsilon, and at the end the minimum epsilon should be reached. Hence, minimum epsilon should be decreased for it's upper limit. Based on the epsilon, new values for epsilon decay are calculated to ensure the epsilon is not lowered too fast or too slow, in just a couple of episodes nearing 0 or always staying high around its own value. With these new values, better results are expected.

Agent	LR	DF	ϵ	ϵ_d	ϵ_m	Neurons	HL	CT
1	1.0E-05	0.99	1.0E-07	1.0E-10	1.0E-13	64	3	271.21
2	1.0E-04	0.75	1	1.0E-06	1.0E-13	128	2	272.05
3	1.0E-05	0.99	1.0E-07	1.0E-02	1.0E-13	64	5	280.23
4	1.0E-05	0.875	1.0E-04	1.0E-10	1.0E-13	256	5	861.62

5	1.0E-03	0.875	1.0E-07	1.0E-06	1.0E-13	64	4	293.39
6	1.0E-03	0.75	1.0E-07	1.0E-10	1.0E-13	64	3	286.62
7	1.0E-05	0.875	1.0E-07	1.0E-10	1.0E-01	256	2	419.34
8	1.0E-05	0.99	1	1.0E-10	1.0E-07	128	3	341.50
9	1.0E-04	0.99	1.0E-07	1.0E-06	1.0E-13	128	5	464.20
10	1.0E-04	0.875	1.0E-07	1.0E-06	1.0E-01	256	2	426.49
11	1.0E-04	0.99	1.0E-07	1.0E-06	1.0E-07	128	1	257.95
12	1.0E-05	0.875	1	1.0E-02	1.0E-01	64	4	297.95
13	1.0E-05	0.875	1	1.0E-10	1.0E-07	256	1	262.72
14	1.0E-05	0.875	1.0E-04	1.0E-02	1.0E-01	256	3	512.23
15	1.0E-05	0.875	1.0E-07	1.0E-06	1.0E-13	256	3	587.09
16	1.0E-03	0.75	1.0E-04	1.0E-02	1.0E-07	256	2	398.53
17	1.0E-05	0.99	1	1.0E-10	1.0E-01	256	4	703.72
18	1.0E-03	0.75	1	1.0E-02	1.0E-13	64	1	252.23
19	1.0E-05	0.99	1.0E-04	1.0E-06	1.0E-07	256	1	293.05
20	1.0E-04	0.75	1	1.0E-02	1.0E-01	128	1	251.87
21	1.0E-05	0.875	1.0E-04	1.0E-02	1.0E-13	128	1	254.04
22	1.0E-03	0.75	1.0E-07	1.0E-02	1.0E-13	128	3	337.55
23	1.0E-05	0.75	1.0E-04	1.0E-06	1.0E-13	64	2	253.31
24	1.0E-04	0.875	1.0E-07	1.0E-02	1.0E-13	256	4	797.21
25	1.0E-04	0.875	1	1.0E-02	1.0E-13	128	2	305.38
26	1.0E-05	0.75	1	1.0E-06	1.0E-01	128	4	372.94
27	1.0E-03	0.875	1.0E-04	1.0E-10	1.0E-01	256	3	471.35
28	1.0E-05	0.875	1.0E-07	1.0E-06	1.0E-01	128	4	383.49
29	1.0E-03	0.875	1.0E-07	1.0E-06	1.0E-13	64	4	296.11
30	1.0E-04	0.875	1	1.0E-10	1.0E-01	256	4	633.54
31	1.0E-05	0.75	1.0E-07	1.0E-10	1.0E-13	256	1	304.25
32	1.0E-04	0.75	1	1.0E-06	1.0E-13	256	1	277.23
33	1.0E-04	0.875	1.0E-07	1.0E-02	1.0E-07	64	4	310.58
34	1.0E-03	0.875	1.0E-07	1.0E-06	1.0E-01	128	1	262.53
35	1.0E-04	0.99	1.0E-04	1.0E-06	1.0E-13	64	4	314.04
36	1.0E-04	0.875	1.0E-04	1.0E-10	1.0E-07	64	3	279.28
37	1.0E-05	0.875	1.0E-07	1.0E-02	1.0E-13	64	5	327.09
38	1.0E-05	0.75	1.0E-04	1.0E-10	1.0E-13	256	2	484.78
39	1.0E-03	0.875	1	1.0E-06	1.0E-07	128	1	250.67
40	1.0E-04	0.99	1.0E-07	1.0E-02	1.0E-01	256	1	289.37
41	1.0E-05	0.75	1.0E-07	1.0E-10	1.0E-07	256	2	472.32
42	1.0E-03	0.99	1.0E-04	1.0E-02	1.0E-01	256	2	397.73
43	1.0E-05	0.75	1.0E-07	1.0E-06	1.0E-13	64	2	294.83
44	1.0E-04	0.875	1	1.0E-02	1.0E-07	256	1	334.71
45	1.0E-04	0.75	1	1.0E-02	1.0E-07	256	3	622.65
46	1.0E-05	0.875	1	1.0E-02	1.0E-01	256	5	910.81
47	1.0E-03	0.99	1.0E-04	1.0E-06	1.0E-07	64	4	301.57
48	1.0E-05	0.875	1.0E-07	1.0E-10	1.0E-13	128	3	359.64
49	1.0E-05	0.99	1.0E-07	1.0E-06	1.0E-01	128	1	246.41
50	1.0E-05	0.75	1.0E-04	1.0E-10	1.0E-13	128	4	387.34
51	1.0E-03	0.875	1.0E-07	1.0E-10	1.0E-01	256	4	573.83
52	1.0E-03	0.75	1.0E-07	1.0E-06	1.0E-07	64	5	327.02
53	1.0E-05	0.99	1	1.0E-02	1.0E-07	256	4	656.49
54	1.0E-04	0.875	1.0E-04	1.0E-02	1.0E-13	64	3	280.35
55	1.0E-05	0.875	1	1.0E-10	1.0E-13	128	1	235.55
56	1.0E-04	0.75	1.0E-07	1.0E-02	1.0E-13	256	3	637.95
57	1.0E-04	0.99	1.0E-04	1.0E-06	1.0E-07	128	5	474.38
58	1.0E-05	0.99	1.0E-04	1.0E-06	1.0E-01	256	5	790.79
59	1.0E-04	0.875	1.0E-04	1.0E-02	1.0E-13	128	2	316.88
60	1.0E-03	0.875	1.0E-07	1.0E-10	1.0E-07	64	2	258.73

61	1.0E-03	0.99	1	1.0E-10	1.0E-13	64	1	230.87
62	1.0E-04	0.75	1.0E-04	1.0E-10	1.0E-07	256	4	769.78
63	1.0E-03	0.99	1.0E-04	1.0E-02	1.0E-07	256	4	573.54
64	1.0E-04	0.99	1.0E-07	1.0E-02	1.0E-07	64	5	332.82
65	1.0E-03	0.875	1.0E-07	1.0E-06	1.0E-07	256	2	393.66
66	1.0E-04	0.75	1	1.0E-10	1.0E-07	64	5	308.62
67	1.0E-05	0.75	1.0E-04	1.0E-02	1.0E-01	128	2	294.21
68	1.0E-03	0.75	1.0E-04	1.0E-02	1.0E-01	128	5	415.71
69	1.0E-03	0.875	1	1.0E-02	1.0E-13	128	3	332.45
70	1.0E-03	0.875	1	1.0E-06	1.0E-13	64	3	262.32
71	1.0E-04	0.99	1	1.0E-10	1.0E-13	64	1	223.21
72	1.0E-04	0.875	1	1.0E-02	1.0E-13	64	3	281.38
73	1.0E-04	0.75	1.0E-04	1.0E-06	1.0E-01	64	4	293.41
74	1.0E-03	0.99	1	1.0E-06	1.0E-01	256	2	373.42
75	1.0E-04	0.75	1.0E-04	1.0E-06	1.0E-01	256	2	436.15
76	1.0E-03	0.75	1.0E-04	1.0E-02	1.0E-01	64	3	276.35
77	1.0E-03	0.75	1.0E-07	1.0E-02	1.0E-01	64	1	239.14
78	1.0E-03	0.75	1.0E-07	1.0E-02	1.0E-07	256	1	314.53
79	1.0E-05	0.75	1	1.0E-06	1.0E-01	64	2	244.85
80	1.0E-03	0.99	1.0E-04	1.0E-02	1.0E-13	128	1	265.12
81	1.0E-03	0.75	1.0E-04	1.0E-06	1.0E-01	128	1	261.08
82	1.0E-04	0.99	1.0E-04	1.0E-06	1.0E-07	256	4	772.02
83	1.0E-03	0.875	1	1.0E-02	1.0E-01	256	5	654.99
84	1.0E-03	0.875	1.0E-04	1.0E-02	1.0E-13	64	5	322.35
85	1.0E-04	0.99	1	1.0E-06	1.0E-07	64	5	300.83
86	1.0E-05	0.875	1	1.0E-10	1.0E-13	64	4	286.59
87	1.0E-03	0.875	1	1.0E-06	1.0E-01	256	1	293.48
88	1.0E-04	0.875	1.0E-04	1.0E-06	1.0E-01	128	5	422.82
89	1.0E-05	0.875	1	1.0E-02	1.0E-01	128	3	335.82
90	1.0E-04	0.99	1.0E-07	1.0E-10	1.0E-13	256	1	298.31
91	1.0E-04	0.875	1.0E-07	1.0E-02	1.0E-13	64	5	324.47
92	1.0E-03	0.875	1	1.0E-10	1.0E-07	128	2	276.25
93	1.0E-05	0.75	1	1.0E-02	1.0E-07	64	4	290.45
94	1.0E-03	0.875	1	1.0E-02	1.0E-07	64	3	275.21
95	1.0E-04	0.875	1.0E-04	1.0E-02	1.0E-07	256	4	761.86
96	1.0E-05	0.99	1	1.0E-06	1.0E-13	128	4	381.03
97	1.0E-04	0.75	1.0E-04	1.0E-02	1.0E-07	128	1	255.25
98	1.0E-03	0.99	1	1.0E-10	1.0E-07	64	4	277.80
99	1.0E-03	0.99	1.0E-07	1.0E-06	1.0E-13	256	5	661.38
100	1.0E-03	0.875	1	1.0E-06	1.0E-01	64	5	297.91
101	1.0E-04	0.99	1	1.0E-02	1.0E-01	256	5	807.25
102	1.0E-05	0.75	1.0E-04	1.0E-02	1.0E-07	128	4	458.26
103	1.0E-03	0.99	1.0E-04	1.0E-06	1.0E-07	128	5	465.36
104	1.0E-04	0.875	1	1.0E-06	1.0E-13	64	3	307.96
105	1.0E-04	0.75	1.0E-07	1.0E-06	1.0E-01	128	5	407.99
106	1.0E-03	0.875	1	1.0E-06	1.0E-01	64	4	278.15
107	1.0E-03	0.875	1.0E-07	1.0E-02	1.0E-13	128	5	423.35
108	1.0E-03	0.875	1.0E-07	1.0E-02	1.0E-07	256	5	657.58
109	1.0E-03	0.875	1	1.0E-10	1.0E-07	256	5	619.60
110	1.0E-05	0.875	1.0E-04	1.0E-06	1.0E-07	256	2	413.64
111	1.0E-04	0.875	1.0E-04	1.0E-10	1.0E-01	64	2	250.10
112	1.0E-05	0.875	1.0E-07	1.0E-06	1.0E-07	128	2	305.74
113	1.0E-03	0.75	1	1.0E-10	1.0E-01	64	1	225.55
114	1.0E-05	0.75	1.0E-04	1.0E-06	1.0E-07	256	1	287.71
115	1.0E-04	0.875	1.0E-04	1.0E-02	1.0E-01	64	5	310.62
116	1.0E-03	0.75	1	1.0E-02	1.0E-13	128	2	297.11

117	1.0E-04	0.75	1.0E-07	1.0E-02	1.0E-13	128	2	299.68
118	1.0E-05	0.75	1.0E-04	1.0E-06	1.0E-07	128	5	436.91
119	1.0E-05	0.99	1.0E-07	1.0E-10	1.0E-13	256	1	289.44
120	1.0E-04	0.875	1.0E-07	1.0E-02	1.0E-13	256	2	476.18
121	1.0E-04	0.75	1.0E-07	1.0E-10	1.0E-07	256	2	486.54
122	1.0E-04	0.875	1.0E-04	1.0E-10	1.0E-13	64	2	253.78
123	1.0E-04	0.75	1.0E-07	1.0E-06	1.0E-13	256	1	297.12
124	1.0E-05	0.99	1.0E-04	1.0E-06	1.0E-01	128	4	395.89
125	1.0E-04	0.875	1.0E-07	1.0E-02	1.0E-01	128	3	339.91
126	1.0E-05	0.875	1.0E-07	1.0E-06	1.0E-07	256	4	790.78
127	1.0E-03	0.99	1.0E-07	1.0E-02	1.0E-07	128	4	379.73
128	1.0E-03	0.99	1.0E-07	1.0E-02	1.0E-13	128	4	374.58
129	1.0E-05	0.875	1.0E-04	1.0E-10	1.0E-13	64	1	237.63
130	1.0E-04	0.875	1.0E-04	1.0E-02	1.0E-13	256	5	1036.75
131	1.0E-04	0.875	1.0E-04	1.0E-10	1.0E-07	64	5	325.25
132	1.0E-03	0.875	1.0E-07	1.0E-02	1.0E-13	64	5	325.73
133	1.0E-05	0.75	1.0E-07	1.0E-06	1.0E-07	256	3	616.15
134	1.0E-04	0.75	1	1.0E-02	1.0E-01	256	1	281.51
135	1.0E-03	0.75	1	1.0E-02	1.0E-07	64	2	261.24
136	1.0E-04	0.75	1.0E-07	1.0E-06	1.0E-01	64	1	239.55
137	1.0E-04	0.875	1	1.0E-06	1.0E-13	256	2	416.47
138	1.0E-04	0.875	1	1.0E-10	1.0E-01	64	2	244.80
139	1.0E-04	0.875	1.0E-04	1.0E-02	1.0E-01	64	1	236.37
140	1.0E-04	0.99	1.0E-04	1.0E-06	1.0E-13	64	2	255.60
141	1.0E-05	0.75	1	1.0E-06	1.0E-13	128	1	238.06
142	1.0E-05	0.99	1.0E-04	1.0E-10	1.0E-13	64	4	311.06
143	1.0E-04	0.75	1.0E-04	1.0E-10	1.0E-07	128	4	443.99
144	1.0E-03	0.99	1	1.0E-06	1.0E-13	256	5	634.18
145	1.0E-04	0.875	1.0E-04	1.0E-10	1.0E-01	64	1	231.14
146	1.0E-04	0.99	1	1.0E-06	1.0E-01	64	3	264.91
147	1.0E-05	0.99	1.0E-04	1.0E-06	1.0E-13	128	1	269.53
148	1.0E-03	0.75	1	1.0E-06	1.0E-01	256	4	538.12
149	1.0E-04	0.99	1	1.0E-02	1.0E-07	64	4	302.75
150	1.0E-04	0.75	1.0E-07	1.0E-02	1.0E-07	128	2	310.00
151	1.0E-03	0.75	1.0E-04	1.0E-06	1.0E-07	128	1	265.96
152	1.0E-05	0.99	1.0E-04	1.0E-06	1.0E-13	256	2	503.47
153	1.0E-05	0.75	1.0E-07	1.0E-06	1.0E-13	64	4	298.03
154	1.0E-03	0.75	1	1.0E-02	1.0E-07	256	4	573.69
155	1.0E-03	0.99	1.0E-04	1.0E-06	1.0E-07	64	5	322.68
156	1.0E-04	0.75	1.0E-04	1.0E-02	1.0E-01	256	2	420.75
157	1.0E-03	0.875	1.0E-07	1.0E-02	1.0E-13	256	2	401.28
158	1.0E-04	0.99	1.0E-04	1.0E-02	1.0E-13	64	4	320.67
159	1.0E-04	0.75	1.0E-07	1.0E-06	1.0E-13	64	3	294.98
160	1.0E-03	0.875	1	1.0E-06	1.0E-07	64	5	305.05
161	1.0E-05	0.75	1.0E-04	1.0E-02	1.0E-13	128	5	425.14
162	1.0E-04	0.75	1	1.0E-02	1.0E-01	64	5	352.25
163	1.0E-05	0.75	1.0E-04	1.0E-02	1.0E-01	256	3	593.94
164	1.0E-03	0.75	1.0E-07	1.0E-10	1.0E-13	64	5	377.02
165	1.0E-03	0.99	1.0E-04	1.0E-10	1.0E-07	256	5	697.00
166	1.0E-04	0.875	1.0E-07	1.0E-02	1.0E-07	128	3	362.86
167	1.0E-05	0.75	1.0E-04	1.0E-10	1.0E-01	128	2	306.49
168	1.0E-05	0.875	1.0E-07	1.0E-10	1.0E-01	256	2	420.97
169	1.0E-04	0.99	1.0E-04	1.0E-06	1.0E-01	256	3	520.48
170	1.0E-03	0.875	1.0E-04	1.0E-02	1.0E-01	256	4	556.01
171	1.0E-05	0.875	1.0E-04	1.0E-06	1.0E-13	128	1	253.96
172	1.0E-03	0.75	1.0E-07	1.0E-02	1.0E-07	256	1	313.94

173	1.0E-03	0.875	1.0E-04	1.0E-02	1.0E-13	128	2	304.61
174	1.0E-03	0.875	1.0E-07	1.0E-02	1.0E-01	64	1	245.58
175	1.0E-04	0.99	1	1.0E-02	1.0E-13	64	2	258.71
176	1.0E-05	0.75	1.0E-04	1.0E-10	1.0E-13	64	5	326.66
177	1.0E-04	0.99	1.0E-04	1.0E-02	1.0E-07	128	1	264.59
178	1.0E-05	0.99	1.0E-07	1.0E-10	1.0E-07	64	1	234.92
179	1.0E-04	0.99	1.0E-04	1.0E-10	1.0E-13	128	5	455.25
180	1.0E-03	0.875	1	1.0E-02	1.0E-01	128	4	372.76
181	1.0E-04	0.875	1.0E-07	1.0E-02	1.0E-13	256	2	447.19
182	1.0E-04	0.99	1.0E-04	1.0E-06	1.0E-07	64	2	265.72
183	1.0E-03	0.99	1	1.0E-10	1.0E-07	256	2	369.75
184	1.0E-04	0.75	1.0E-04	1.0E-10	1.0E-01	128	4	375.65
185	1.0E-05	0.875	1	1.0E-06	1.0E-01	256	1	271.29
186	1.0E-04	0.75	1.0E-04	1.0E-02	1.0E-07	128	4	423.68
187	1.0E-03	0.75	1.0E-07	1.0E-06	1.0E-13	64	5	338.72
188	1.0E-05	0.75	1	1.0E-06	1.0E-07	128	3	334.96
189	1.0E-05	0.99	1.0E-04	1.0E-06	1.0E-01	64	3	280.59
190	1.0E-04	0.875	1.0E-04	1.0E-06	1.0E-07	256	2	465.34
191	1.0E-03	0.875	1	1.0E-10	1.0E-13	256	5	637.43
192	1.0E-04	0.99	1	1.0E-02	1.0E-13	256	2	439.62
193	1.0E-03	0.99	1	1.0E-06	1.0E-13	256	1	305.00
194	1.0E-03	0.875	1.0E-04	1.0E-06	1.0E-07	256	3	483.77
195	1.0E-04	0.75	1.0E-07	1.0E-02	1.0E-07	64	4	307.60
196	1.0E-03	0.75	1.0E-04	1.0E-02	1.0E-07	64	4	305.25
197	1.0E-05	0.875	1.0E-04	1.0E-06	1.0E-01	256	2	386.88
198	1.0E-05	0.875	1.0E-04	1.0E-06	1.0E-13	128	1	260.71
199	1.0E-03	0.75	1	1.0E-10	1.0E-13	256	1	300.81
200	1.0E-04	0.75	1.0E-07	1.0E-06	1.0E-01	128	1	261.81
201	1.0E-05	0.99	1.0E-04	1.0E-06	1.0E-01	128	2	293.98
202	1.0E-04	0.875	1.0E-04	1.0E-06	1.0E-07	128	1	264.58
203	1.0E-04	0.99	1	1.0E-02	1.0E-07	64	1	236.87
204	1.0E-05	0.75	1.0E-04	1.0E-10	1.0E-07	256	4	776.13
205	1.0E-03	0.875	1.0E-07	1.0E-06	1.0E-07	64	5	336.20
206	1.0E-05	0.99	1.0E-07	1.0E-02	1.0E-01	256	1	284.99
207	1.0E-03	0.75	1	1.0E-06	1.0E-13	256	4	544.60
208	1.0E-05	0.75	1	1.0E-06	1.0E-01	256	3	531.06
209	1.0E-04	0.99	1	1.0E-02	1.0E-07	256	4	649.17
210	1.0E-05	0.99	1.0E-07	1.0E-02	1.0E-13	256	5	1135.86
211	1.0E-03	0.75	1	1.0E-06	1.0E-13	64	3	270.21
212	1.0E-03	0.99	1.0E-07	1.0E-02	1.0E-07	64	4	307.23
213	1.0E-05	0.75	1	1.0E-06	1.0E-13	128	4	384.97
214	1.0E-03	0.875	1	1.0E-10	1.0E-01	128	1	270.56
215	1.0E-03	0.875	1.0E-07	1.0E-10	1.0E-07	256	4	575.55
216	1.0E-04	0.875	1	1.0E-10	1.0E-07	256	3	530.46
217	1.0E-05	0.99	1.0E-07	1.0E-02	1.0E-13	128	3	369.04
218	1.0E-05	0.99	1	1.0E-02	1.0E-01	128	3	338.27
219	1.0E-05	0.99	1.0E-04	1.0E-02	1.0E-01	64	5	330.10
220	1.0E-03	0.875	1	1.0E-10	1.0E-01	128	2	293.13
221	1.0E-03	0.99	1.0E-04	1.0E-10	1.0E-07	256	2	446.53
222	1.0E-03	0.99	1	1.0E-10	1.0E-01	64	1	275.27
223	1.0E-03	0.875	1.0E-04	1.0E-06	1.0E-13	128	2	355.96
224	1.0E-04	0.99	1.0E-07	1.0E-10	1.0E-13	256	3	696.67
225	1.0E-03	0.99	1	1.0E-06	1.0E-01	64	1	236.15
226	1.0E-04	0.875	1	1.0E-02	1.0E-01	256	1	285.11
227	1.0E-04	0.75	1.0E-07	1.0E-06	1.0E-07	64	4	317.13
228	1.0E-03	0.875	1.0E-04	1.0E-06	1.0E-13	64	1	247.62

229	1.0E-04	0.99	1	1.0E-02	1.0E-07	256	2	441.71
230	1.0E-03	0.875	1.0E-04	1.0E-10	1.0E-01	256	2	383.40
231	1.0E-04	0.99	1.0E-04	1.0E-06	1.0E-01	256	4	663.00
232	1.0E-03	0.75	1.0E-04	1.0E-10	1.0E-01	256	2	395.33
233	1.0E-04	0.875	1.0E-07	1.0E-10	1.0E-01	128	1	257.58
234	1.0E-03	0.99	1.0E-04	1.0E-06	1.0E-13	64	1	250.32
235	1.0E-04	0.875	1.0E-04	1.0E-06	1.0E-01	128	2	297.33
236	1.0E-04	0.875	1.0E-07	1.0E-06	1.0E-01	64	5	310.79
237	1.0E-05	0.75	1	1.0E-06	1.0E-01	64	5	307.36
238	1.0E-04	0.99	1	1.0E-06	1.0E-01	128	2	290.46
239	1.0E-05	0.75	1.0E-07	1.0E-02	1.0E-13	256	2	516.44
240	1.0E-05	0.875	1.0E-07	1.0E-02	1.0E-01	128	1	255.69
241	1.0E-03	0.75	1.0E-07	1.0E-02	1.0E-07	64	5	321.81
242	1.0E-03	0.75	1	1.0E-02	1.0E-13	64	5	314.74
243	1.0E-03	0.875	1.0E-04	1.0E-10	1.0E-13	256	1	324.15
244	1.0E-05	0.99	1	1.0E-06	1.0E-13	128	1	239.38
245	1.0E-04	0.99	1.0E-07	1.0E-10	1.0E-01	256	5	768.07
246	1.0E-05	0.875	1	1.0E-06	1.0E-07	256	1	272.28
247	1.0E-05	0.875	1.0E-07	1.0E-10	1.0E-13	256	5	1068.44
248	1.0E-05	0.99	1.0E-07	1.0E-10	1.0E-07	64	2	268.14
249	1.0E-05	0.75	1.0E-07	1.0E-02	1.0E-01	256	1	283.13
250	1.0E-03	0.875	1.0E-04	1.0E-10	1.0E-13	256	4	588.08

Table 66: All settings for the agents created in test 3

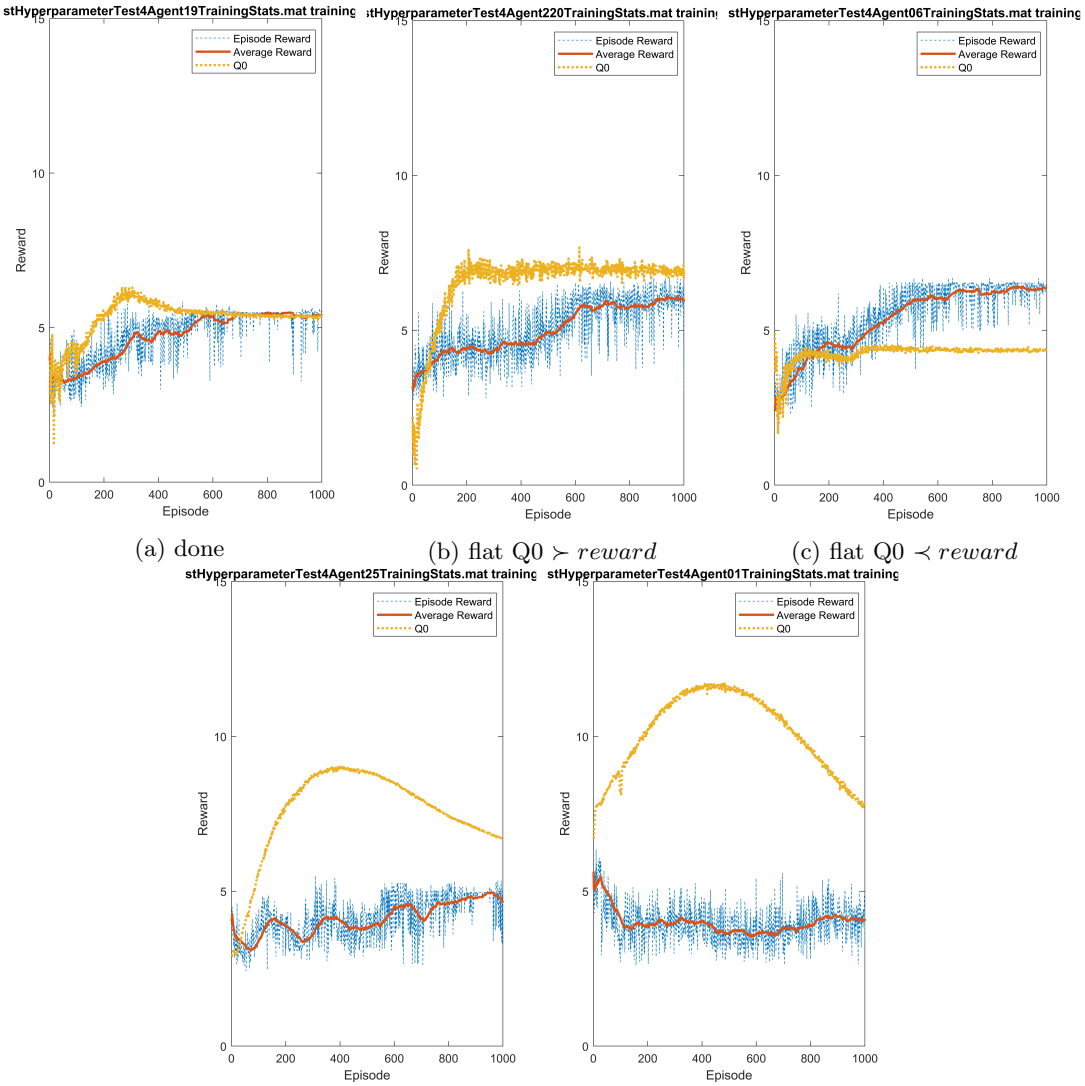
K.4 Fourth iteration

the new settings are shown in Table 67.

LR	DF	ϵ	ϵ_m	ϵ_d	Neurons	Hidden layers
1e-3 1e-4 1e-5	0.75, 0.875, 0.99	0.1 0.5 1	1e-13 1e-7 0.01	1e-3 5e-4 1e-4	64 128 256	3 to 5

Table 67: Possible values fourth initialization hyperparameter test

With these settings a lot more consistency is found. Again categorizing the graphs, into 5 categories this time. The categories are done learning (meaning flat Q0 on same level as average reward gained), a flat Q0 above the gained rewards, a flat Q0 below the gained rewards, Q0 that is still high with increasing rewards and others. Figure 65 shows a training graph per category.



(a) done (b) flat Q0 $\succ$ *reward* (c) flat Q0 $\prec$ *reward*

(d) Q0 high (e) Others

Figure 65: Representation of different categories in test 4

When categorizing and looking at the different parameters, given in Table 69 and the categorized agents in Table 68, the first category shows that most of these graphs have a ϵ_d of 0.1, indicating little exploration, hence not learning. The second category is found to have a discount factor of 0.99 with a learning rate of 1e-3, considering the category, that means that it learns to quickly and is overestimating the Q0 values. The third category shows a discount factor of 0.75, which is the lowest. The category means that the the agent is underestimating the Q0 values. The fourth category shows that a high discount factor, being 0.99, shows that the agent is able to estimate the rewards better as well as still increasing the average reward. The others category shows lower learning rates with lower discount factors, making the learning behavior unpredictable, thus not being able to really place them into a category.

Category	Agents
Low & flat Q0	10 13 14 16 20 24 27 30 35 36 39 41 46 50 55 56 69 77 79 81 90 98
High & flat Q0	5 11 17 21 32 42 49 65 71 73 83 88
Overshoot	4 6 8 12 15 22 33 34 52 57 61 63 64 76 78 80 82 85 87 94 95 96
No Q0 found	40 48 51 62 84 86 92
Overshoot (2) below reward	2 25 29 59 67 68 70 93 97 99 100
Spikes	1 3 5 7 18 19 26 28 37 43 44 45 53 54 60 66 72 74 91
Undershoot/instant decrease	9 23 31 38 47 58 89

Table 68: Overview of different categories and agents in test 2

Agent	LR	DF	ϵ	ϵ_d	ϵ_m	Neurons	HL	CT
1	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	64	4	260.17
2	1.0E-05	0.99	1	1.0E-04	1.0E-13	256	4	583.30
3	1.0E-04	0.875	1	5.0E-04	1.0E-02	256	5	746.72
4	1.0E-04	0.75	1	1.0E-04	1.0E-02	64	3	263.03
5	1.0E-03	0.75	1	5.0E-04	1.0E-13	256	5	663.70
6	1.0E-04	0.75	0.1	1.0E-03	1.0E-07	64	3	257.79
7	1.0E-05	0.99	1	5.0E-04	1.0E-07	64	3	257.64
8	1.0E-03	0.75	0.5	1.0E-04	1.0E-02	256	4	540.61
9	1.0E-03	0.75	0.1	1.0E-03	1.0E-13	256	5	662.32
10	1.0E-05	0.75	1	1.0E-03	1.0E-13	256	5	983.54
11	1.0E-05	0.75	0.1	1.0E-03	1.0E-02	256	3	562.19
12	1.0E-03	0.875	0.1	1.0E-03	1.0E-02	256	3	461.01
13	1.0E-03	0.875	1	5.0E-04	1.0E-02	128	4	363.32
14	1.0E-04	0.99	0.1	1.0E-04	1.0E-02	64	5	293.11
15	1.0E-05	0.875	0.1	1.0E-03	1.0E-13	256	5	892.80
16	1.0E-03	0.99	0.5	1.0E-03	1.0E-02	256	3	449.66
17	1.0E-04	0.75	0.1	1.0E-04	1.0E-02	64	4	277.11
18	1.0E-04	0.875	1	5.0E-04	1.0E-02	256	5	786.16
19	1.0E-04	0.875	0.1	5.0E-04	1.0E-07	128	3	327.76
20	1.0E-03	0.75	0.1	1.0E-03	1.0E-13	256	3	458.55
21	1.0E-03	0.75	1	1.0E-03	1.0E-13	128	3	324.59
22	1.0E-04	0.99	1	5.0E-04	1.0E-13	256	3	542.60
23	1.0E-05	0.875	0.5	1.0E-03	1.0E-13	128	4	372.59
24	1.0E-04	0.875	1	1.0E-03	1.0E-13	256	3	514.57
25	1.0E-05	0.99	1	1.0E-03	1.0E-07	64	5	303.48
26	1.0E-04	0.99	0.5	1.0E-03	1.0E-02	256	3	523.42
27	1.0E-05	0.875	0.1	1.0E-03	1.0E-13	64	4	279.97
28	1.0E-04	0.75	1	1.0E-04	1.0E-13	64	4	282.35
29	1.0E-04	0.875	1	1.0E-04	1.0E-13	64	4	282.15
30	1.0E-05	0.99	1	1.0E-03	1.0E-02	64	5	303.15
31	1.0E-04	0.875	1	1.0E-03	1.0E-02	256	3	525.19
32	1.0E-04	0.99	0.1	1.0E-03	1.0E-02	256	3	512.28
33	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	128	3	345.44
34	1.0E-03	0.875	0.5	5.0E-04	1.0E-07	128	3	321.96
35	1.0E-04	0.99	0.5	5.0E-04	1.0E-02	128	5	422.29
36	1.0E-05	0.75	1	1.0E-03	1.0E-13	256	5	833.08
37	1.0E-03	0.99	0.5	1.0E-03	1.0E-02	64	4	282.32
38	1.0E-03	0.75	0.1	1.0E-03	1.0E-13	128	5	400.20
39	1.0E-04	0.75	0.5	1.0E-03	1.0E-02	128	3	327.14
40	1.0E-03	0.875	0.5	5.0E-04	1.0E-13	256	5	641.49
41	1.0E-03	0.75	0.5	5.0E-04	1.0E-13	64	3	264.73
42	1.0E-05	0.99	0.1	5.0E-04	1.0E-02	256	3	533.83
43	1.0E-03	0.875	0.5	1.0E-03	1.0E-07	64	5	303.96
44	1.0E-05	0.75	0.5	1.0E-04	1.0E-07	256	4	715.71

45	1.0E-05	0.99	1	1.0E-04	1.0E-13	128	5	424.06
46	1.0E-04	0.875	1	1.0E-04	1.0E-02	128	5	430.39
47	1.0E-03	0.99	0.1	5.0E-04	1.0E-13	64	4	284.63
48	1.0E-03	0.875	0.5	5.0E-04	1.0E-13	128	4	387.91
49	1.0E-04	0.875	0.5	1.0E-04	1.0E-13	64	4	282.37
50	1.0E-04	0.75	0.5	1.0E-04	1.0E-07	128	4	363.44
51	1.0E-03	0.875	0.1	1.0E-03	1.0E-13	128	3	324.23
52	1.0E-05	0.875	0.1	1.0E-04	1.0E-13	128	3	325.02
53	1.0E-04	0.99	1	1.0E-03	1.0E-02	64	5	298.15
54	1.0E-04	0.875	0.1	1.0E-04	1.0E-02	256	3	520.92
55	1.0E-05	0.75	0.1	5.0E-04	1.0E-13	128	4	369.55
56	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	128	4	368.61
57	1.0E-04	0.75	1	5.0E-04	1.0E-02	256	3	536.03
58	1.0E-03	0.99	1	1.0E-03	1.0E-13	256	3	459.69
59	1.0E-03	0.75	0.1	1.0E-04	1.0E-13	128	4	370.61
60	1.0E-05	0.99	1	1.0E-04	1.0E-02	128	3	324.33
61	1.0E-03	0.875	1	1.0E-03	1.0E-07	64	4	285.81
62	1.0E-04	0.99	1	1.0E-04	1.0E-02	128	3	328.71
63	1.0E-05	0.875	0.1	5.0E-04	1.0E-13	128	4	415.57
64	1.0E-05	0.75	1	1.0E-03	1.0E-13	128	4	424.55
65	1.0E-05	0.75	1	5.0E-04	1.0E-07	256	4	797.84
66	1.0E-05	0.99	0.1	1.0E-03	1.0E-13	128	3	338.12
67	1.0E-05	0.75	1	5.0E-04	1.0E-07	256	5	821.77
68	1.0E-05	0.99	1	1.0E-03	1.0E-13	128	3	329.06
69	1.0E-04	0.99	0.1	1.0E-03	1.0E-02	256	5	791.45
70	1.0E-04	0.875	1	1.0E-03	1.0E-02	64	5	312.69
71	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	64	5	314.16
72	1.0E-04	0.75	0.1	1.0E-03	1.0E-13	64	3	266.23
73	1.0E-03	0.99	0.1	1.0E-03	1.0E-02	256	5	629.46
74	1.0E-05	0.875	1	1.0E-03	1.0E-13	128	5	424.75
75	1.0E-05	0.75	0.1	5.0E-04	1.0E-02	256	4	741.87
76	1.0E-03	0.75	0.5	5.0E-04	1.0E-13	64	3	266.68
77	1.0E-05	0.75	1	5.0E-04	1.0E-07	128	5	416.65
78	1.0E-05	0.99	1	1.0E-04	1.0E-07	256	3	512.75
79	1.0E-04	0.75	0.1	1.0E-04	1.0E-13	256	4	607.30
80	1.0E-03	0.75	0.5	1.0E-03	1.0E-02	64	5	308.08
81	1.0E-03	0.75	0.5	1.0E-04	1.0E-13	64	4	284.28
82	1.0E-04	0.75	0.1	1.0E-04	1.0E-07	64	5	309.91
83	1.0E-03	0.875	1	5.0E-04	1.0E-13	64	5	303.25
84	1.0E-04	0.75	0.5	1.0E-03	1.0E-07	256	3	534.46
85	1.0E-05	0.75	0.5	1.0E-04	1.0E-13	256	3	519.87
86	1.0E-05	0.75	0.1	1.0E-04	1.0E-13	256	3	546.22
87	1.0E-03	0.75	0.1	1.0E-04	1.0E-02	64	4	284.51
88	1.0E-05	0.75	0.1	1.0E-03	1.0E-07	64	5	311.72
89	1.0E-04	0.75	1	1.0E-03	1.0E-13	128	5	421.91
90	1.0E-04	0.99	0.5	1.0E-04	1.0E-07	256	5	682.32
91	1.0E-05	0.99	0.1	1.0E-03	1.0E-07	128	4	384.14
92	1.0E-04	0.875	1	1.0E-03	1.0E-07	64	5	307.44
93	1.0E-04	0.99	0.1	1.0E-04	1.0E-02	128	3	336.62
94	1.0E-05	0.99	0.1	1.0E-04	1.0E-07	128	4	387.65
95	1.0E-04	0.99	0.5	1.0E-03	1.0E-07	64	3	260.96
96	1.0E-04	0.875	0.1	1.0E-04	1.0E-02	128	3	324.52
97	1.0E-03	0.99	1	1.0E-04	1.0E-02	256	5	632.16
98	1.0E-04	0.875	1	1.0E-03	1.0E-13	256	4	686.43
99	1.0E-04	0.75	1	1.0E-04	1.0E-02	256	4	631.40
100	1.0E-04	0.875	0.1	1.0E-03	1.0E-07	64	4	289.00

101	1.0E-03	0.99	0.1	1.0E-03	1.0E-07	256	4	545.23
102	1.0E-04	0.99	0.5	1.0E-04	1.0E-02	128	5	421.98
103	1.0E-03	0.99	0.1	1.0E-03	1.0E-07	256	4	534.01
104	1.0E-05	0.99	0.5	5.0E-04	1.0E-13	128	4	381.64
105	1.0E-05	0.875	0.1	1.0E-04	1.0E-07	64	3	267.17
106	1.0E-05	0.75	1	1.0E-04	1.0E-13	256	3	544.75
107	1.0E-04	0.99	0.5	1.0E-03	1.0E-02	256	5	784.59
108	1.0E-05	0.99	0.5	1.0E-03	1.0E-02	64	4	282.71
109	1.0E-04	0.875	0.5	1.0E-04	1.0E-02	256	4	603.84
110	1.0E-03	0.75	0.1	1.0E-03	1.0E-02	256	3	473.43
111	1.0E-04	0.875	0.1	1.0E-04	1.0E-13	64	4	293.25
112	1.0E-04	0.75	0.5	5.0E-04	1.0E-13	64	4	285.66
113	1.0E-03	0.75	0.1	5.0E-04	1.0E-13	64	4	291.09
114	1.0E-04	0.75	0.5	1.0E-03	1.0E-02	128	4	405.15
115	1.0E-05	0.75	0.1	1.0E-04	1.0E-07	256	4	768.17
116	1.0E-03	0.875	0.5	1.0E-03	1.0E-02	64	4	330.94
117	1.0E-04	0.75	0.5	1.0E-04	1.0E-07	64	4	320.23
118	1.0E-03	0.875	0.1	1.0E-03	1.0E-07	64	4	291.22
119	1.0E-03	0.875	0.5	1.0E-03	1.0E-07	256	4	543.75
120	1.0E-04	0.75	0.1	5.0E-04	1.0E-07	64	4	287.60
121	1.0E-05	0.75	1	1.0E-04	1.0E-07	256	3	544.81
122	1.0E-04	0.75	0.1	5.0E-04	1.0E-02	256	4	639.79
123	1.0E-03	0.75	0.5	1.0E-03	1.0E-13	256	3	469.77
124	1.0E-05	0.875	0.5	5.0E-04	1.0E-07	128	4	372.99
125	1.0E-04	0.875	0.1	1.0E-04	1.0E-07	128	3	338.70
126	1.0E-03	0.99	1	1.0E-03	1.0E-13	128	5	405.66
127	1.0E-04	0.75	1	1.0E-03	1.0E-13	128	3	337.72
128	1.0E-05	0.75	0.1	1.0E-04	1.0E-13	128	3	332.93
129	1.0E-04	0.99	0.5	1.0E-03	1.0E-02	128	3	331.52
130	1.0E-04	0.875	0.5	1.0E-04	1.0E-07	128	5	428.26
131	1.0E-03	0.75	1	1.0E-03	1.0E-13	256	5	660.70
132	1.0E-05	0.75	1	1.0E-03	1.0E-07	256	4	730.45
133	1.0E-04	0.75	0.1	1.0E-03	1.0E-02	256	4	644.99
134	1.0E-03	0.875	1	1.0E-04	1.0E-07	256	3	460.47
135	1.0E-05	0.99	0.5	5.0E-04	1.0E-13	64	5	313.71
136	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	128	3	339.05
137	1.0E-05	0.875	0.1	1.0E-04	1.0E-07	256	5	829.90
138	1.0E-05	0.875	0.1	1.0E-03	1.0E-07	64	3	267.05
139	1.0E-04	0.875	1	5.0E-04	1.0E-02	128	4	383.96
140	1.0E-05	0.75	1	5.0E-04	1.0E-13	128	3	334.26
141	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	256	3	503.19
142	1.0E-05	0.875	0.1	1.0E-03	1.0E-07	128	4	369.96
143	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	256	3	554.04
144	1.0E-03	0.99	0.5	1.0E-03	1.0E-13	256	5	636.49
145	1.0E-05	0.99	0.5	1.0E-04	1.0E-13	256	5	880.30
146	1.0E-05	0.875	0.1	1.0E-04	1.0E-13	128	5	440.52
147	1.0E-05	0.75	1	1.0E-03	1.0E-02	256	4	729.13
148	1.0E-03	0.75	0.1	1.0E-04	1.0E-02	64	4	288.47
149	1.0E-03	0.875	0.1	1.0E-04	1.0E-07	128	5	411.96
150	1.0E-04	0.875	0.5	1.0E-03	1.0E-13	64	4	289.38
151	1.0E-05	0.99	1	1.0E-04	1.0E-02	128	3	335.17
152	1.0E-04	0.875	1	5.0E-04	1.0E-13	256	4	671.41
153	1.0E-04	0.75	1	1.0E-04	1.0E-13	128	3	346.97
154	1.0E-04	0.875	1	1.0E-04	1.0E-02	256	3	513.59
155	1.0E-05	0.75	0.5	1.0E-04	1.0E-02	128	5	444.69
156	1.0E-04	0.99	0.1	5.0E-04	1.0E-13	64	5	307.73

157	1.0E-04	0.75	0.5	1.0E-03	1.0E-07	64	3	268.17
158	1.0E-03	0.875	1	5.0E-04	1.0E-07	256	4	560.04
159	1.0E-05	0.875	1	5.0E-04	1.0E-02	256	4	674.09
160	1.0E-03	0.99	1	1.0E-03	1.0E-13	64	5	310.34
161	1.0E-03	0.875	1	5.0E-04	1.0E-07	256	3	483.51
162	1.0E-04	0.99	1	5.0E-04	1.0E-13	128	4	394.89
163	1.0E-05	0.875	1	5.0E-04	1.0E-07	256	3	559.72
164	1.0E-03	0.75	0.1	1.0E-03	1.0E-07	128	5	451.64
165	1.0E-03	0.875	1	5.0E-04	1.0E-02	128	3	378.45
166	1.0E-05	0.99	0.5	5.0E-04	1.0E-13	256	5	996.24
167	1.0E-05	0.875	0.1	1.0E-03	1.0E-07	256	3	538.31
168	1.0E-03	0.99	0.5	1.0E-03	1.0E-07	128	4	370.59
169	1.0E-03	0.99	1	5.0E-04	1.0E-02	64	5	304.21
170	1.0E-04	0.75	1	1.0E-03	1.0E-02	64	5	310.60
171	1.0E-04	0.75	1	5.0E-04	1.0E-13	128	4	395.54
172	1.0E-05	0.75	1	1.0E-04	1.0E-13	64	4	290.54
173	1.0E-05	0.875	0.5	1.0E-03	1.0E-07	256	5	802.67
174	1.0E-03	0.99	1	5.0E-04	1.0E-07	64	3	274.59
175	1.0E-05	0.875	0.1	5.0E-04	1.0E-02	128	5	424.70
176	1.0E-04	0.99	0.1	1.0E-03	1.0E-02	64	5	322.50
177	1.0E-03	0.75	0.1	5.0E-04	1.0E-07	256	3	482.50
178	1.0E-05	0.75	1	1.0E-04	1.0E-02	256	3	587.41
179	1.0E-03	0.875	0.1	1.0E-04	1.0E-02	64	3	275.45
180	1.0E-03	0.99	0.1	1.0E-04	1.0E-02	128	3	335.23
181	1.0E-03	0.75	0.5	1.0E-04	1.0E-07	128	3	330.99
182	1.0E-03	0.99	1	1.0E-03	1.0E-13	64	4	290.07
183	1.0E-03	0.875	0.5	1.0E-04	1.0E-07	128	3	328.61
184	1.0E-03	0.875	0.1	1.0E-04	1.0E-13	64	3	273.81
185	1.0E-05	0.75	0.1	1.0E-04	1.0E-07	128	3	349.75
186	1.0E-03	0.99	1	1.0E-04	1.0E-13	64	3	269.58
187	1.0E-05	0.75	0.5	1.0E-03	1.0E-07	64	5	320.96
188	1.0E-05	0.99	0.1	1.0E-03	1.0E-02	128	4	402.10
189	1.0E-05	0.75	0.1	5.0E-04	1.0E-07	256	4	767.26
190	1.0E-05	0.875	0.5	1.0E-03	1.0E-02	256	3	606.15
191	1.0E-04	0.99	0.5	1.0E-04	1.0E-07	256	3	555.26
192	1.0E-03	0.875	0.5	1.0E-03	1.0E-13	64	5	321.58
193	1.0E-04	0.99	0.1	1.0E-04	1.0E-07	64	5	318.34
194	1.0E-04	0.75	1	5.0E-04	1.0E-13	64	4	290.46
195	1.0E-05	0.875	1	1.0E-04	1.0E-07	128	5	464.11
196	1.0E-05	0.99	0.5	1.0E-03	1.0E-13	64	5	310.88
197	1.0E-03	0.75	1	1.0E-04	1.0E-07	64	5	305.59
198	1.0E-05	0.99	1	1.0E-03	1.0E-13	128	4	406.50
199	1.0E-03	0.75	0.1	5.0E-04	1.0E-02	256	4	578.96
200	1.0E-03	0.99	0.1	1.0E-04	1.0E-13	128	5	428.04
201	1.0E-04	0.99	0.5	1.0E-03	1.0E-02	64	5	310.88
202	1.0E-04	0.99	0.5	1.0E-04	1.0E-02	64	5	308.77
203	1.0E-04	0.875	0.5	5.0E-04	1.0E-02	64	5	305.85
204	1.0E-04	0.75	1	1.0E-04	1.0E-07	256	4	707.49
205	1.0E-03	0.75	0.5	1.0E-04	1.0E-02	128	5	425.98
206	1.0E-03	0.75	1	5.0E-04	1.0E-07	256	5	681.94
207	1.0E-04	0.99	0.1	1.0E-04	1.0E-07	256	5	777.83
208	1.0E-04	0.75	0.5	1.0E-03	1.0E-13	256	3	572.76
209	1.0E-04	0.875	0.5	5.0E-04	1.0E-02	64	3	270.04
210	1.0E-05	0.875	0.1	5.0E-04	1.0E-13	64	4	283.88
211	1.0E-04	0.75	0.5	1.0E-04	1.0E-02	128	5	451.48
212	1.0E-04	0.99	1	1.0E-04	1.0E-13	256	4	680.08

213	1.0E-04	0.75	0.1	1.0E-03	1.0E-02	256	3	530.31
214	1.0E-05	0.99	0.1	1.0E-03	1.0E-02	64	3	277.73
215	1.0E-05	0.75	1	1.0E-04	1.0E-02	128	5	450.31
216	1.0E-05	0.875	0.5	1.0E-03	1.0E-13	256	3	570.63
217	1.0E-05	0.75	1	1.0E-03	1.0E-02	256	5	892.16
218	1.0E-03	0.75	1	1.0E-03	1.0E-02	64	3	325.64
219	1.0E-05	0.75	0.5	1.0E-03	1.0E-13	64	3	318.56
220	1.0E-03	0.99	0.5	1.0E-04	1.0E-13	128	5	453.51
221	1.0E-04	0.99	0.1	1.0E-03	1.0E-02	64	4	288.39
222	1.0E-04	0.75	0.5	5.0E-04	1.0E-07	64	5	328.56
223	1.0E-05	0.99	1	1.0E-04	1.0E-07	64	3	272.50
224	1.0E-05	0.99	1	1.0E-04	1.0E-13	64	3	277.99
225	1.0E-04	0.99	1	5.0E-04	1.0E-02	256	4	696.82
226	1.0E-03	0.75	1	1.0E-04	1.0E-02	256	4	563.93
227	1.0E-05	0.875	1	1.0E-04	1.0E-07	256	4	710.89
228	1.0E-04	0.875	0.1	5.0E-04	1.0E-13	64	3	272.83
229	1.0E-03	0.75	0.5	1.0E-04	1.0E-13	64	3	272.12
230	1.0E-05	0.875	0.1	5.0E-04	1.0E-02	64	4	287.60
231	1.0E-05	0.75	0.5	1.0E-04	1.0E-13	64	4	293.08
232	1.0E-05	0.75	1	1.0E-04	1.0E-02	128	3	359.79
233	1.0E-05	0.75	1	1.0E-03	1.0E-07	128	4	403.45
234	1.0E-04	0.875	1	1.0E-03	1.0E-02	64	5	309.00
235	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	64	3	277.21
236	1.0E-03	0.75	0.1	5.0E-04	1.0E-13	256	3	494.33
237	1.0E-05	0.75	0.5	5.0E-04	1.0E-02	64	4	293.83
238	1.0E-05	0.875	0.1	5.0E-04	1.0E-02	256	5	924.44
239	1.0E-04	0.875	1	5.0E-04	1.0E-07	64	5	321.40
240	1.0E-05	0.875	0.1	1.0E-03	1.0E-02	128	3	360.94
241	1.0E-03	0.875	0.5	5.0E-04	1.0E-07	256	4	574.62
242	1.0E-03	0.75	0.5	1.0E-04	1.0E-02	64	5	311.10
243	1.0E-04	0.875	1	1.0E-03	1.0E-02	128	4	409.63
244	1.0E-04	0.75	0.5	5.0E-04	1.0E-13	256	5	910.58
245	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	128	3	361.25
246	1.0E-05	0.75	0.5	1.0E-04	1.0E-07	64	3	275.54
247	1.0E-03	0.99	0.1	1.0E-04	1.0E-13	256	4	577.54
248	1.0E-04	0.99	0.1	1.0E-04	1.0E-02	128	3	366.20
249	1.0E-04	0.875	0.1	5.0E-04	1.0E-13	256	5	809.21
250	1.0E-05	0.75	1	1.0E-04	1.0E-02	128	4	410.73

Table 69: All settings for the agents created in test 4

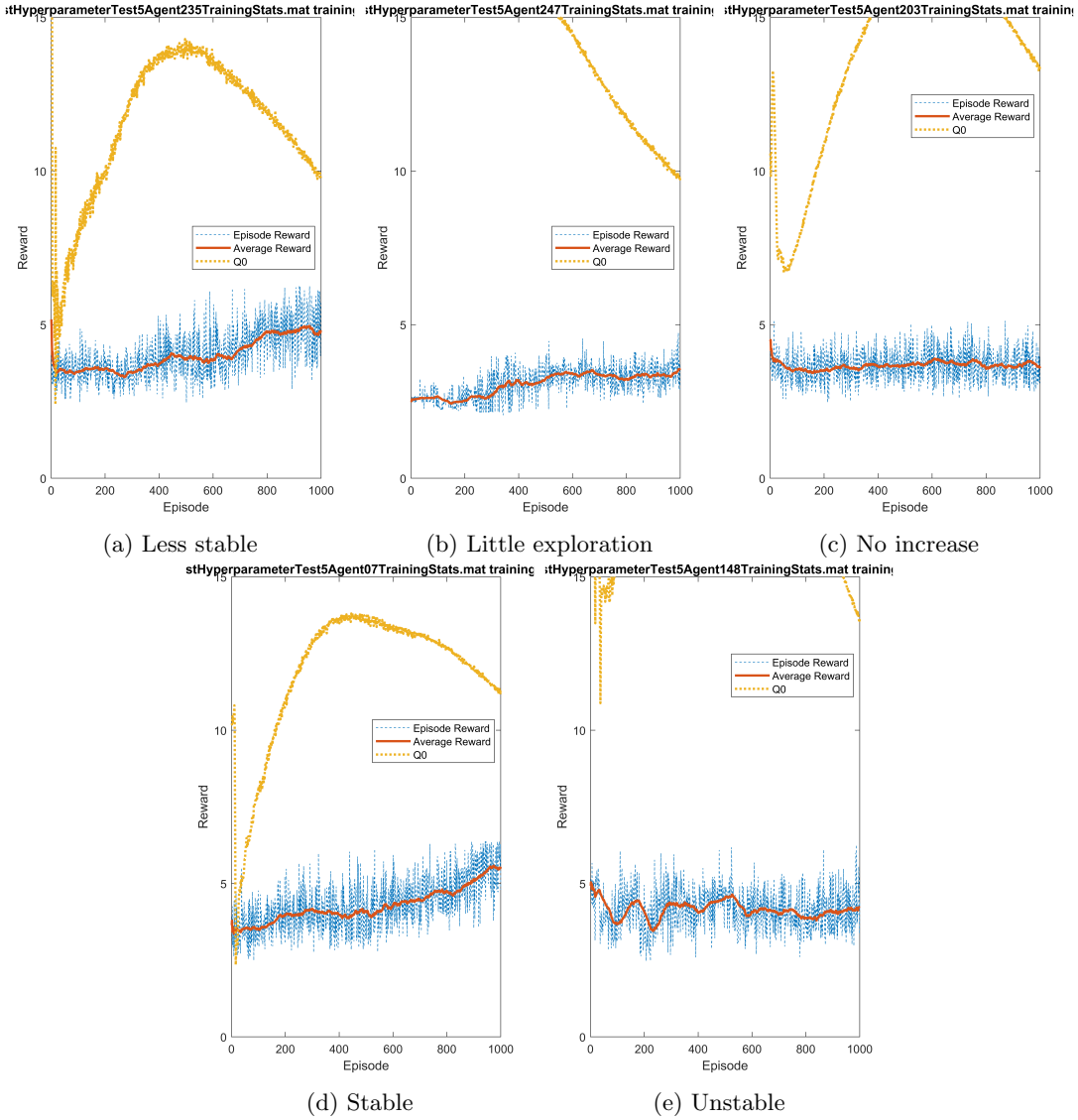
K.5 fifth iteration

Again new settings are defined in Table 70. Due to the converge, now only 324 possible combinations are left. Hence, this is the final iteration that only uses the initialization phase.

LR	DF	ϵ	ϵ_m	ϵ_d	Neurons	Hidden layers
1e-4 1e-5	0.99	0.1 0.5 1	1e-13 1e-7 0.01	5e-4 1e-4	64 128 256	3 to 5

Table 70: Possible values fifth initialization hyperparameter test

For this iteration, again five categories were defined, depicted in Figure 66. The categories are agents that are less stable but have an increasing reward, little exploration, no increase in average reward, stable and and unstable. From the figures it is found that they can be categorized. However, it can be seen that the categories become closer together and less recognizable. Hence, for the next test the full training will be done.



(a) Less stable (b) Little exploration (c) No increase

(d) Stable (e) Unstable

Figure 66: Representation of different categories in test 5

The categorized agents are given in Table 71 and the specific values for the agents are given in Table 53. The agents with a less stable increase in reward have a higher learning rate, with low ϵ, while the agents with little exploration have a low learning rate with a higher ϵ decay rate. Although some

conclusions can be made out of these tests, it is hard to really define what works and what doesn't. A lower epsilon decay is chosen to increase the exploration early on, which is the only change for the next test, while increasing the number of episodes.

Category	**Agents**
Less stable	8 10 14 15 16 18 19 21 22 23 26 36 39 42 44 48 50 51 54 59 67 69 72 74 76 77 81 82 86 89 91 92 94 95 96 97 99 101 102 105 107 114 119 120 121 126 135 137 153 157 166 172 184 187 199 205 212 214 218 221 222 229 230 233 235 238 245 248
Little exploration	9 100 103 106 111 124 127 136 158 183 198 202 224 237 247
No increase	411 12 13 24 28 33 34 35 40 49 52 56 58 60 64 68 70 71 78 90 104 109 110 112 113 116 117 118 122 123 125 128 131 138 139 140 142 143 150 156 159 160 161 165 171 175 177 178 180 182 185 186 190 193 196 201 203 204 210 211 217 225 228 232 236 244 246 249
Stable	3 5 6 7 17 25 27 37 38 41 46 47 65 84 108 132 146 149 151 152 155 168 174 176 179 181 191 200 207 219 223 239
Unstable	1 2 4 20 29 30 31 32 43 45 53 55 57 61 62 63 66 73 75 79 80 83 85 87 88 93 98 115 129 130 133 134 141 144 145 147 148 154 162 163 164 167 169 170 173 188 189 192 194 195 197 206 208 209 213 215 216 220 226 227 231 234 240 241 242 243 250

Table 71: Overview of different categories and agents in test 2

Agent	**LR**	**DF**	ϵ	ϵ_d	ϵ_m	**Neurons**	**HL**	**CT**
1	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	64	4	290.82
2	1.0E-05	0.99	0.5	1.0E-04	1.0E-13	128	3	347.41
3	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	128	4	402.14
4	1.0E-05	0.99	0.1	5.0E-04	1.0E-07	64	5	316.36
5	1.0E-05	0.99	0.5	5.0E-04	1.0E-13	64	5	295.87
6	1.0E-04	0.99	1	5.0E-04	1.0E-13	256	5	737.68
7	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	128	4	390.70
8	1.0E-04	0.99	0.1	5.0E-04	1.0E-13	128	4	370.15
9	1.0E-05	0.99	1	5.0E-04	1.0E-13	128	4	388.46
10	1.0E-04	0.99	0.1	1.0E-04	1.0E-02	256	4	618.11
11	1.0E-04	0.99	0.5	1.0E-04	1.0E-02	64	3	265.26
12	1.0E-05	0.99	1	5.0E-04	1.0E-13	128	3	359.49
13	1.0E-04	0.99	0.5	5.0E-04	1.0E-02	64	4	280.56
14	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	128	3	337.01
15	1.0E-04	0.99	0.1	1.0E-04	1.0E-13	64	5	300.25
16	1.0E-04	0.99	1	5.0E-04	1.0E-02	64	5	306.63
17	1.0E-05	0.99	1	5.0E-04	1.0E-07	256	5	822.30
18	1.0E-04	0.99	0.1	5.0E-04	1.0E-07	256	4	681.41
19	1.0E-05	0.99	0.5	5.0E-04	1.0E-02	256	4	664.38
20	1.0E-04	0.99	0.1	5.0E-04	1.0E-13	64	5	302.16
21	1.0E-04	0.99	0.1	5.0E-04	1.0E-07	256	5	669.17
22	1.0E-05	0.99	0.1	1.0E-04	1.0E-07	64	3	271.67
23	1.0E-04	0.99	0.1	1.0E-04	1.0E-13	256	3	564.23
24	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	64	3	273.33
25	1.0E-04	0.99	0.1	5.0E-04	1.0E-07	128	4	368.29
26	1.0E-04	0.99	1	5.0E-04	1.0E-07	64	5	317.84
27	1.0E-04	0.99	1	1.0E-04	1.0E-13	256	5	799.12
28	1.0E-05	0.99	1	5.0E-04	1.0E-02	128	5	432.37
29	1.0E-04	0.99	0.1	1.0E-04	1.0E-13	128	5	408.58
30	1.0E-04	0.99	1	5.0E-04	1.0E-07	128	4	385.12
31	1.0E-05	0.99	0.5	5.0E-04	1.0E-02	64	3	272.69

32	1.0E-04	0.99	1	5.0E-04	1.0E-13	128	3	339.67
33	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	64	4	287.61
34	1.0E-05	0.99	1	1.0E-04	1.0E-02	64	5	302.24
35	1.0E-05	0.99	1	1.0E-04	1.0E-07	256	3	574.88
36	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	128	3	352.58
37	1.0E-05	0.99	0.5	1.0E-04	1.0E-02	256	4	707.72
38	1.0E-04	0.99	0.5	1.0E-04	1.0E-07	128	3	348.81
39	1.0E-04	0.99	0.5	5.0E-04	1.0E-13	128	5	413.97
40	1.0E-05	0.99	1	1.0E-04	1.0E-13	256	5	838.24
41	1.0E-04	0.99	1	1.0E-04	1.0E-07	256	5	774.17
42	1.0E-04	0.99	0.5	1.0E-04	1.0E-07	128	5	425.73
43	1.0E-04	0.99	0.1	1.0E-04	1.0E-13	256	3	512.74
44	1.0E-04	0.99	0.5	1.0E-04	1.0E-02	64	5	311.96
45	1.0E-05	0.99	0.1	5.0E-04	1.0E-07	128	4	403.07
46	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	64	3	269.30
47	1.0E-04	0.99	0.5	1.0E-04	1.0E-07	256	3	508.27
48	1.0E-04	0.99	1	5.0E-04	1.0E-07	256	5	735.48
49	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	128	4	390.49
50	1.0E-04	0.99	0.1	5.0E-04	1.0E-07	256	3	541.66
51	1.0E-04	0.99	0.1	5.0E-04	1.0E-13	64	5	305.85
52	1.0E-05	0.99	1	5.0E-04	1.0E-02	64	3	271.55
53	1.0E-05	0.99	1	5.0E-04	1.0E-02	64	4	285.67
54	1.0E-04	0.99	1	5.0E-04	1.0E-07	64	3	268.20
55	1.0E-04	0.99	0.5	1.0E-04	1.0E-07	128	3	335.08
56	1.0E-05	0.99	1	1.0E-04	1.0E-02	128	4	386.28
57	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	64	5	310.95
58	1.0E-04	0.99	1	1.0E-04	1.0E-02	256	5	687.75
59	1.0E-04	0.99	0.1	1.0E-04	1.0E-07	128	3	338.23
60	1.0E-05	0.99	0.1	5.0E-04	1.0E-02	128	3	334.25
61	1.0E-05	0.99	0.1	1.0E-04	1.0E-07	64	4	293.99
62	1.0E-04	0.99	0.5	5.0E-04	1.0E-13	128	3	340.43
63	1.0E-05	0.99	1	5.0E-04	1.0E-07	128	3	347.66
64	1.0E-05	0.99	1	1.0E-04	1.0E-07	256	4	683.15
65	1.0E-05	0.99	0.5	1.0E-04	1.0E-13	256	5	875.87
66	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	128	3	343.65
67	1.0E-04	0.99	1	1.0E-04	1.0E-13	64	4	287.20
68	1.0E-04	0.99	1	1.0E-04	1.0E-07	128	3	339.29
69	1.0E-04	0.99	0.5	1.0E-04	1.0E-07	256	4	614.51
70	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	64	3	272.18
71	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	128	3	341.56
72	1.0E-05	0.99	0.5	5.0E-04	1.0E-13	256	4	674.91
73	1.0E-04	0.99	1	1.0E-04	1.0E-13	64	3	265.39
74	1.0E-04	0.99	0.5	1.0E-04	1.0E-07	128	4	374.09
75	1.0E-05	0.99	0.1	5.0E-04	1.0E-13	64	4	288.40
76	1.0E-04	0.99	1	5.0E-04	1.0E-13	128	4	390.59
77	1.0E-04	0.99	1	1.0E-04	1.0E-13	128	5	427.57
78	1.0E-04	0.99	1	1.0E-04	1.0E-13	64	3	263.81
79	1.0E-05	0.99	0.5	5.0E-04	1.0E-02	64	5	304.18
80	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	256	5	845.92
81	1.0E-04	0.99	1	1.0E-04	1.0E-13	128	4	368.25
82	1.0E-04	0.99	0.1	1.0E-04	1.0E-02	256	4	605.80
83	1.0E-04	0.99	1	5.0E-04	1.0E-02	256	4	654.13
84	1.0E-04	0.99	0.5	1.0E-04	1.0E-02	128	5	411.53
85	1.0E-05	0.99	1	5.0E-04	1.0E-13	128	5	437.92
86	1.0E-05	0.99	0.5	5.0E-04	1.0E-13	256	4	634.42
87	1.0E-04	0.99	1	5.0E-04	1.0E-02	128	5	421.68

88	1.0E-04	0.99	0.1	1.0E-04	1.0E-13	128	4	375.97
89	1.0E-04	0.99	1	1.0E-04	1.0E-07	64	5	299.37
90	1.0E-05	0.99	1	1.0E-04	1.0E-07	128	3	342.56
91	1.0E-04	0.99	0.1	5.0E-04	1.0E-07	64	5	303.51
92	1.0E-04	0.99	0.1	1.0E-04	1.0E-07	128	5	390.12
93	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	64	5	363.68
94	1.0E-04	0.99	0.1	5.0E-04	1.0E-07	256	3	594.72
95	1.0E-04	0.99	0.1	5.0E-04	1.0E-02	128	5	463.36
96	1.0E-04	0.99	1	1.0E-04	1.0E-07	64	4	332.20
97	1.0E-04	0.99	0.1	1.0E-04	1.0E-07	128	4	386.22
98	1.0E-05	0.99	0.1	5.0E-04	1.0E-13	128	3	351.53
99	1.0E-04	0.99	0.5	1.0E-04	1.0E-02	256	3	512.41
100	1.0E-04	0.99	0.5	5.0E-04	1.0E-13	64	5	308.23
101	1.0E-04	0.99	0.1	5.0E-04	1.0E-02	64	3	263.67
102	1.0E-04	0.99	0.1	1.0E-04	1.0E-13	256	4	567.09
103	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	128	3	335.02
104	1.0E-05	0.99	1	1.0E-04	1.0E-13	128	4	392.35
105	1.0E-04	0.99	1	5.0E-04	1.0E-13	64	5	304.99
106	1.0E-04	0.99	0.5	5.0E-04	1.0E-02	64	3	271.82
107	1.0E-05	0.99	0.5	5.0E-04	1.0E-02	128	4	380.19
108	1.0E-05	0.99	0.1	1.0E-04	1.0E-07	256	3	593.63
109	1.0E-05	0.99	0.1	5.0E-04	1.0E-13	128	5	418.46
110	1.0E-05	0.99	0.5	1.0E-04	1.0E-02	128	4	398.25
111	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	256	3	590.73
112	1.0E-04	0.99	0.1	5.0E-04	1.0E-13	128	3	341.16
113	1.0E-05	0.99	0.5	1.0E-04	1.0E-13	64	5	301.44
114	1.0E-04	0.99	0.1	1.0E-04	1.0E-07	128	5	444.09
115	1.0E-04	0.99	1	1.0E-04	1.0E-07	128	5	403.53
116	1.0E-05	0.99	0.1	5.0E-04	1.0E-07	256	5	801.86
117	1.0E-05	0.99	0.1	1.0E-04	1.0E-02	128	3	346.21
118	1.0E-04	0.99	0.5	1.0E-04	1.0E-13	64	5	307.96
119	1.0E-05	0.99	0.1	5.0E-04	1.0E-02	256	5	849.45
120	1.0E-04	0.99	0.1	1.0E-04	1.0E-07	128	4	363.33
121	1.0E-04	0.99	0.1	5.0E-04	1.0E-07	128	5	385.05
122	1.0E-05	0.99	0.1	5.0E-04	1.0E-07	256	5	734.79
123	1.0E-05	0.99	0.1	1.0E-04	1.0E-02	256	4	712.00
124	1.0E-05	0.99	1	1.0E-04	1.0E-02	64	5	302.38
125	1.0E-05	0.99	1	1.0E-04	1.0E-07	128	3	345.99
126	1.0E-04	0.99	0.5	5.0E-04	1.0E-02	256	5	711.03
127	1.0E-05	0.99	0.5	5.0E-04	1.0E-02	256	3	567.80
128	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	256	5	825.78
129	1.0E-05	0.99	0.1	5.0E-04	1.0E-02	256	3	575.69
130	1.0E-05	0.99	1	5.0E-04	1.0E-13	128	3	341.65
131	1.0E-05	0.99	0.1	1.0E-04	1.0E-07	64	5	310.66
132	1.0E-04	0.99	0.1	1.0E-04	1.0E-07	256	3	524.57
133	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	128	4	399.25
134	1.0E-05	0.99	0.1	1.0E-04	1.0E-02	256	5	804.56
135	1.0E-04	0.99	0.1	5.0E-04	1.0E-02	256	5	716.08
136	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	128	4	392.85
137	1.0E-04	0.99	0.5	1.0E-04	1.0E-13	256	5	756.97
138	1.0E-05	0.99	0.1	1.0E-04	1.0E-07	64	4	297.10
139	1.0E-05	0.99	1	1.0E-04	1.0E-13	64	4	294.27
140	1.0E-05	0.99	1	1.0E-04	1.0E-02	64	3	267.98
141	1.0E-05	0.99	0.1	5.0E-04	1.0E-07	256	3	559.33
142	1.0E-04	0.99	0.5	5.0E-04	1.0E-02	128	3	333.99
143	1.0E-05	0.99	0.5	1.0E-04	1.0E-13	128	5	469.28

144	1.0E-05	0.99	0.1	5.0E-04	1.0E-07	64	3	321.15
145	1.0E-04	0.99	0.1	1.0E-04	1.0E-13	64	4	337.46
146	1.0E-05	0.99	0.1	5.0E-04	1.0E-02	256	5	815.15
147	1.0E-04	0.99	1	5.0E-04	1.0E-07	64	3	316.63
148	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	256	3	566.66
149	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	256	4	639.13
150	1.0E-04	0.99	1	5.0E-04	1.0E-02	128	3	336.59
151	1.0E-04	0.99	0.1	5.0E-04	1.0E-07	256	3	549.12
152	1.0E-04	0.99	0.5	5.0E-04	1.0E-02	256	3	487.84
153	1.0E-04	0.99	0.5	1.0E-04	1.0E-02	128	4	380.64
154	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	64	4	295.37
155	1.0E-04	0.99	0.1	1.0E-04	1.0E-07	64	5	306.17
156	1.0E-05	0.99	0.1	1.0E-04	1.0E-02	256	4	697.21
157	1.0E-04	0.99	0.5	1.0E-04	1.0E-13	256	3	540.62
158	1.0E-05	0.99	0.5	5.0E-04	1.0E-13	256	5	713.87
159	1.0E-05	0.99	1	5.0E-04	1.0E-07	256	4	680.94
160	1.0E-05	0.99	0.5	1.0E-04	1.0E-02	64	5	302.71
161	1.0E-04	0.99	1	5.0E-04	1.0E-13	64	3	274.16
162	1.0E-05	0.99	1	5.0E-04	1.0E-07	64	3	274.91
163	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	128	4	387.93
164	1.0E-05	0.99	0.1	5.0E-04	1.0E-02	256	4	714.24
165	1.0E-05	0.99	0.1	5.0E-04	1.0E-13	128	5	471.93
166	1.0E-04	0.99	0.5	5.0E-04	1.0E-13	128	4	390.99
167	1.0E-05	0.99	0.5	5.0E-04	1.0E-02	128	5	474.63
168	1.0E-04	0.99	0.1	5.0E-04	1.0E-02	128	3	338.63
169	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	128	4	416.74
170	1.0E-04	0.99	1	5.0E-04	1.0E-02	128	4	379.37
171	1.0E-04	0.99	1	1.0E-04	1.0E-07	64	4	294.01
172	1.0E-04	0.99	1	5.0E-04	1.0E-02	64	3	273.92
173	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	128	3	357.53
174	1.0E-04	0.99	0.5	1.0E-04	1.0E-02	128	3	337.68
175	1.0E-05	0.99	0.1	5.0E-04	1.0E-07	128	3	357.21
176	1.0E-05	0.99	1	1.0E-04	1.0E-07	64	4	288.94
177	1.0E-04	0.99	0.5	1.0E-04	1.0E-13	64	3	272.95
178	1.0E-05	0.99	0.1	5.0E-04	1.0E-02	128	4	375.31
179	1.0E-04	0.99	0.5	1.0E-04	1.0E-13	256	5	726.97
180	1.0E-05	0.99	0.5	1.0E-04	1.0E-02	64	5	310.23
181	1.0E-04	0.99	0.1	5.0E-04	1.0E-02	64	4	294.91
182	1.0E-05	0.99	1	5.0E-04	1.0E-02	64	5	316.39
183	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	64	4	302.59
184	1.0E-04	0.99	0.1	1.0E-04	1.0E-07	64	3	272.39
185	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	128	5	438.87
186	1.0E-04	0.99	1	1.0E-04	1.0E-13	128	3	347.31
187	1.0E-04	0.99	0.5	1.0E-04	1.0E-13	256	5	761.92
188	1.0E-04	0.99	0.5	5.0E-04	1.0E-02	64	4	308.73
189	1.0E-04	0.99	0.1	1.0E-04	1.0E-02	256	5	772.97
190	1.0E-04	0.99	1	1.0E-04	1.0E-02	64	3	270.25
191	1.0E-04	0.99	0.5	1.0E-04	1.0E-07	128	3	352.18
192	1.0E-05	0.99	0.5	5.0E-04	1.0E-02	256	3	558.50
193	1.0E-04	0.99	1	1.0E-04	1.0E-13	64	3	270.36
194	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	256	4	629.06
195	1.0E-04	0.99	1	1.0E-04	1.0E-13	256	3	494.57
196	1.0E-05	0.99	1	1.0E-04	1.0E-02	64	4	299.14
197	1.0E-04	0.99	0.5	5.0E-04	1.0E-02	128	5	436.41
198	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	64	4	313.84
199	1.0E-04	0.99	0.5	5.0E-04	1.0E-13	256	4	640.59

200	1.0E-04	0.99	0.5	1.0E-04	1.0E-02	64	4	331.96
201	1.0E-04	0.99	0.5	5.0E-04	1.0E-13	128	4	417.84
202	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	64	4	301.84
203	1.0E-05	0.99	1	1.0E-04	1.0E-07	128	3	350.52
204	1.0E-04	0.99	1	1.0E-04	1.0E-02	64	5	306.81
205	1.0E-04	0.99	1	1.0E-04	1.0E-07	64	3	267.32
206	1.0E-05	0.99	1	5.0E-04	1.0E-07	256	5	797.46
207	1.0E-04	0.99	0.1	5.0E-04	1.0E-13	256	3	513.48
208	1.0E-05	0.99	0.1	1.0E-04	1.0E-02	128	3	350.67
209	1.0E-04	0.99	0.1	1.0E-04	1.0E-13	256	3	530.47
210	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	128	3	343.56
211	1.0E-04	0.99	0.5	5.0E-04	1.0E-02	128	4	375.72
212	1.0E-04	0.99	0.1	1.0E-04	1.0E-02	256	5	640.01
213	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	64	4	302.16
214	1.0E-05	0.99	0.1	1.0E-04	1.0E-02	256	5	864.52
215	1.0E-05	0.99	1	5.0E-04	1.0E-13	64	4	298.01
216	1.0E-05	0.99	0.5	5.0E-04	1.0E-02	64	4	298.50
217	1.0E-05	0.99	1	5.0E-04	1.0E-13	64	3	272.46
218	1.0E-05	0.99	0.1	5.0E-04	1.0E-02	128	3	349.64
219	1.0E-04	0.99	1	1.0E-04	1.0E-02	256	5	721.89
220	1.0E-05	0.99	1	5.0E-04	1.0E-02	64	5	314.43
221	1.0E-04	0.99	1	1.0E-04	1.0E-02	128	4	378.24
222	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	256	5	768.06
223	1.0E-05	0.99	0.1	1.0E-04	1.0E-13	256	4	708.24
224	1.0E-05	0.99	0.5	5.0E-04	1.0E-07	256	3	587.69
225	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	64	5	309.38
226	1.0E-04	0.99	1	5.0E-04	1.0E-02	64	4	295.31
227	1.0E-05	0.99	0.1	1.0E-04	1.0E-02	128	3	347.92
228	1.0E-05	0.99	0.5	5.0E-04	1.0E-02	128	3	352.06
229	1.0E-05	0.99	0.1	1.0E-04	1.0E-02	256	4	683.27
230	1.0E-04	0.99	1	1.0E-04	1.0E-02	128	4	399.64
231	1.0E-05	0.99	0.5	5.0E-04	1.0E-13	128	4	400.80
232	1.0E-05	0.99	1	5.0E-04	1.0E-02	128	3	362.39
233	1.0E-04	0.99	1	5.0E-04	1.0E-07	128	5	445.40
234	1.0E-05	0.99	0.5	1.0E-04	1.0E-02	64	5	314.67
235	1.0E-04	0.99	0.5	1.0E-04	1.0E-07	128	3	347.39
236	1.0E-05	0.99	1	1.0E-04	1.0E-13	128	5	425.34
237	1.0E-05	0.99	0.1	5.0E-04	1.0E-07	256	5	895.66
238	1.0E-04	0.99	1	1.0E-04	1.0E-07	64	3	271.35
239	1.0E-04	0.99	0.1	1.0E-04	1.0E-02	128	3	347.46
240	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	64	5	312.75
241	1.0E-05	0.99	0.5	5.0E-04	1.0E-13	128	4	381.87
242	1.0E-04	0.99	0.5	5.0E-04	1.0E-07	256	4	668.36
243	1.0E-04	0.99	0.1	5.0E-04	1.0E-02	256	3	549.43
244	1.0E-05	0.99	1	5.0E-04	1.0E-13	256	3	570.20
245	1.0E-04	0.99	1	5.0E-04	1.0E-02	64	3	276.19
246	1.0E-04	0.99	0.1	1.0E-04	1.0E-13	64	3	278.63
247	1.0E-05	0.99	0.1	5.0E-04	1.0E-13	64	3	274.31
248	1.0E-05	0.99	0.1	5.0E-04	1.0E-07	256	4	681.39
249	1.0E-05	0.99	1	1.0E-04	1.0E-07	64	5	309.67
250	1.0E-05	0.99	0.1	1.0E-04	1.0E-02	256	4	700.64

Table 72: All settings for the agents created in test 5

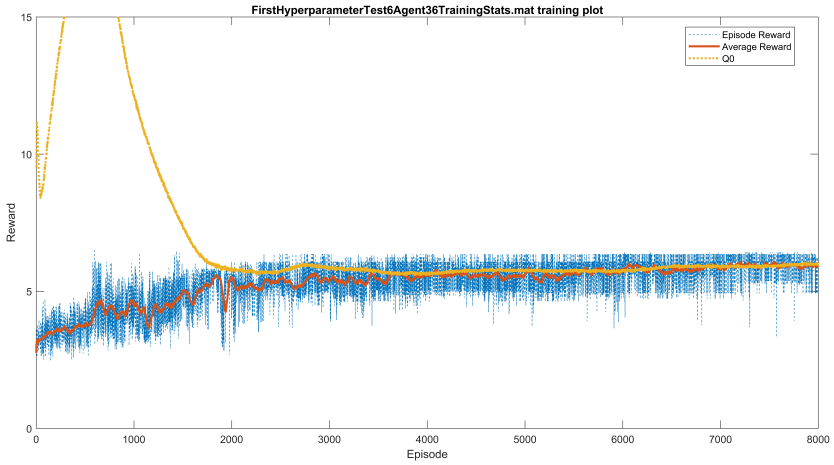
K.6 sixth iteration: full training

Now, the runs will become full training sessions. in Table 73 the possible values are shown. 162 possible combinations can be made, so 40 iterations are chosen to be done to cover 25 percent of the possibilities, while also being able to assess the performance instead of only looking at the initialization phase of the training.

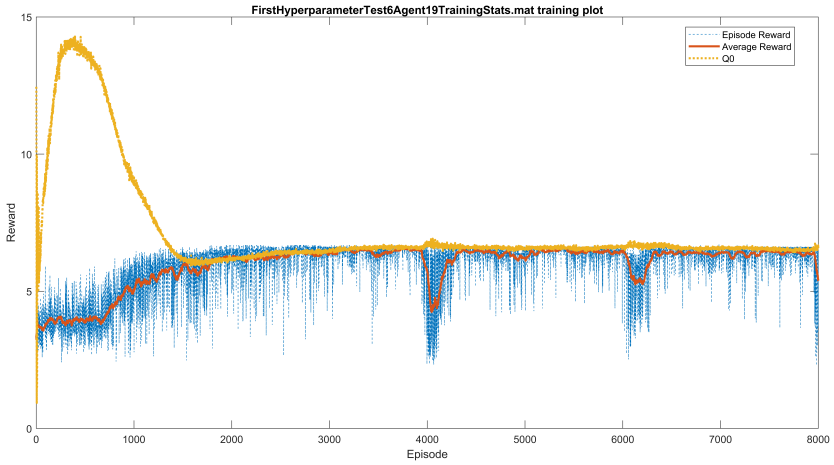
LR	DF	ϵ	ϵ_m	ϵ_d	Neurons	Hidden layers
1e-4 1e-5	0.99	0.5 0.75 1	1e-13 1e-7 0.01	1e-4	64 128 256	3 to 5

Table 73: Possible values sixth initialization hyperparameter test

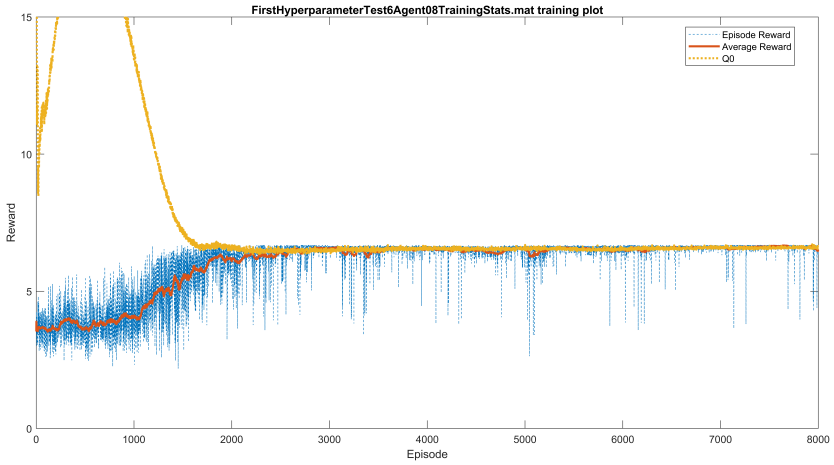
From this, three categories are made from the training results, depicted in Figure 67. The categories being graphs showing exploration at the end of the training episodes, graphs with unstable rewards and finally showing stable training with mainly exploitation at the end of training.



(a) Exploration



(b) Unstable



(c) Exploitation

Figure 67: Representation of different categories in test 6

In Table 74 the category per agent is given, and in Table 75 the agents settings are given. Also, Figure 68 shows a comparison of agents in terms of performance. Here, the stable agents found are 3, 8, 9, 10, 18, 23, and 24 in terms of makespan. These are in all three categories. When combining results from the stable ones in combination with the stable training ones, new settings can be chosen for the seventh and final iteration.

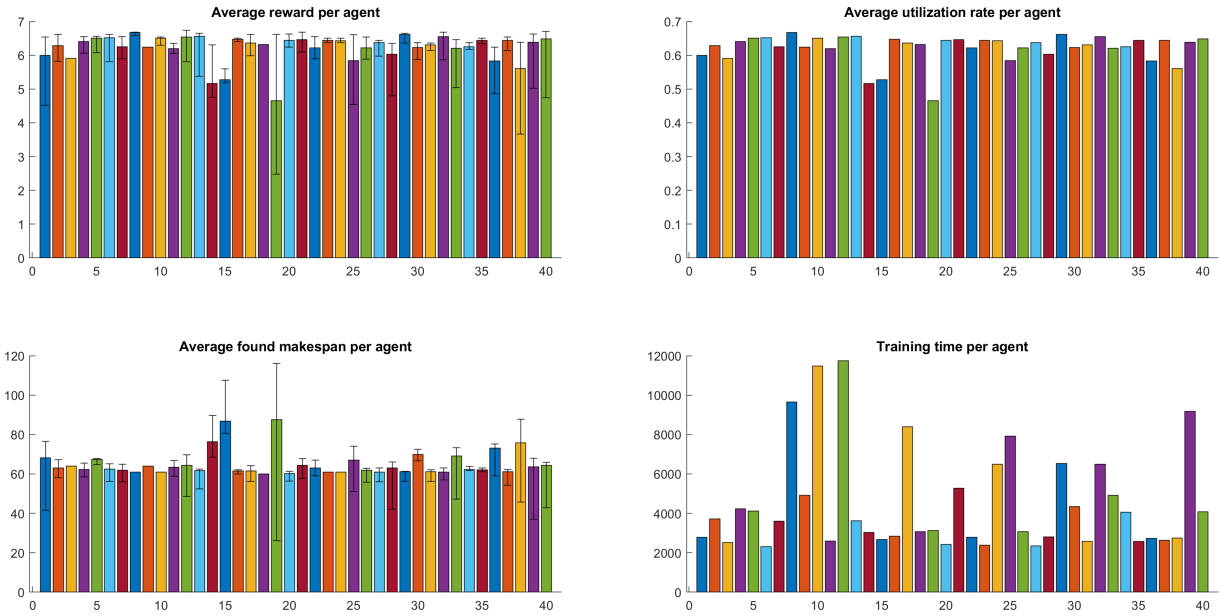


Figure 68: Performance comparison training iteration 6

Category	Agents
Exploration	1 9 11 14 15 17 20 32 36 38 39
Unstable	2 3 4 5 7 10 13 16 18 19 21 22 23 24 25 26 28 29 31 34 35 37
Exploitation	6 8 12 27 30 33 40

Table 74: Overview of different categories and agents in test 6

Agent	LR	DF	ϵ	ϵ_d	ϵ_m	Neurons	HL	CT
1	1.0E-05	0.99	0.5	1.0E-04	1.0E-13	64	4	2778.22
2	1.0E-04	0.99	1	1.0E-04	1.0E-07	128	4	3724.97
3	1.0E-05	0.99	0.5	1.0E-04	1.0E-13	64	3	2526.79
4	1.0E-05	0.99	1	1.0E-04	1.0E-13	128	4	4234.48
5	1.0E-05	0.99	1	1.0E-04	1.0E-02	128	4	4111.20
6	1.0E-04	0.99	0.75	1.0E-04	1.0E-13	64	3	2308.14
7	1.0E-04	0.99	1	1.0E-04	1.0E-07	128	4	3606.20
8	1.0E-05	0.99	1	1.0E-04	1.0E-07	256	4	9646.26
9	1.0E-05	0.99	0.75	1.0E-04	1.0E-02	128	5	4907.46
10	1.0E-05	0.99	0.5	1.0E-04	1.0E-13	256	5	11474.52
11	1.0E-05	0.99	1	1.0E-04	1.0E-13	64	3	2593.60
12	1.0E-05	0.99	0.5	1.0E-04	1.0E-13	256	5	11741.57
13	1.0E-04	0.99	1	1.0E-04	1.0E-02	128	4	3622.98
14	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	64	5	3038.69
15	1.0E-05	0.99	0.75	1.0E-04	1.0E-02	64	3	2668.57
16	1.0E-04	0.99	1	1.0E-04	1.0E-02	64	5	2834.22
17	1.0E-05	0.99	1	1.0E-04	1.0E-07	256	4	8400.16
18	1.0E-05	0.99	0.75	1.0E-04	1.0E-07	64	5	3071.95
19	1.0E-04	0.99	1	1.0E-04	1.0E-02	128	3	3127.29
20	1.0E-05	0.99	1	1.0E-04	1.0E-02	64	3	2421.49
21	1.0E-04	0.99	0.75	1.0E-04	1.0E-02	256	3	5269.88
22	1.0E-04	0.99	0.5	1.0E-04	1.0E-13	64	5	2783.81
23	1.0E-04	0.99	0.75	1.0E-04	1.0E-07	64	3	2381.85
24	1.0E-04	0.99	1	1.0E-04	1.0E-07	256	4	6494.44
25	1.0E-04	0.99	0.75	1.0E-04	1.0E-02	256	5	7922.19
26	1.0E-04	0.99	1	1.0E-04	1.0E-13	64	5	3073.58

27	1.0E-04	0.99	1	1.0E-04	1.0E-02	64	3	2353.94
28	1.0E-05	0.99	0.75	1.0E-04	1.0E-02	64	5	2805.84
29	1.0E-04	0.99	0.75	1.0E-04	1.0E-02	256	4	6533.90
30	1.0E-05	0.99	1	1.0E-04	1.0E-13	128	4	4353.42
31	1.0E-04	0.99	1	1.0E-04	1.0E-13	64	4	2568.41
32	1.0E-05	0.99	0.5	1.0E-04	1.0E-02	256	3	6489.63
33	1.0E-05	0.99	1	1.0E-04	1.0E-13	128	5	4919.18
34	1.0E-04	0.99	0.5	1.0E-04	1.0E-13	128	5	4067.48
35	1.0E-04	0.99	1	1.0E-04	1.0E-13	64	4	2570.96
36	1.0E-05	0.99	0.5	1.0E-04	1.0E-07	64	4	2733.36
37	1.0E-04	0.99	1	1.0E-04	1.0E-07	64	4	2626.14
38	1.0E-05	0.99	1	1.0E-04	1.0E-02	64	4	2740.69
39	1.0E-05	0.99	0.75	1.0E-04	1.0E-02	256	4	9182.09
40	1.0E-05	0.99	0.5	1.0E-04	1.0E-13	128	4	4087.77

Table 75: All settings for the agents created in test 6

K.7 Seventh iteration

In Table 76, the values are given for the final iteration test. The possible combinations given are 36. 20 of the possible combinations are trained through random hyperparameter tuning, to cover more than 50 percent.

LR	DF	ϵ	ϵ_m	ϵ_d	Neurons	Hidden layers
1e-5	0.99	0.75 1	1e-13 1e-7	1e-4	64 128 256	3 to 5

Table 76: Possible values seventh initialization hyperparameter test

In Figure 69 the results based on testing on the trained problem for 50 times per agent is given. It is found that the agents that have a longer computational time also perform better. Based on found rewards, agents 6, 7, 10, 16, 17 and 18 find the highest rewards, with most consistency. All these

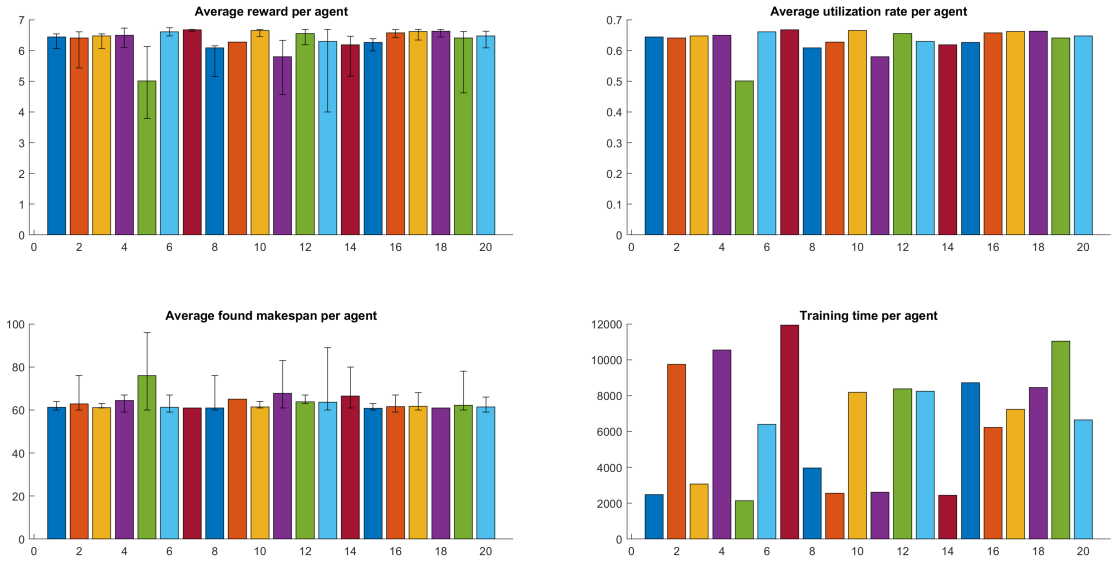


Figure 69: Comparison of performs on trained problem seventh iteration

Now looking at performance on similar, but not identical, problems. We find that there is no agent really able to complete all problems on a significant level. Some agents perform well on some problems, and perform really bad on other problems. Hence, the focus is on using the agent with the best computational time, to be able to compare the agents with not much

The values are defined by agent 6, given in Table 77, and the training graph given in Figure 70.

LR	DF	ϵ	ϵ_m	ϵ_d	Neurons	Hidden layers
1e-5	0.99	0.75	1e-7	1e-4	256	3

Table 77: Determined optimal hyperparameters

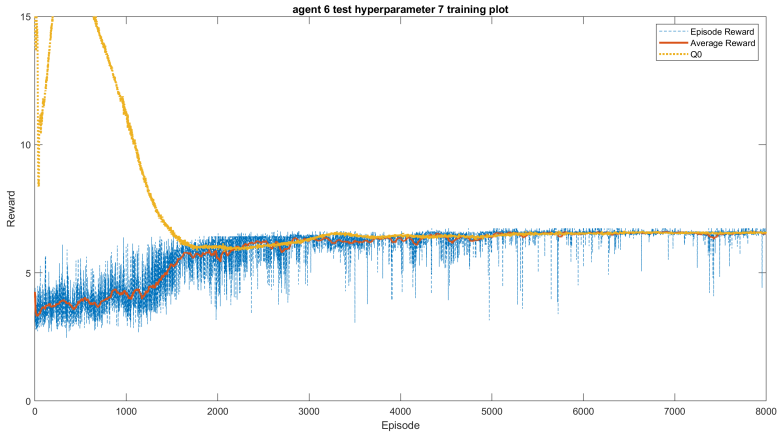


Figure 70: Training graph agent 6 test 7

Agent	**LR**	**DF**	ϵ	ϵ_d	ϵ_m	**Neurons**	**HL**	**CT**
1	1E-05	0.99	0.75	1E-13	1E-04	64	4	2484.51
2	1E-05	0.99	1	1E-07	1E-04	256	5	9739.89
3	1E-05	0.99	0.75	1E-07	1E-04	64	5	3073.82
4	1E-05	0.99	1	1E-13	1E-04	256	5	10543.04
5	1E-05	0.99	1	1E-07	1E-04	64	3	2148.82
6	1E-05	0.99	0.75	1E-07	1E-04	256	3	6394.07
7	1E-05	0.99	0.75	1E-13	1E-04	256	5	11938.24
8	1E-05	0.99	0.75	1E-07	1E-04	128	4	3965.65
9	1E-05	0.99	1	1E-13	1E-04	64	4	2553.69
10	1E-05	0.99	1	1E-07	1E-04	256	4	8186.35
11	1E-05	0.99	0.75	1E-13	1E-04	64	4	2607.94
12	1E-05	0.99	0.75	1E-07	1E-04	256	4	8384.92
13	1E-05	0.99	1	1E-13	1E-04	256	4	8246.96
14	1E-05	0.99	0.75	1E-13	1E-04	64	3	2440.28
15	1E-05	0.99	0.75	1E-07	1E-04	256	4	8712.42
16	1E-05	0.99	1	1E-07	1E-04	256	3	6230.01
17	1E-05	0.99	0.75	1E-13	1E-04	256	3	7232.91
18	1E-05	0.99	0.75	1E-13	1E-04	256	4	8444.50
19	1E-05	0.99	0.75	1E-07	1E-04	256	5	11034.15
20	1E-05	0.99	1	1E-13	1E-04	256	3	6654.72

Table 78: All settings for the agents created in test 7

Agent	Agent 6		
metric	**Reward**	**Makespan**	U_t
UB	6.625664	59	0.662566
LB	6.625664	59	0.662566
Mean	6.625664	59	0.662566
Var	0	0	0
StDev	0	0	0

Table 79: statistical results agent 6 iteration 7

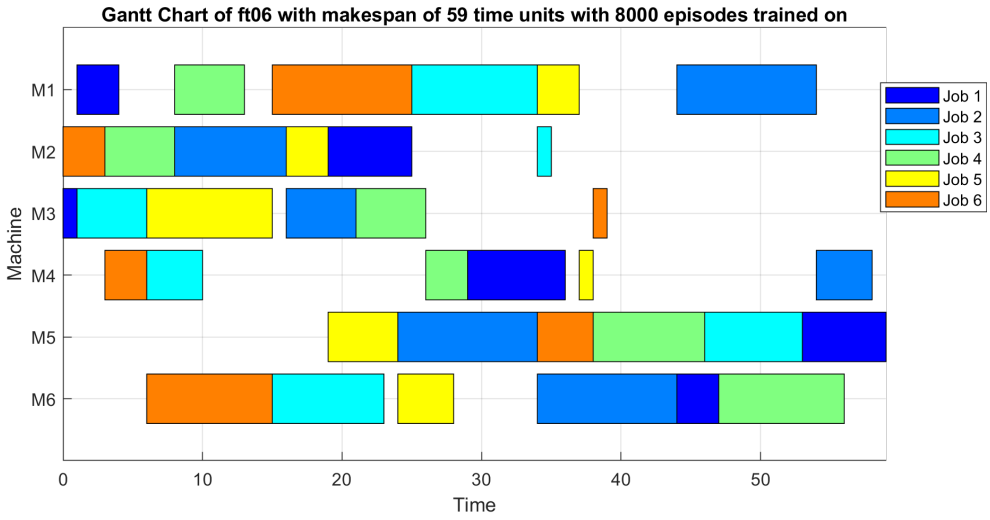


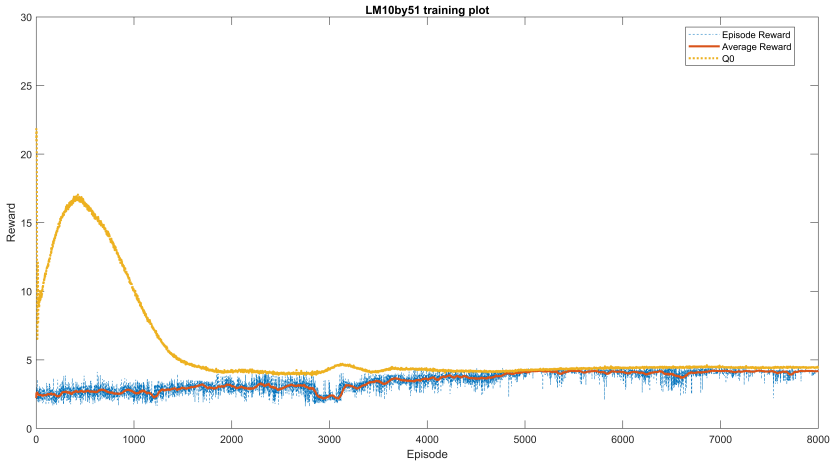
Figure 71: Gantt chart of solution found by agent 6 of seventh iteration

L Training on multiple problems for static job shop

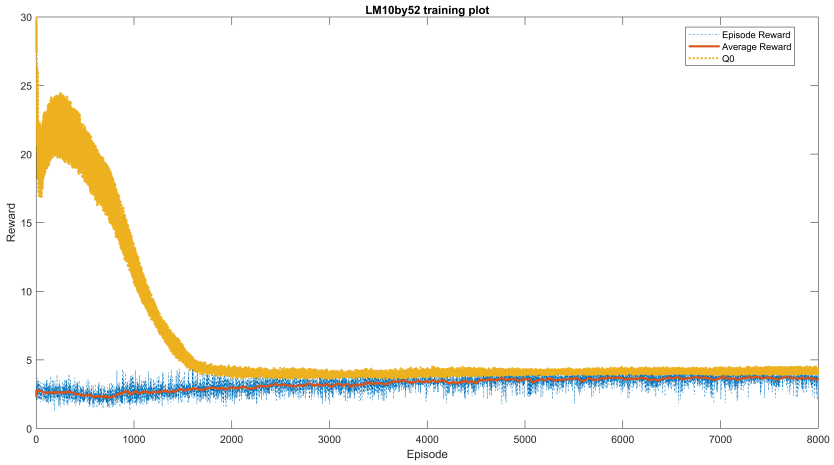
In this section, the tests on multiple problems are set out. First, agents have been trained, using the found hyperparameters from the section before, to train on problems with 10, 15 and 20 problems, all with 5 jobs. Here, five different agents are created per number of machines, where an agent is trained on 1, 5, 10, 25 or 50 different problems. For example, if an agent is trained on 5 problems, the order would be problem 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, etc. etc. Following the findings from these tests, the same tests are done on problems with 6 jobs and 6 machines, similar to problem ft06 where the hyperparameters where found for. These tests are done to assess what the capabilities are, and if a general solver can be created using this method to solve multiple problems on a considerable level. After training, each agent is tested on the problems they are trained on, problems they have not been trained on but with the same number of machines, jobs and processing time distribution, and finally tested on problems with a different processing time distribution.

L.1 Test on 10, 15 and 20 machine problems

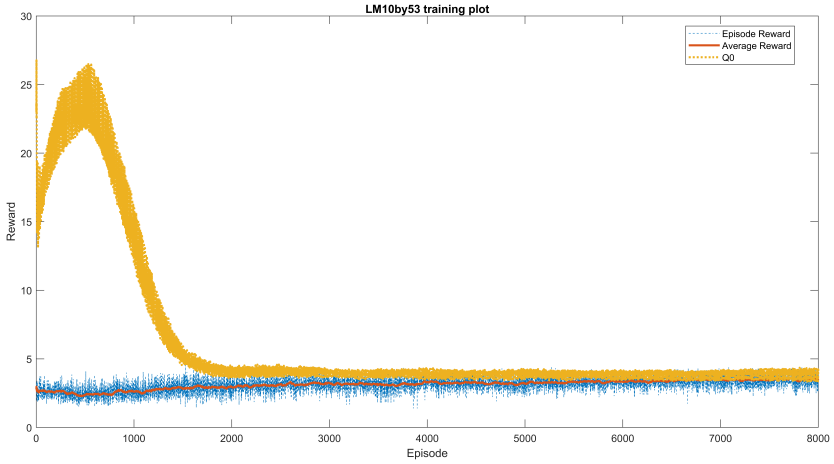
First, the agents created for 10 machine problems are assessed. The training graphs for each agent is given in Figure 72 and 73. Here, it can be seen that the Q0 becomes higher the more problems an agent is trained on, as well as an increase in oscillation. This is to be expected as each problem has its own highest and lowest possible reward to gain.



(a) 1 problem

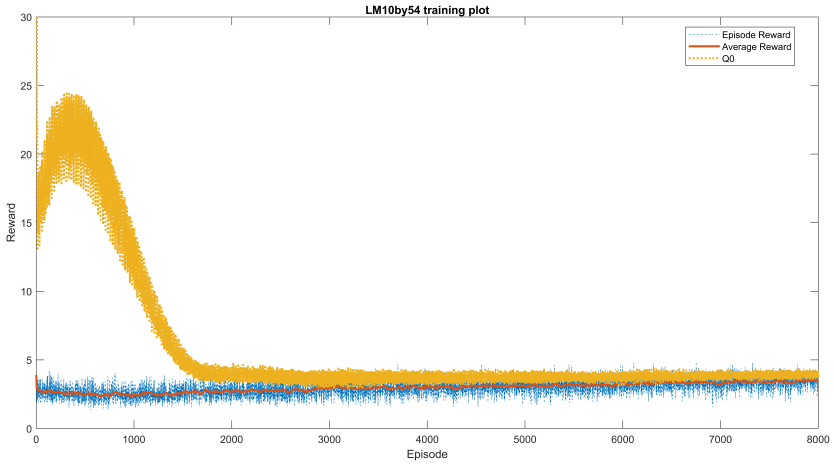


(b) 5 problems

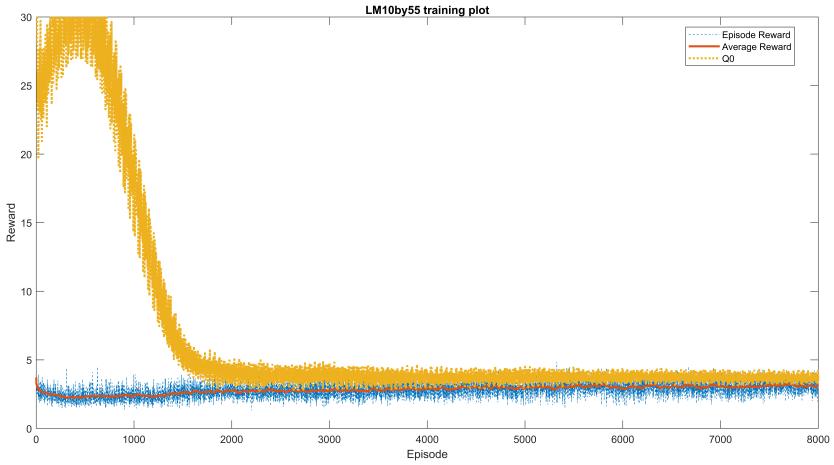


(c) 10 problem

Figure 72: Different training behaviors for agents trained on 10 machines by 5 jobs problems



(a) 25 problems



(b) 50 problems

Figure 73: Different training behaviors for agents trained on 10 machines by 5 jobs problems

The rewards for the problems the agents are trained on are given in Table 80. It is found that the agent trained on 25 different problems, is able to perform the best overall for the agents. However, the agent specifically trained on a single problem, being instance 1, is able to outperform the dispatching rules.

instance	LB	LRPT	HRPT	LPT	HPT	HTPT	LTPT	FIFO	LIFO	LOR	MOR	A1	A2	A3	A4	A5
1	**64**	240	74	141	202	240	240	217	218	217	86	**72**	87	91	102	156
2	**66**	211	73	159	205	250	211	220	206	220	**72**	164	126	90	81	120
3	**68**	252	92	145	223	226	252	272	292	272	**86**	113	105	109	108	165
4	**55**	247	**79**	150	179	223	247	221	214	221	83	173	86	**79**	96	123
5	**64**	210	**83**	185	179	198	210	228	193	228	91	120	109	112	109	127
6	**72**	214	117	245	170	239	214	212	238	212	**92**	180	100	107	105	123
7	**67**	200	**78**	185	206	223	200	220	197	220	89	213	156	110	90	114
8	**57**	200	**78**	155	185	198	200	202	182	202	80	166	200	88	86	110
9	**49**	196	**68**	145	162	198	196	193	186	193	72	104	134	73	73	74
10	**68**	278	99	169	217	258	278	243	272	243	**83**	221	141	107	100	141
11	**68**	227	90	202	169	241	227	217	252	217	**86**	226	105	103	95	162
12	**67**	224	121	197	165	229	224	266	251	266	**93**	191	159	139	93	110
13	**67**	250	81	177	210	252	250	247	227	247	**76**	134	118	173	114	134
14	**68**	221	**74**	153	197	217	221	225	222	225	83	147	129	129	103	122
15	**63**	266	100	200	211	241	266	263	224	263	83	105	158	116	103	134
16	**66**	224	86	194	183	220	224	223	206	223	71	198	87	107	85	174
17	**68**	269	97	217	189	266	269	265	240	265	97	118	225	126	123	111
18	**70**	235	92	192	218	242	235	203	233	203	80	111	152	132	120	132
19	**65**	261	107	186	216	248	261	220	228	220	97	181	207	123	105	129
20	**68**	232	81	180	151	224	232	248	243	248	92	193	160	112	100	147
21	**66**	263	93	203	212	245	263	254	239	254	98	230	174	109	113	135
22	**61**	220	86	160	182	229	220	217	221	217	80	113	182	87	81	154
23	**63**	216	93	132	177	200	216	153	170	153	92	200	121	137	**85**	133
24	**64**	207	91	138	179	211	207	216	189	216	84	154	153	121	90	108
25	**61**	198	81	185	167	210	198	199	220	199	79	142	129	101	85	105
26	**71**	239	111	149	182	229	239	206	192	206	87	138	208	129	109	125
27	**78**	247	111	253	170	254	247	261	260	261	96	117	183	121	118	148
28	**66**	221	84	155	202	207	221	235	191	235	86	158	**81**	87	87	87
29	**64**	209	77	213	138	233	209	233	224	233	77	108	145	94	115	117
30	**63**	200	79	191	142	206	200	234	254	234	77	199	120	147	101	121
31	**62**	228	84	170	182	249	228	233	193	233	77	212	130	107	115	94
32	**63**	205	97	200	191	212	205	228	234	228	95	166	136	105	104	132
33	**68**	239	**79**	163	158	271	239	259	233	259	82	230	151	194	97	93
34	**67**	224	84	160	155	214	224	213	229	213	**83**	117	224	116	134	106
35	**65**	230	77	156	211	232	230	237	227	237	**73**	111	230	100	82	101
36	**66**	224	**81**	218	149	266	224	260	210	260	93	230	108	106	115	133
37	**60**	230	85	156	157	213	230	208	223	208	**77**	106	123	127	97	100
38	**59**	216	**74**	161	191	193	216	213	221	213	78	95	154	138	109	152
39	**59**	216	84	208	147	219	216	226	221	226	**75**	202	109	108	125	101
40	**68**	228	**81**	220	184	214	228	253	243	253	85	184	147	132	94	123
41	**63**	187	87	178	158	209	187	204	198	204	**78**	108	126	100	90	150
42	**64**	196	**79**	189	170	229	196	219	222	219	90	156	170	140	102	116
43	**72**	269	97	237	192	224	269	276	220	276	95	186	269	116	**92**	132
44	**62**	152	77	172	145	200	152	158	181	158	**75**	121	92	106	89	160
45	**72**	242	94	167	228	220	242	250	227	250	**87**	132	228	116	102	120
46	**76**	253	94	165	197	263	253	236	228	236	**86**	198	137	137	126	103
47	**62**	253	**102**	171	202	253	253	243	265	243	103	109	133	112	120	146
48	**70**	220	105	166	205	235	220	239	241	239	**84**	129	150	183	107	148
49	**61**	217	**77**	147	172	215	217	211	210	211	79	139	175	148	121	99
50	**75**	225	100	147	205	258	225	253	269	253	**79**	217	252	113	139	92
Average	**65.42**	**226.62**	**88.28**	**178.14**	**183.74**	**228.92**	**226.62**	**228.64**	**223.58**	**228.64**	**84.44**	**156.74**	**149.68**	**117.26**	**102.7**	**124.84**

Table 80: Results for agents and dispatching rules on trained instances

Now, the agents and dispatching rules are tested on problems that they are not trained on, but have the same machines, jobs and distribution of processing times to assess if problems in the scope changes the performance or not. These results are given in Table 81. Here, it is found that the results are similar in performance for both the actions as well as the agents, in comparison to the problems the agents were trained on.

instance	LB	LRPT	HRPT	LPT	HPT	HTPT	LTPT	FIFO	LIFO	LOR	MOR	A1	A2	A3	A4	A5
1	**57**	235	73	166	186	213	235	240	214	240	81	198	227	85	119	97
2	**61**	196	93	134	196	184	196	220	219	220	94	86	120	104	92	115
3	**67**	238	93	190	200	230	238	221	243	221	85	204	164	131	113	120
4	**61**	210	88	106	159	217	210	210	206	210	73	197	134	101	90	94
5	**71**	249	104	115	202	246	249	255	236	255	100	200	154	134	115	171
6	**70**	238	90	160	188	208	238	224	236	224	74	158	**72**	133	117	89
7	**72**	231	86	220	188	213	231	235	227	235	87	117	191	148	91	93
8	**70**	258	109	222	185	241	258	242	256	242	106	137	184	182	152	161
9	**70**	259	95	147	196	184	259	236	244	236	101	141	173	139	103	120
10	**48**	163	72	134	132	171	163	169	141	169	84	90	93	78	84	89
11	**65**	253	84	157	217	241	253	268	244	268	76	89	133	234	85	120
12	**61**	240	85	155	223	234	240	269	241	269	104	129	129	113	143	120
13	**70**	265	121	157	179	222	265	233	249	233	93	166	157	140	168	171
14	**67**	217	95	153	183	220	217	228	231	228	91	143	175	148	119	145
15	**72**	252	92	161	182	241	252	238	224	238	85	202	144	126	101	134
16	**72**	227	88	192	195	238	227	242	246	242	81	183	94	126	98	162
17	**60**	193	94	182	183	202	193	226	204	226	83	211	105	122	94	108
18	**76**	234	104	186	193	211	234	253	237	253	93	161	171	171	**92**	187
19	**72**	245	88	203	189	245	245	252	212	252	84	106	150	167	140	153
20	**59**	229	75	165	202	242	229	222	236	222	78	158	109	123	100	109
21	**63**	228	90	164	153	196	228	245	202	245	83	219	171	104	84	126
22	**64**	206	84	131	153	196	206	186	209	186	92	164	134	118	117	134
23	**59**	232	73	139	179	206	232	221	201	221	83	158	121	107	81	124
24	**60**	235	**77**	131	158	196	235	208	203	208	80	141	130	123	**77**	89
25	**64**	240	86	129	185	211	240	201	242	201	76	98	125	99	105	108
26	**59**	207	71	173	182	206	207	221	237	221	88	112	136	91	99	147
27	**68**	242	85	152	182	216	242	227	223	227	83	141	102	135	121	93
28	**61**	186	80	159	181	213	186	215	209	215	72	165	103	83	79	80
29	**61**	191	88	197	165	185	191	204	179	204	82	137	99	123	93	138
30	**58**	171	100	174	124	161	171	173	158	173	93	159	140	150	111	110
31	**67**	193	91	190	146	195	193	206	203	206	85	112	107	100	109	105
32	**67**	242	92	178	242	291	242	260	280	260	93	94	135	136	117	161
33	**65**	219	90	203	183	261	219	277	253	277	85	105	134	121	128	158
34	**60**	230	94	131	222	239	230	239	229	239	83	90	143	129	114	160
35	**63**	201	84	156	177	219	201	228	203	228	90	134	99	123	114	94
36	**64**	236	93	149	188	239	236	228	238	228	98	98	116	**80**	100	134
37	**78**	228	90	187	198	222	228	232	228	232	88	184	163	136	95	110
38	**60**	230	84	168	184	248	230	241	214	241	72	115	108	138	101	107
39	**65**	258	76	164	229	238	258	223	246	223	83	185	250	83	128	100
40	**64**	218	93	182	174	226	218	213	183	213	86	171	159	128	119	88
41	**66**	208	76	142	185	211	208	231	236	231	81	163	106	127	95	120
42	**62**	168	78	158	137	198	168	210	194	210	**74**	190	180	174	**74**	124
43	**60**	199	96	138	147	194	199	216	215	216	92	105	150	116	129	111
44	**63**	220	83	188	197	234	220	221	225	221	88	84	202	87	117	84
45	**61**	241	86	205	170	250	241	241	250	241	96	107	178	235	114	122
46	**65**	219	93	171	197	246	219	229	201	229	96	133	109	99	97	127
47	**56**	229	71	178	139	194	229	211	200	211	70	149	115	99	87	86
48	**67**	230	81	209	209	212	230	206	233	206	80	141	121	102	88	122
49	**61**	193	77	132	152	240	193	236	200	236	78	162	124	103	78	79
50	**61**	189	72	136	175	181	189	187	188	187	77	135	169	110	112	116
Average	**64.26**	**222.42**	**87.26**	**164.38**	**181.82**	**218.54**	**222.42**	**226.38**	**220.56**	**226.38**	**85.6**	**144.54**	**140.16**	**125.28**	**105.98**	**120.3**

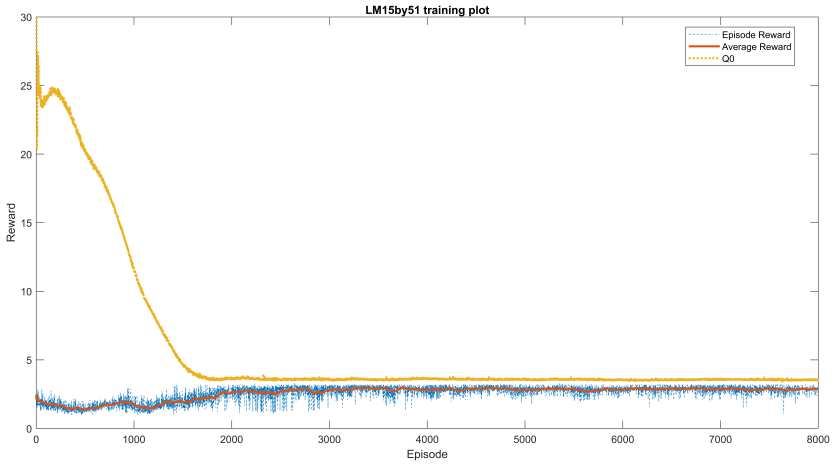
Table 81: Results for agents and dispatching rules on instances with same jobs, machines and processing times

Finally the agents are tested on problems where the processing time distribution is changed from U[1,10] to U[20, 99] to assess how such a change in observations will influence the performance. The results are given in Table 82. Here, the performance of the agent with 50 problems trained on performs better than the agent with 25 problems. It is assumed that this is due to the number of different problems seen, being able to generalize better for different types of problems. It is also seen that the agents overall performance is greatly reduced, while the agent with 50 problems trained on is able to perform slightly better. The same tests are done on the agents trained on 15 and 20 machine problems.

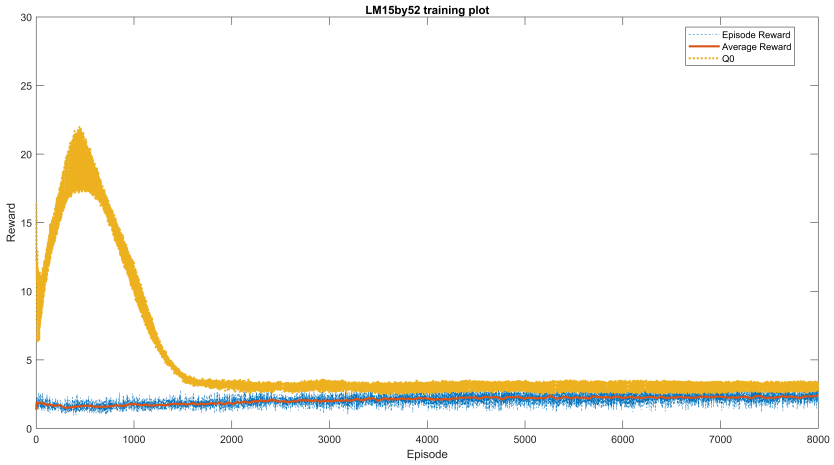
instance	LB	LRPT	HRPT	LPT	HPT	HTPT	LTPT	FIFO	LIFO	LOR	MOR	A1	A2	A3	A4	A5
1	**673**	2633	780	1657	2261	2594	2633	2352	2369	2352	906	**1028**	1931	1639	2339	1055
2	**675**	2297	901	1881	1915	2682	2297	2386	2211	2386	769	2400	2159	1901	2060	1092
3	**686**	2650	958	1687	2517	2387	2650	2877	3082	2877	881	1361	1872	2042	1803	1362
4	**607**	2456	809	1673	2124	2380	2456	2464	2331	2464	895	1906	1909	**2283**	2253	1093
5	**664**	2269	745	2000	1939	2105	2269	2416	2098	2416	939	1568	1527	1458	2000	858
6	**739**	2363	959	2363	1616	2602	2363	2319	2628	2319	927	1037	1643	2265	2203	1373
7	**700**	2216	818	2013	2009	2421	2216	2412	2189	2412	882	1452	2010	1760	1922	1506
8	**627**	2179	886	1607	1645	2151	2179	2200	1998	2200	825	1915	1769	1348	1764	1296
9	**540**	2179	759	1900	1467	2210	2179	2162	2105	2162	780	1999	1864	1444	1844	971
10	**695**	2947	950	1845	2345	2719	2947	2575	2869	2575	855	2054	2290	2448	2106	971
11	**709**	2453	912	2063	1684	2554	2453	2323	2711	2323	875	1920	1680	1947	2146	955
12	**682**	2265	1222	2039	2314	2309	2265	2796	2634	2796	956	1680	2143	1580	1890	1374
13	**696**	2627	801	2183	2079	2653	2627	2616	2362	2616	801	1818	1700	2271	2348	932
14	**708**	2437	725	2070	1794	2365	2437	2473	2399	2473	857	1467	2079	1826	1823	824
15	**664**	2790	1061	1899	2055	2619	2790	2776	2348	2776	860	1687	1493	2030	1956	1031
16	**667**	2406	823	2074	2091	2348	2406	2426	2227	2426	752	1885	1783	1912	2329	823
17	**693**	2798	1007	1916	1872	2805	2798	2795	2529	2795	999	2374	1510	1695	1958	1017
18	**718**	2360	961	1957	2113	2582	2360	2154	2492	2154	807	1369	1441	2363	1942	1808
19	**681**	2641	1120	1929	2224	2484	2641	2292	2427	2292	972	2196	1976	1751	1768	1344
20	**700**	2428	877	1945	1839	2346	2428	2623	2562	2623	933	2236	2239	2510	2106	1057
21	**690**	2762	1012	2120	2460	2537	2762	2640	2515	2640	1003	2730	1947	2102	2174	976
22	**641**	2364	890	1871	2109	2471	2364	2336	2350	2336	825	1968	1845	1987	1645	1427
23	**663**	2396	1045	1400	2014	2221	2396	1696	1888	1696	963	1035	1384	1726	1711	1252
24	**651**	2270	939	1516	1976	2313	2270	2295	2039	2295	834	1399	1593	2200	1868	1030
25	**646**	2165	834	1705	1744	2295	2165	2162	2383	2162	846	1742	1598	1922	1806	902
26	**728**	2591	1088	1835	1667	2606	2591	2172	2097	2172	892	2056	1619	1926	2100	1306
27	**776**	2556	1190	2509	1781	2711	2556	2731	2700	2731	953	2201	2651	2420	2052	1336
28	**696**	2354	945	1687	2197	2297	2354	2487	2066	2487	892	1174	1758	2487	2242	960
29	**667**	2540	786	2220	1418	2659	2540	2500	2413	2500	825	1956	1474	2264	2032	856
30	**664**	2164	813	2435	1866	2290	2164	2536	2720	2536	785	2254	2084	1906	1578	813
31	**648**	2422	802	1613	1865	2371	2422	2560	2095	2560	803	1650	1573	2426	1970	900
32	**659**	2293	1021	1916	1756	2332	2293	2496	2502	2496	955	1501	1320	2419	2294	1085
33	**700**	2514	848	1562	1727	2895	2514	2748	2465	2748	831	1423	2134	2201	2187	1048
34	**682**	2265	877	1762	1606	2589	2265	2293	2444	2293	882	2062	2265	1594	1575	1336
35	**678**	2489	780	1241	2087	2420	2489	2558	2460	2558	748	2377	1866	2055	1938	1181
36	**680**	2400	824	2150	1752	2862	2400	2746	2269	2746	948	2163	2400	1981	1519	1025
37	**632**	2425	867	1717	1799	2204	2425	2230	2414	2230	812	1433	1836	2270	1750	867
38	**618**	2323	786	1729	1763	2092	2323	2265	2399	2265	817	1245	1991	2246	1615	1019
39	**630**	2340	779	2283	1789	2409	2340	2464	2392	2464	775	1944	1609	1579	1584	1092
40	**706**	2465	802	1937	1858	2335	2465	2716	2617	2716	889	2239	2141	1861	1896	971
41	**648**	2020	910	2150	2130	2285	2020	2261	2155	2261	785	1840	1647	1928	1933	1216
42	**670**	2120	872	1913	2039	2461	2120	2341	2401	2341	943	2069	1793	1823	1782	1227
43	**735**	2760	836	1982	2134	2803	2760	2914	2328	2914	945	1584	2341	2317	**1682**	929
44	**647**	1709	817	1729	1584	2185	1709	1793	1998	1793	760	1359	1553	1507	2091	915
45	**723**	2575	995	2001	2396	2559	2575	2621	2417	2621	880	1955	1527	1753	1741	1329
46	**764**	2678	1009	1710	1970	2824	2678	2531	2442	2531	867	1765	2606	1809	2247	1199
47	**654**	2926	986	1561	2288	2671	2926	2580	2786	2580	1062	1347	1919	2146	2671	1136
48	**711**	2343	925	1805	2128	2503	2343	2519	2556	2519	847	1465	2099	2266	2168	1281
49	**644**	2271	896	1454	1617	2423	2271	2323	2309	2323	842	1036	2038	2427	1876	1113
50	**745**	2589	951	1602	1857	2697	2589	2688	2863	2688	827	1413	1875	1726	1797	1212
Average	**678.4**	**2429.66**	**903.98**	**1876.32**	**1945.6**	**2472.72**	**2429.66**	**2452.78**	**2401.08**	**2452.78**	**870.14**	**1754.74**	**1870.08**	**1994.94**	**1961.68**	**1113.62**

Table 82: Results for agents and dispatching rules on instances with same jobs, machines with a processing time distribution of U[20, 99]

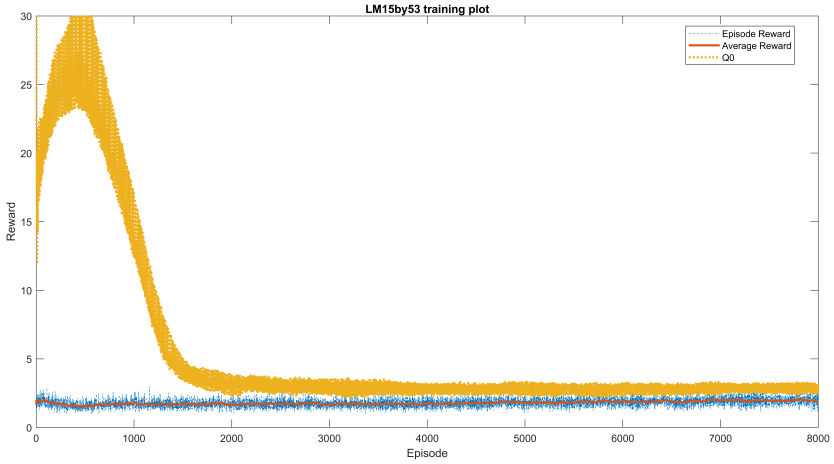
For the other agents, the same analysis will be done. In Figure 74 and 75 as well as Figure 76 and 77 the training graphs are given of the agents trained on 15 and 20 machines respectively. Here, it is seen that the more machines, the bigger the difference between the Q0 and found average reward. Hence, it is assumed that even though no changes in architecture of the neural network are done, the increase in steps and different states still needs different hyperparameters to perform optimally.



(a) 1 problem

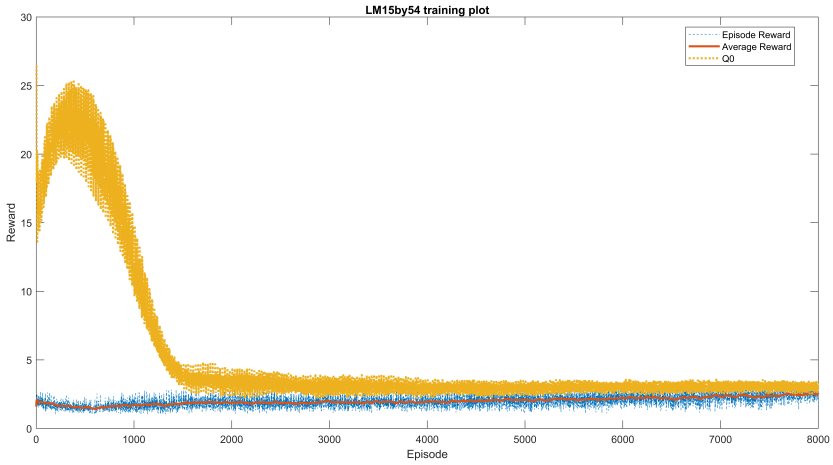


(b) 5 problems

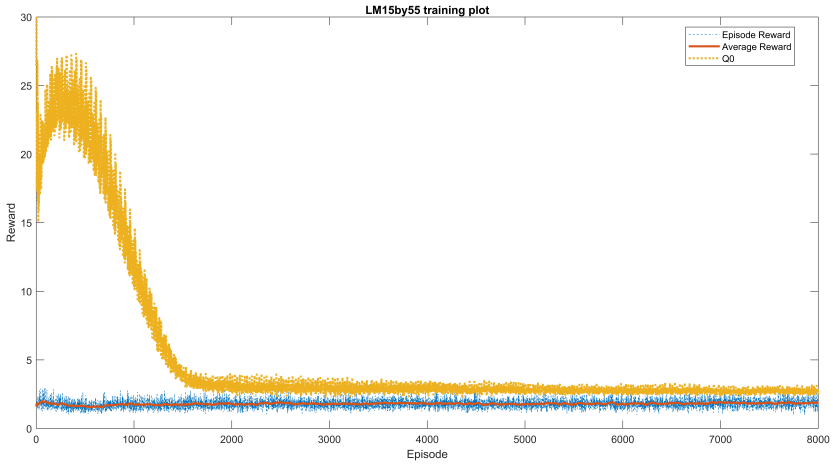


(c) 10 problem

Figure 74: Different training behaviors for agents trained on 15 machines by 5 jobs problems

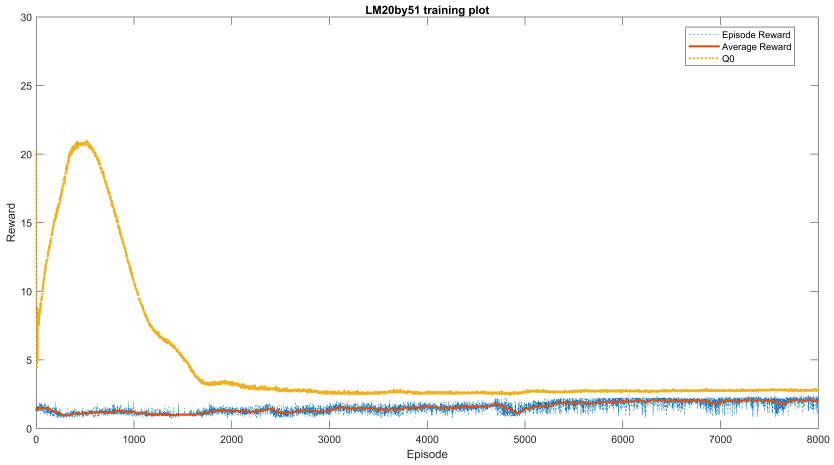


(a) 25 problems

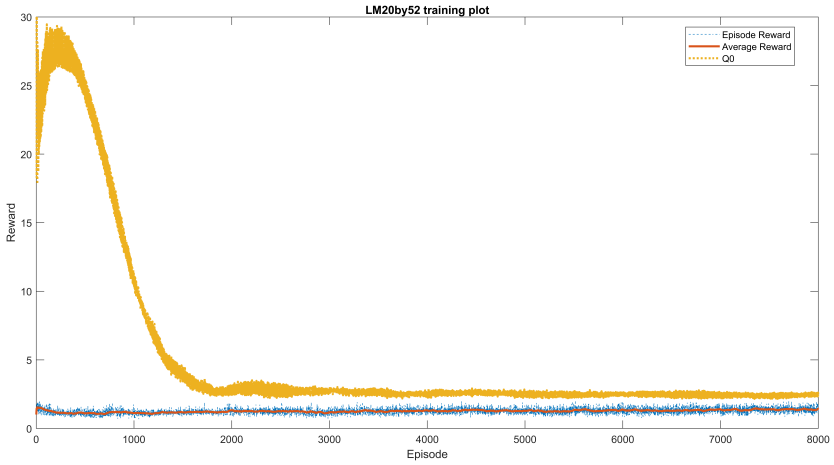


(b) 50 problems

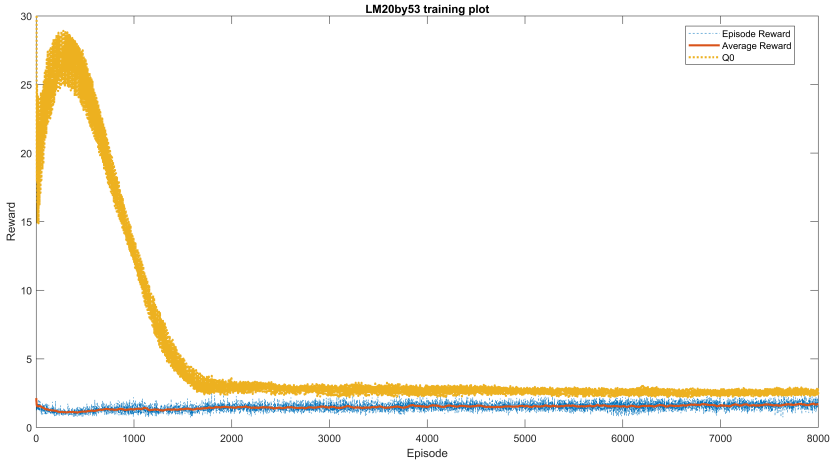
Figure 75: Different training behaviors for agents trained on 15 machines by 5 jobs problems



(a) 1 problem

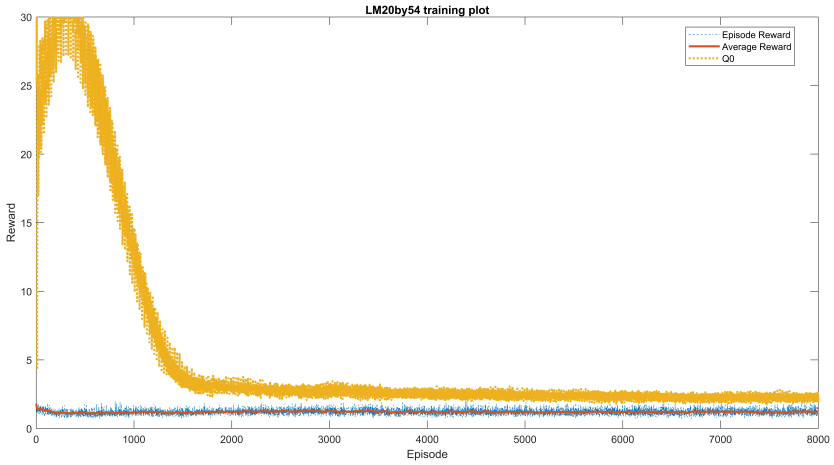


(b) 5 problems

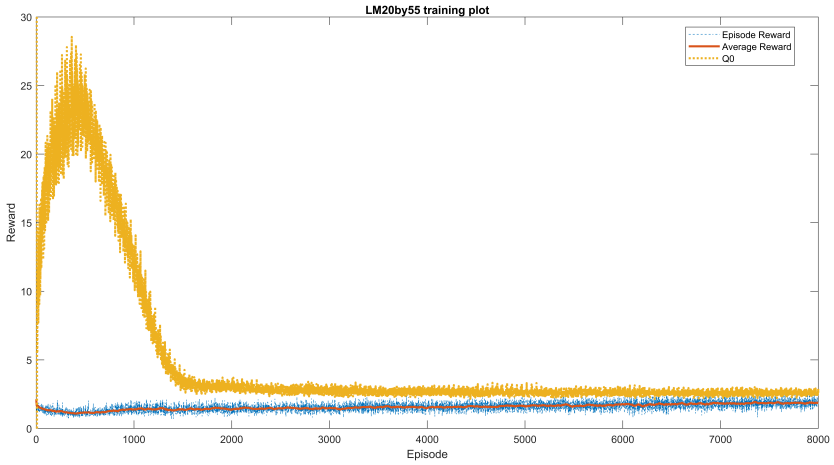


(c) 10 problem

Figure 76: Different training behaviors for agents trained on 20 machines by 5 jobs problems



(a) 25 problems



(b) 50 problems

Figure 77: Different training behaviors for agents trained on 20 machines by 5 jobs problems

In Table 83 the comparison of performance of the agents in terms of found makespan compared to the found lower bound of the problems is shown. From this table it can be found that like the 10 machine problems, the 15 machine problems show the same results in terms of performance. For the problems with 20 machines, the performance is better for the agent trained on 50 problems. When looking at the problems with a changed time distribution, the agents trained on 20 machine problems, show real different results in terms of performance in comparison to 10 and 15 machines. This could be because of the hyperparameters not being compatible to ensure correct learning behavior or just being randomly well suited neural networks for the agent to solve such problems.

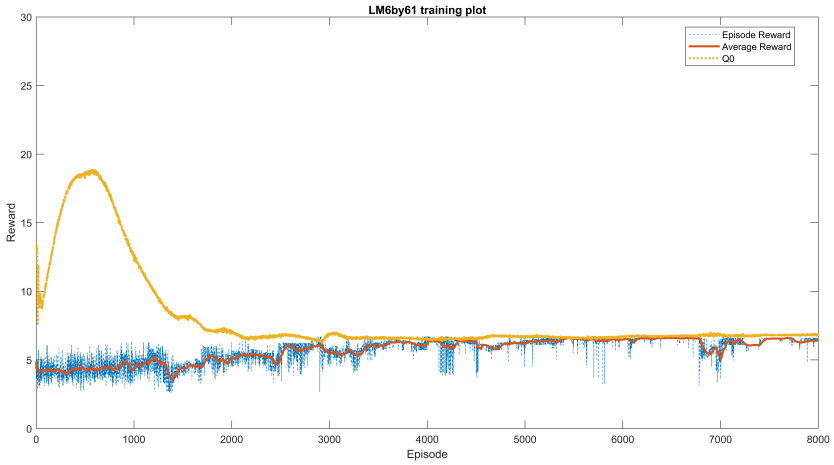
	10 by 5			15 by 5			20 by 5		
solver	**trained**	**untrained**	**U[20,99]**	**trained**	**untrained**	**U[20,99]**	**trained**	**untrained**	**U[20,99]**
LRPT	-246%	-246%	-258%	-262%	-256%	-274%	-268%	-271%	-281%
HRPT	-35%	-36%	-33%	-25%	-26%	-23%	-18%	-20%	-17%
LPT	-172%	-156%	-177%	-186%	-185%	-185%	-197%	-198%	-186%
HPT	-181%	-183%	-187%	-201%	-202%	-201%	-212%	-215%	-201%
HTPT	-250%	-240%	-264%	-267%	-260%	-276%	-268%	-278%	-278%
LTPT	-246%	-246%	-258%	-262%	-256%	-274%	-268%	-271%	-281%
FIFO	-249%	-252%	-262%	-262%	-259%	-271%	-272%	-275%	-283%
LIFO	-242%	-243%	-254%	-266%	-256%	-275%	-270%	-276%	-281%
LOR	-249%	-252%	-262%	-262%	-259%	-271%	-272%	-275%	-283%
MOR	-29%	-33%	-28%	-24%	-25%	-22%	-18%	-21%	-16%
1 problem	-140%	-125%	-159%	-97%	-97%	-151%	-171%	-174%	-23%
5 problems	-129%	-118%	-176%	-113%	-112%	-147%	-154%	-157%	-213%
10 problems	-79%	-95%	-194%	-123%	-129%	-124%	-86%	-102%	-190%
25 problems	-57%	-65%	-189%	-55%	-57%	-237%	-145%	-152%	-26%
50 problems	-91%	-87%	-64%	-121%	-117%	-140%	-71%	-73%	-80%

Table 83: Overview of performance per dispatching rule and agents, in comparison to the found lower bound

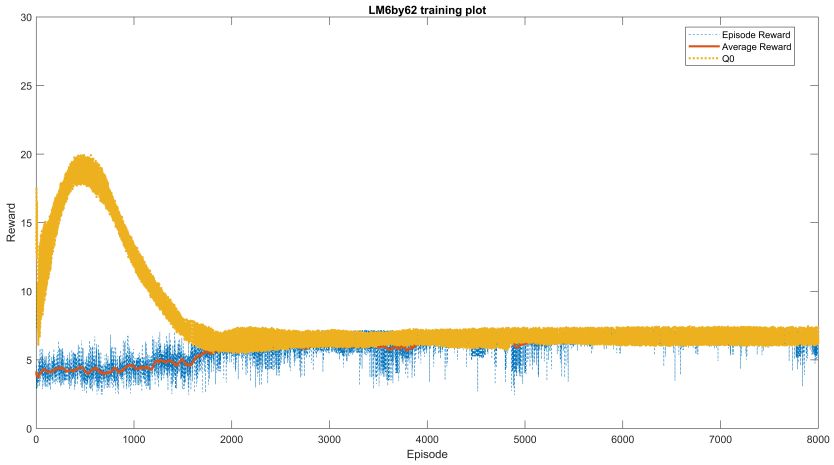
As the training graphs show that the bigger the problem becomes, the less compatible the hyperparameters are, another test is done for agents trained on problems of 6 jobs and 6 machines, similar to the ft06 problem to better assess the performance.

L.2 Test on 6 by 6 problems

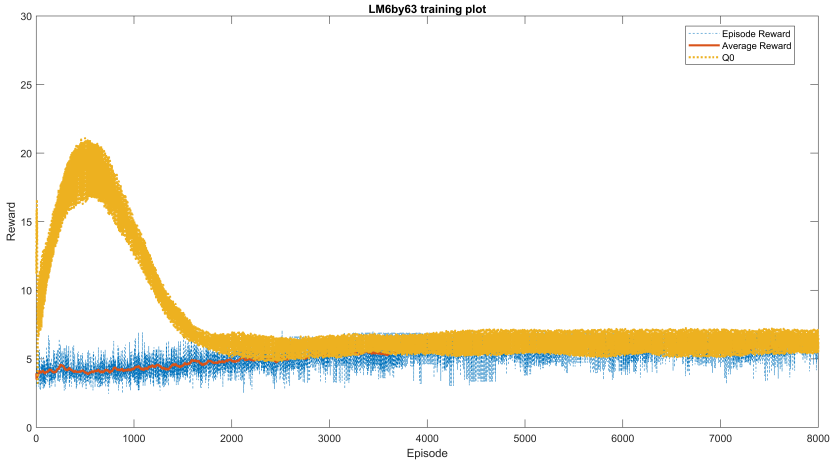
As it was found that from these tests the hyperparameters do not transfer optimally if the problems to solve themselves change, although not changing the size of the inputs, the agents are trained on problems of 6 machines and 6 jobs as the hyperparameters were found for such a problem. In Figures 78 and 79 the different training graphs are shown. Here, it can be seen that these hyperparameters are more suited to solve these kind of problems as expected.



(a) 1 problem

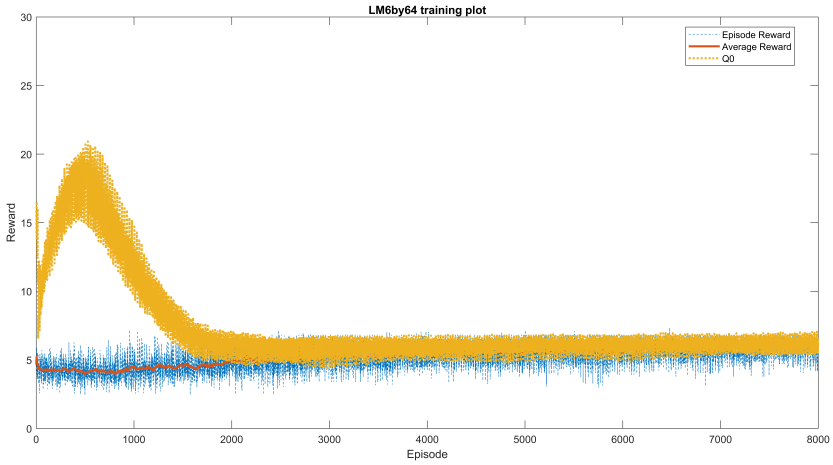


(b) 5 problems

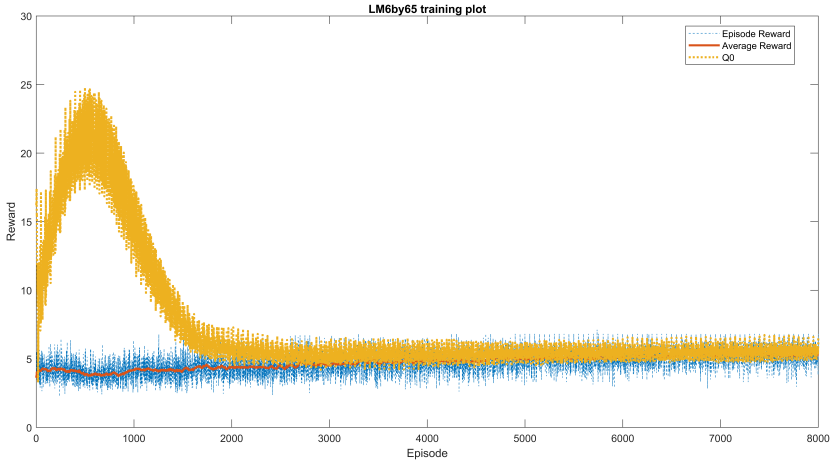


(c) 10 problems

Figure 78: Different training behaviors for agents trained on 6 machines by 6 jobs problems



(a) 25 problems



(b) 50 problem

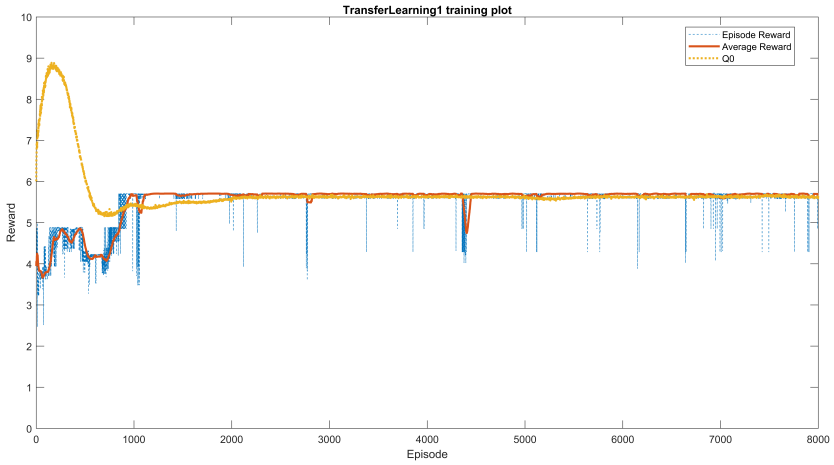
Figure 79: Different training behaviors for agents trained on 6 machines by 6 jobs problems

M Transfer Learning for static job shops

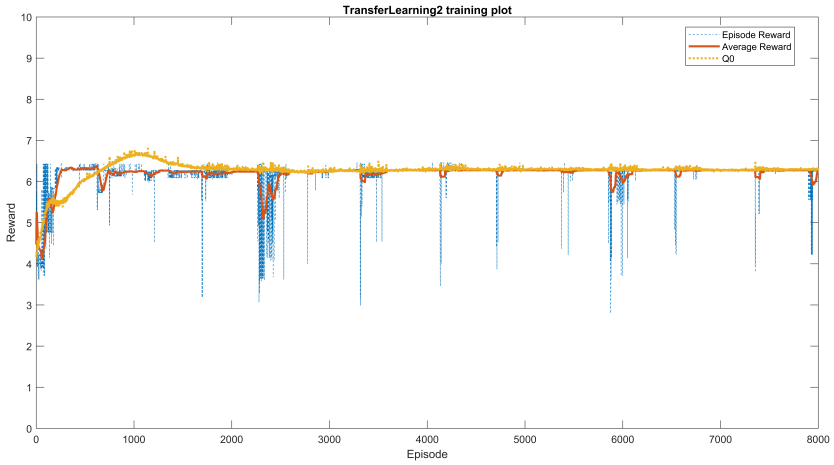
As now there are pre-trained agents, these agents might be able to perform better than using creating an agent from scratch and training it on such a problem. Hence, the pre-trained agents are defined as the agents that are trained on the multiple problems. These are tested again on training with 8000 episodes. Here, three different approaches are taken, being instantly deploying without changing any parameters, only emptying the experience buffer and with the changing of hyperparameters as well as emptying the experience buffer to assess differences between these methods.

M.1 Transfer learning without changing parameters

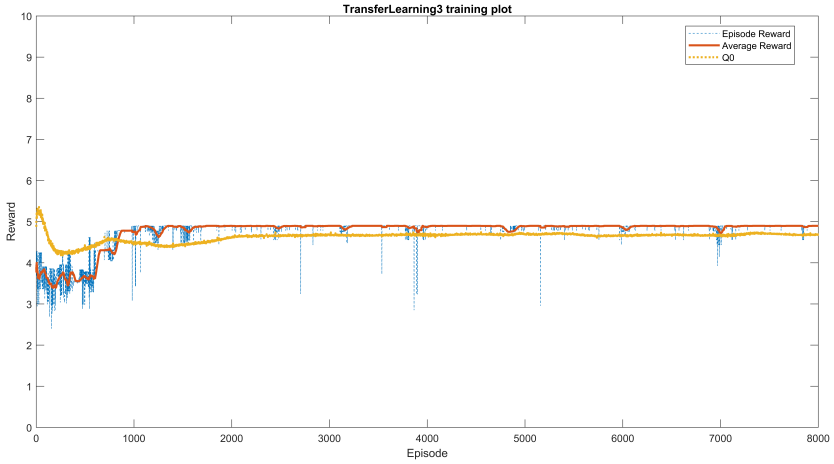
First, the pre-trained agents are deployed directly without changing any settings. Hence, the epsilon is already decayed, the experience buffer is full and other hyperparameters might be changed. In Figures 80 and 81 the training graphs are given. As expected, these training graphs show little exploration as the epsilon value has already decayed. A difference can be seen at the speed of learning an optimal policy as well as the initial estimate Q0. Agent 1, trained on a single and different problem, shows the biggest Q0 thus initially being able to estimate the reward the worse. However, it also shows that it is able to learn an optimal policy the earliest, at around 2000 episodes. The higher the number of problems trained on, the later it seems that the agent learns an optimal policy, with agent 5 in Figure 81b being around 7000 episodes.



(a) 1 problem

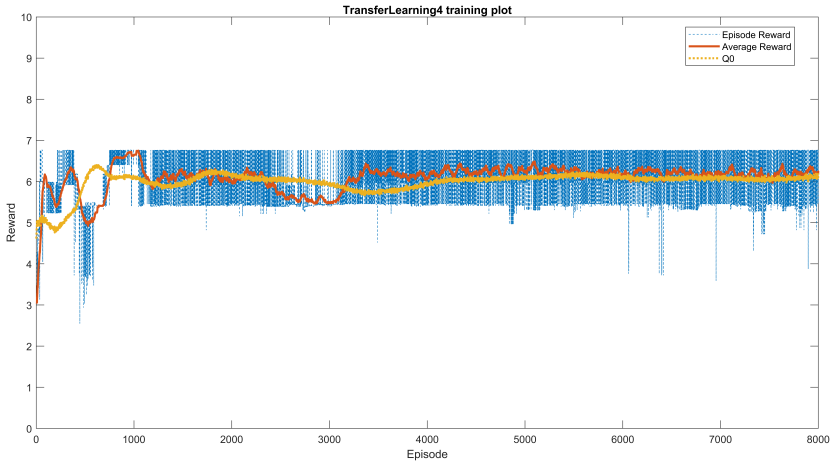


(b) 5 problems

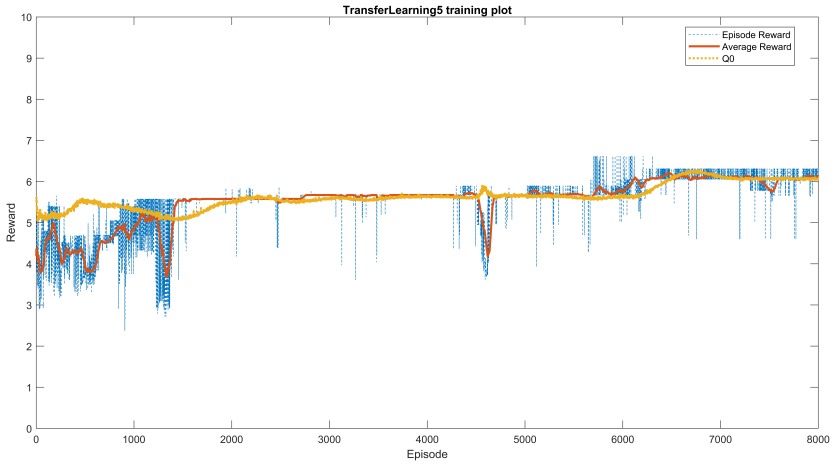


(c) 10 problems

Figure 80: Training graphs of transfer learning agents



(a) 25 problems



(b) 50 problems

Figure 81: Training graphs of transfer learning agents

Looking at the value of the reward, the third agent seems to find a reward around 5, and agent 1, 2 and 5 around 6. Agent 4 seems to find really high rewards around 7 with also rewards which are much lower, around 5. To assess the actual performance, in Table 84 the comparison of each agent is given, tested on the specific problem ft06. Hence it is clear that agent 4 performs best in terms of rewards, which was also the agent able to perform the best in general terms, found in Appendix L.2.

Agent	**N problems**	U_t	**Makespan**	**Reward**
1	1	0.599	60	5.99
2	5	0.624	60	6.24
3	10	0.595	65	5.95
4	25	0.676	60	6.76
5	50	0.631	60	6.31

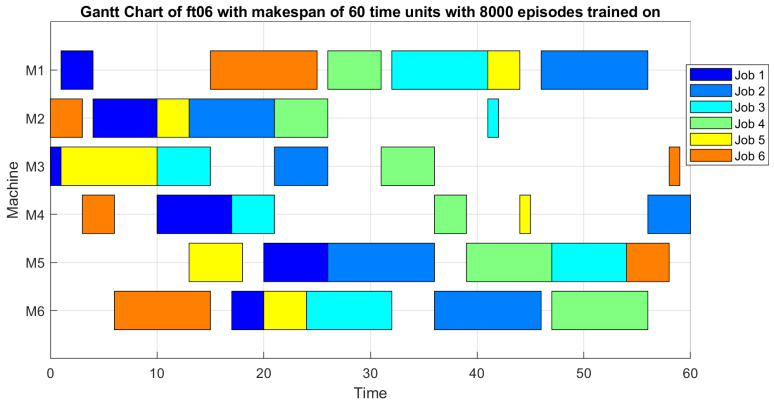
Table 84: Results of transfer learning agents trained on ft06

Finally, looking at the statistics found from testing the agents on ft06 for 50 times, given in Table 85 (where R is the reward, C the makespan and U_t the average utilization rate), it is found that all agents have learned an optimal policy, as each of them consistently finds the same reward and the variance and standard deviation are really small, thus showing consistency in found rewards, utilization rate and makespan. Hence the results show that each agent has found an optimal policy.

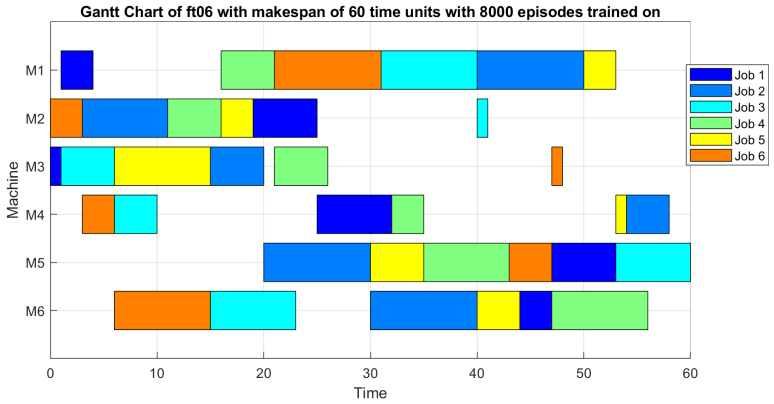
Agent	Agent 1		Agent 2		Agent 3		Agent 4		Agent 5	
metric	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**
UB	5.71	66	6.24	60	4.90	76	6.76	60	6.31	60
LB	5.71	66	6.24	60	4.90	76	6.76	60	6.31	60
Mean	5.71	66	6.24	60	4.90	76	6.76	60	6.315	60
Var	0.00E+00	0	0	0	8.05E-31	0	3.22E-30	0	8.05E-31	0
StDev	0.00E+00	0	0.00E+00	0	8.97E-16	0	1.79E-15	0	8.97E-16	

Table 85: Statistical results of 50 runs on ft06 from transfer learning agents

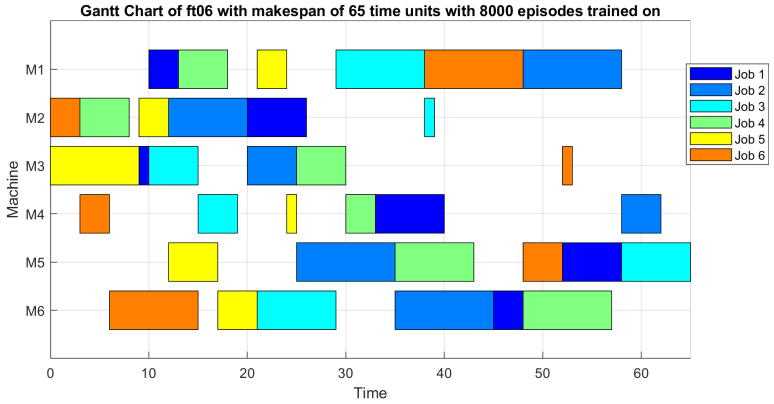
To finally compare the learning, in Figures **??** and 83 Gantt charts for each agents found solution are given. From these figures it can be seen that although equal in found makespan, the agents do find different methods to solve the problem.



(a) 1 problem

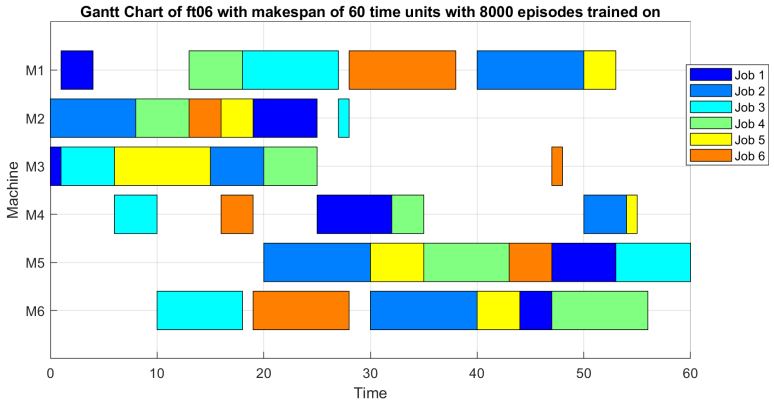


(b) 5 problems

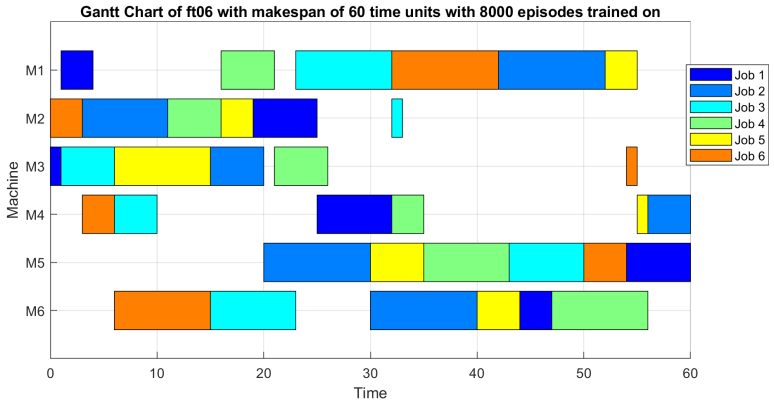


(c) 10 problems

Figure 82: Gantt charts of transfer learning agents



(a) 25 problems

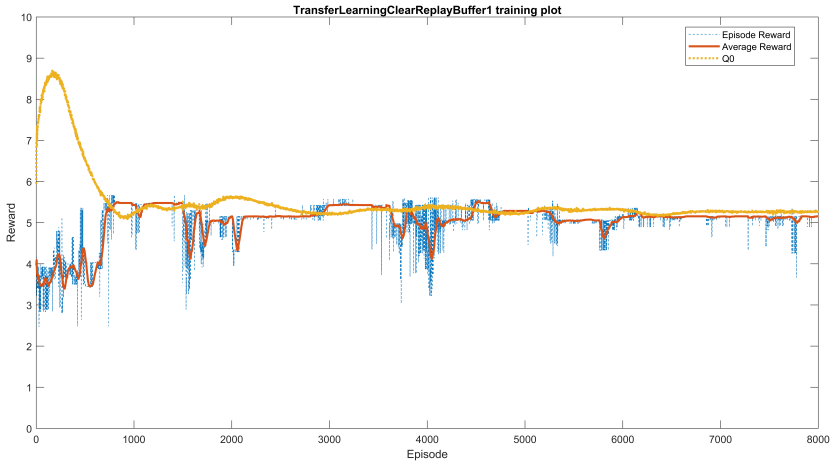


(b) 50 problems

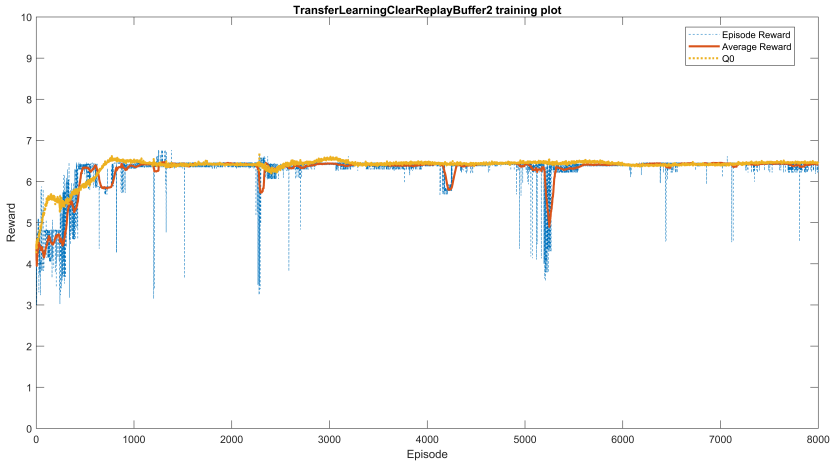
Figure 83: Gantt charts of transfer learning agents

M.2 Transfer learning with emptying experience buffer

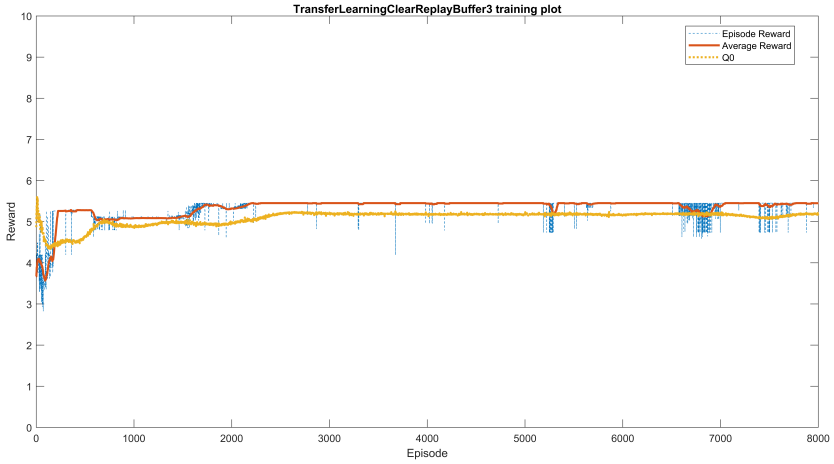
To assess what changes when the experience buffer is emptied, thus only learning from new experiences gained instead of old experiences, again the 5 pre-trained agents are retrained on the ft06 problem. In Figures **??** and 85 the different training graphs per agent are shown. The first 4 agents show similar behavior found in the earlier test, only the fifth agent showing that it is less able to learn.



(a) 1 problem

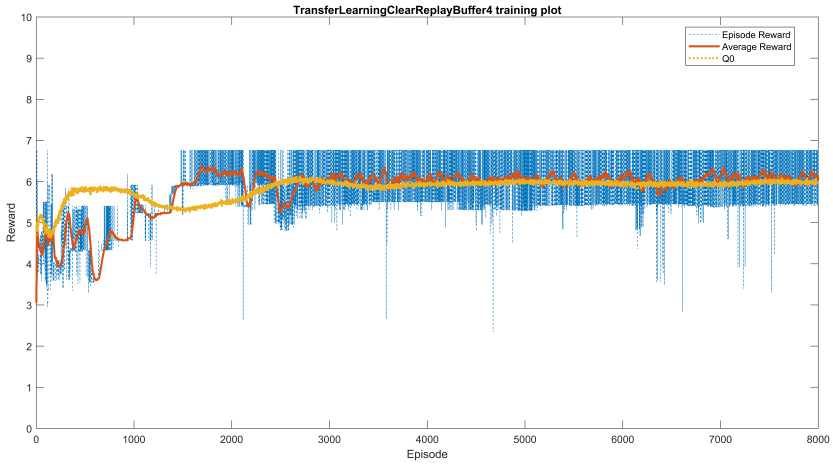


(b) 5 problems

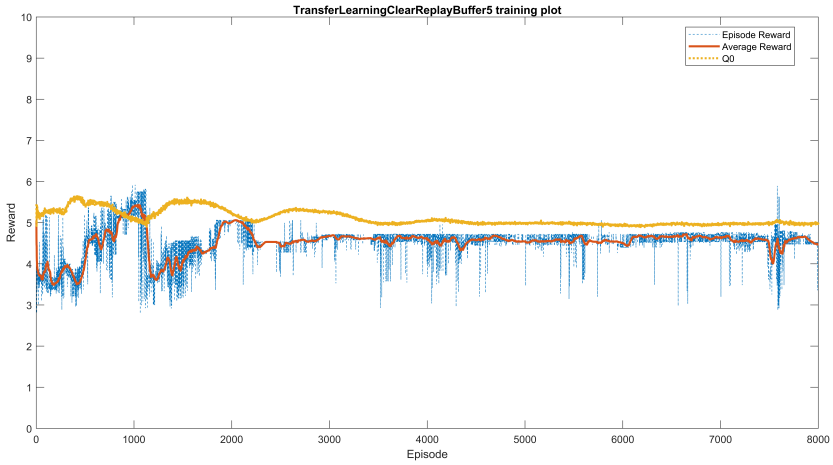


(c) 10 problems

Figure 84: Training graphs of transfer learning agents with experience buffer emptied



(a) 25 problems



(b) 50 problems

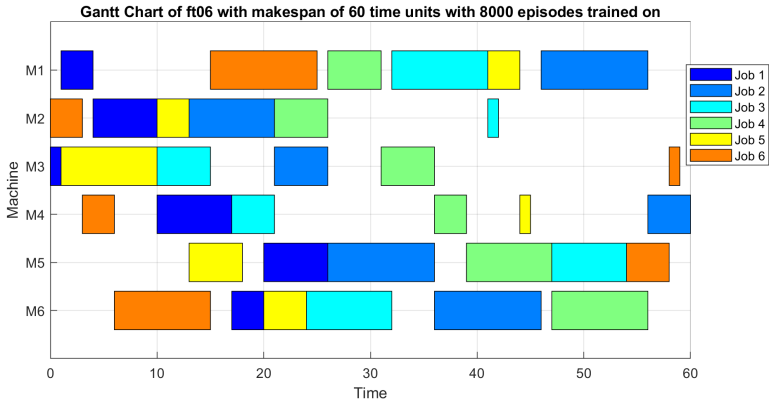
Figure 85: Training graphs of transfer learning agents with experience buffer emptied

To completely assess how this changes the performance, the statistics are given in Table 86. Here, it can be seen that for the first four agents a slight increase in performance, given the rewards, is found. However, the performance of the fifth agent drastically decreases, which is probably a result from the difference in learning behavior. As the learning behavior is not what is found from the benchmark agent, in the final test the hyperparameters of the learning rate, ϵ and ϵ decay are adjusted to be as they were before the training of the agent, to assess a difference in performance.

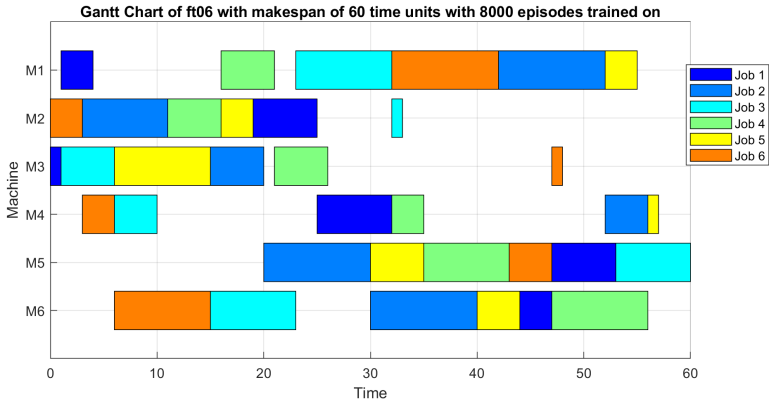
Agent	**Agent 1**			**Agent 2**			**Agent 3**			**Agent 4**			**Agent 5**		
metric	**R**	**C**	U_t	**R**	**C**	U_t	**R**	**C**	U_t	**R**	**C**	U_t	**R**	**C**	U_t
UB	6.00	60	0.600	6.46	60	0.646	6.25	61	0.625	6.77	60	0.677	4.73	77	0.473
LB	6.00	60	0.600	6.46	60	0.646	6.25	61	0.625	6.77	60	0.677	4.73	77	0.473
Mean	6.00	60	0.600	6.46	60	0.646	6.25	61	0.625	6.77	60	0.677	4.73	77	0.473
Var	8.05E-31	0	1.26E-32	7.24E-30	0	0	8.05E-31	0	5.03E-32	3.22E-30	0	1.26E-32	8.05E-31	0	2.83E-32
StDev	8.97E-16	0	1.12E-16	2.69E-15	0	0	8.97E-16	0	2.24E-16	1.79E-15	0	1.12E-16	8.97E-16	0	1.68E-16

Table 86: Statistical results of 50 runs on ft06 from transfer learning agents with emptied experience buffer

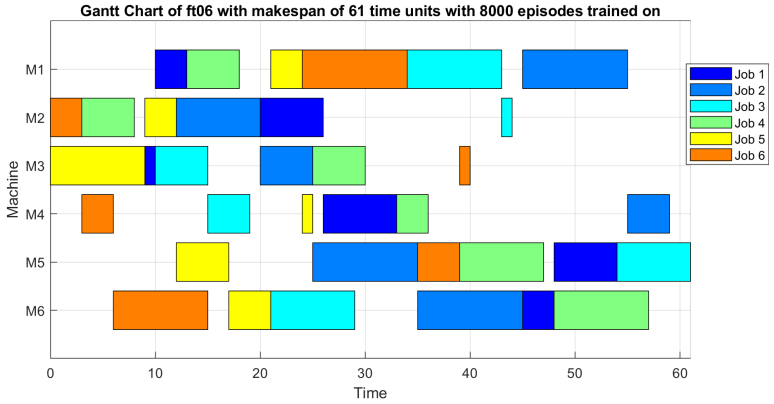
Finally, to show the differences between agents, in Figures 86 and 87 a Gantt chart per agent is given. As can be seen in Figures 86a, 86b and 87a each of the agents find a different approach to reaching the makespan of 60, showing difference in learned policy while receiving a comparable reward.



(a) 1 problem

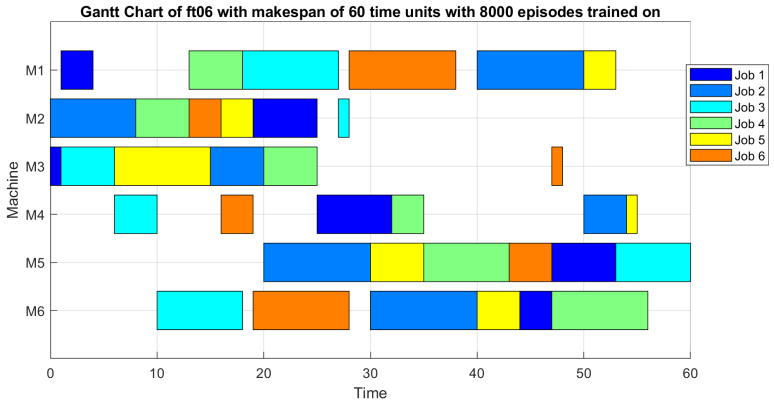


(b) 5 problems

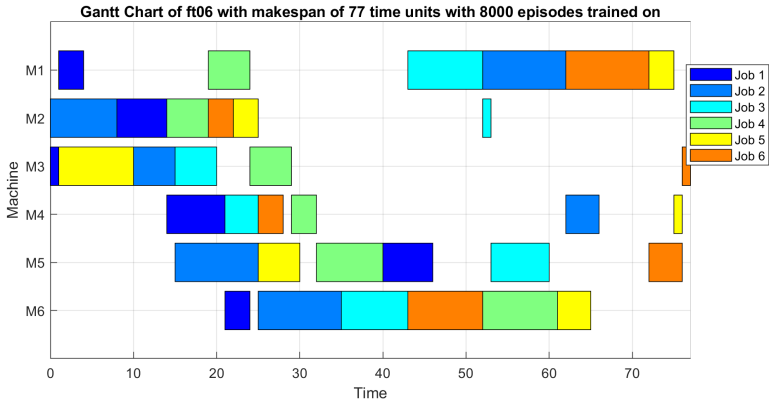


(c) 10 problems

Figure 86: Gantt charts of transfer learning agents with experience buffer emptied



(a) 25 problems

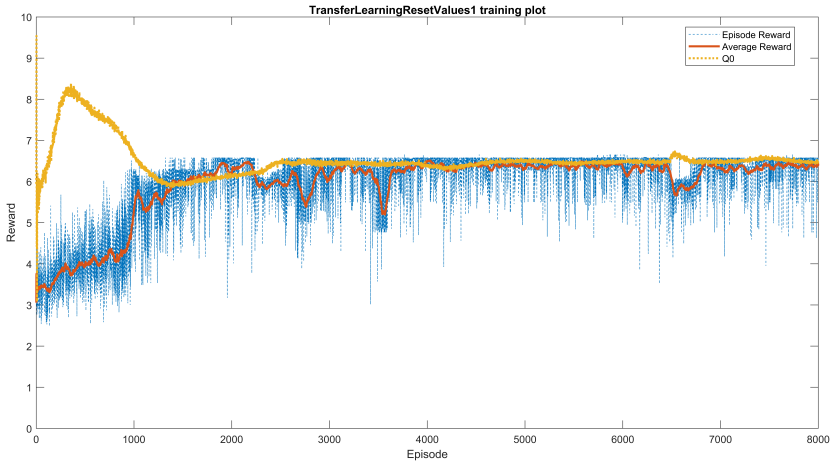


(b) 50 problems

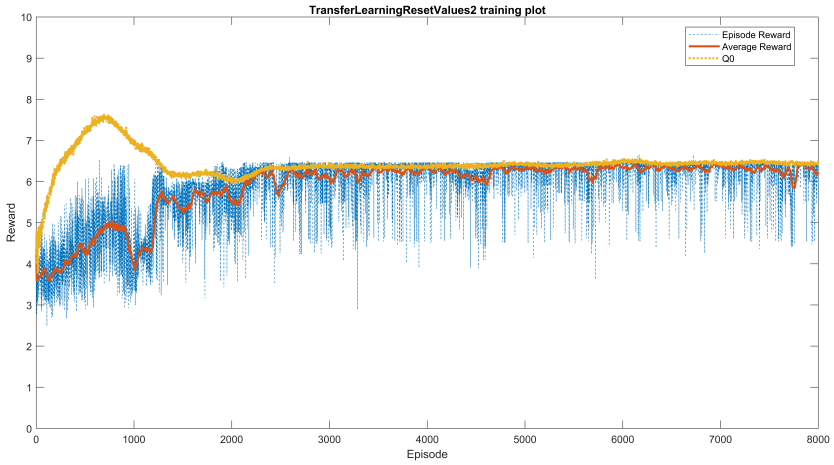
Figure 87: Gantt charts of transfer learning agents with experience buffer emptied

M.3 Transfer learning with changing parameters and emptying experience buffer

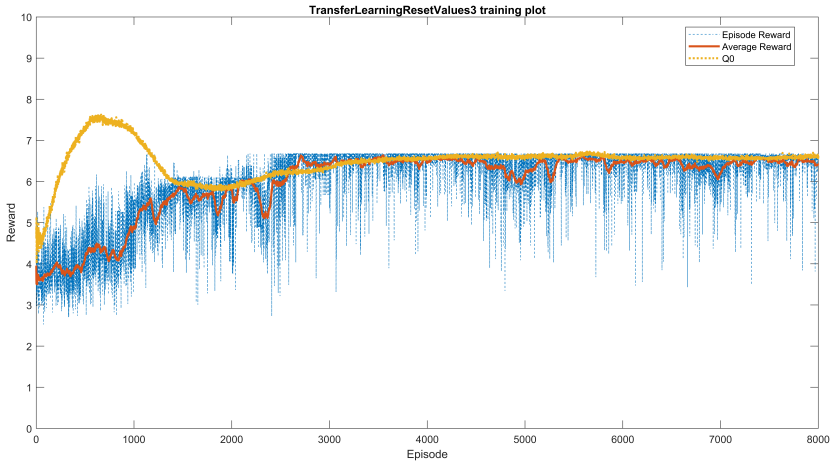
In this section, the agents are trained with both an emptied experience buffer and some of the hyperparameters are reset. In Figure 88 and 89 the training graphs are shown. Here, it can be seen that they are comparable to what is expected from the learning behavior, as it is the same behavior the final hyperparameter tuned agent shows in Figure 70 in Appendix K.7.



(a) 1 problem

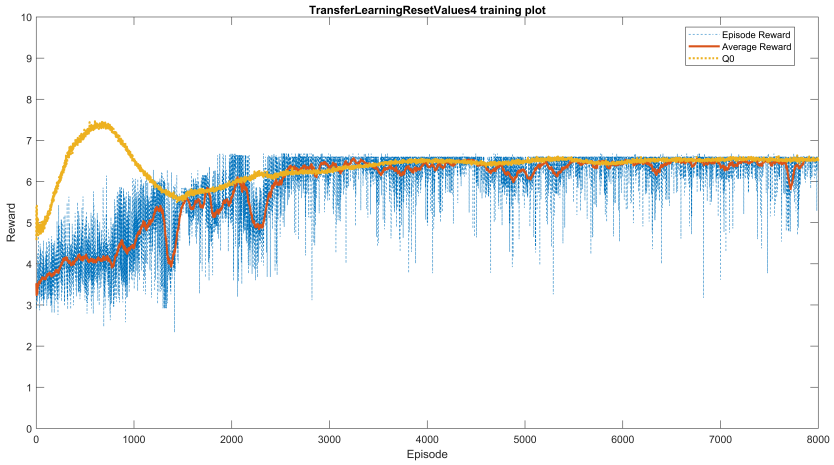


(b) 5 problems

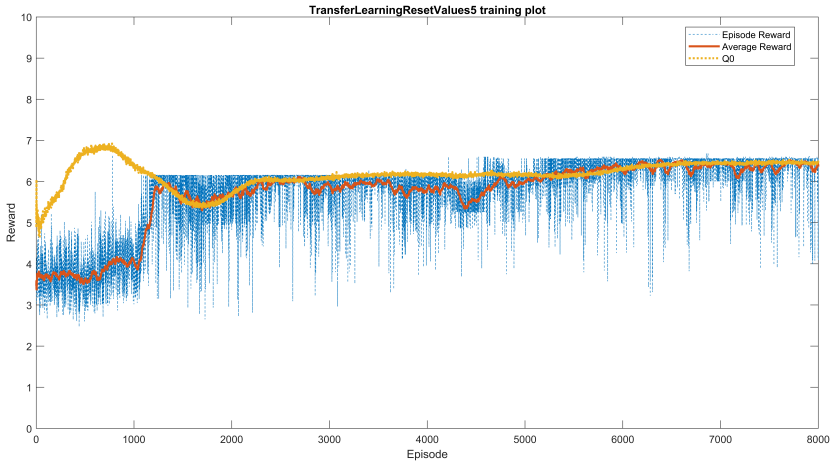


(c) 10 problems

Figure 88: Training graphs of transfer learning agents with values reset



(a) 25 problems



(b) 50 problems

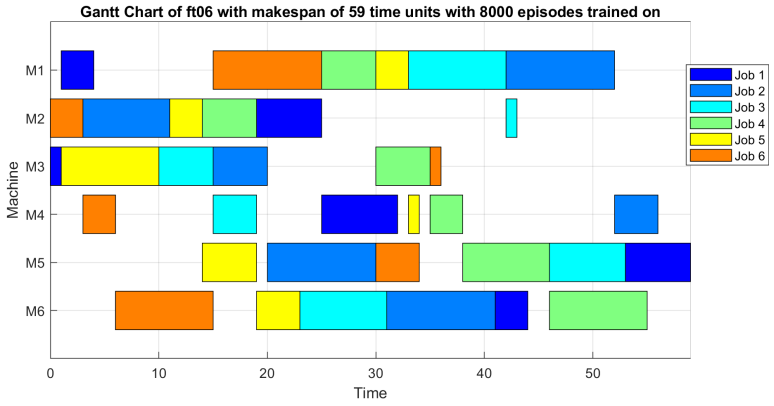
Figure 89: Training graphs of transfer learning agents with values reset

When assessing the performance, the training graphs show comparable behavior and rewards found. Hence, in Table 87 the statistical performances are given. It is found that by changing the hyperparameters the agents overall show better performance each, which is to be expected as this is equal learning to the finalized hyperparameter tuned agent. However, in terms of reward they do not improve upon that agent, only gaining a comparable reward.

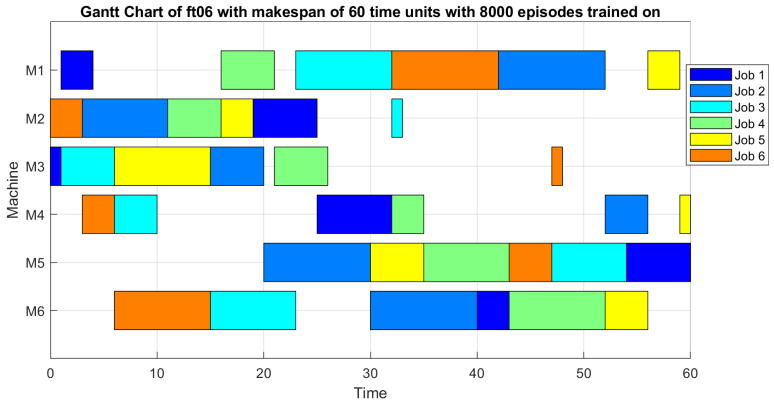
Agent	**Agent 1**			**Agent 2**			**Agent 3**			**Agent 4**			**Agent 5**		
metric	**R**	**C**	U_t	**R**	**C**	U_t	**R**	**C**	U_t	**R**	**C**	U_t	**R**	**C**	U_t
UB	6.58	59	0.658	6.35	60	0.635	6.50	61	0.650	6.60	60	0.660	6.60	60	0.660
LB	6.58	59	0.658	6.35	60	0.635	6.50	61	0.650	6.60	60	0.660	6.60	60	0.660
Mean	6.58	59	0.658	6.35	60	0.635	6.50	61	0.650	6.60	60	0.660	6.60	60	0.660
Var	8.05E-31	0	1.26E-32	3.22E-30	0	0	3.22E-30	0	1.26E-32	0	0	1.26E-32	0	0	1.26E-32
StDev	8.97E-16	0	1.12E-16	1.79E-15	0	0	1.79E-15	0	1.12E-16	0	0	1.12E-16	0	0	1.12E-16

Table 87: Statistical results of 50 runs on ft06 from transfer learning agents

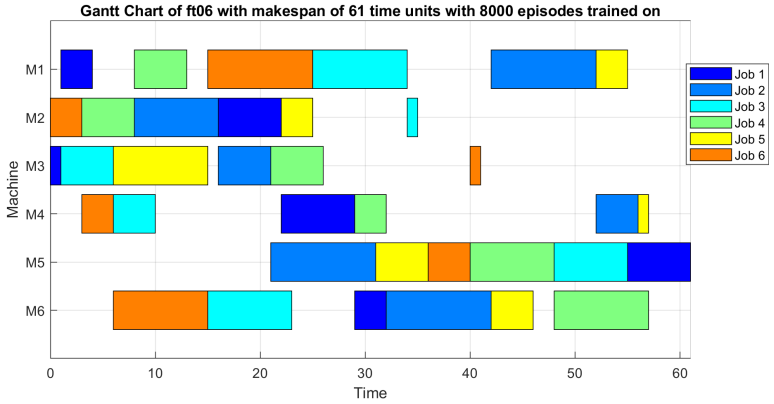
In Figure 90 and 91 Gantt charts are shown for each agent. Although earlier the agents found different results, here agent 4 and 5 create the same schedule for the jobs as they found the same reward and makespan. From the Gantt charts it can also be seen that these agents are way more comparable to each other, probably due to the comparable learning behavior.



(a) 1 problem

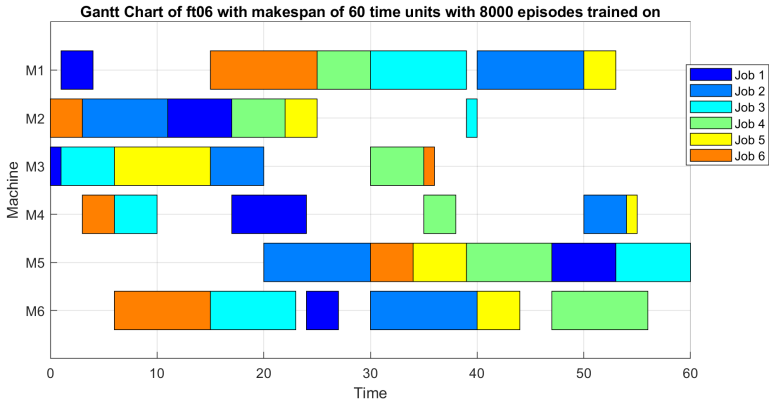


(b) 5 problems

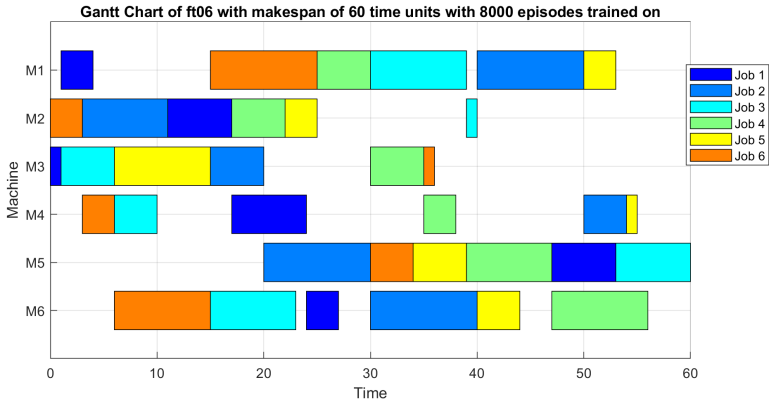


(c) 10 problems

Figure 90: Gantt charts of transfer learning agents with reset hyperparameters



(a) 25 problems



(b) 50 problems

Figure 91: Gantt charts of transfer learning agents with reset hyperparameters

From these results, the transfer learning emptying the experience buffer without changing the hyperparameters seem to work best, increasing the reward slightly over the chosen hyperparameter tuned agent.

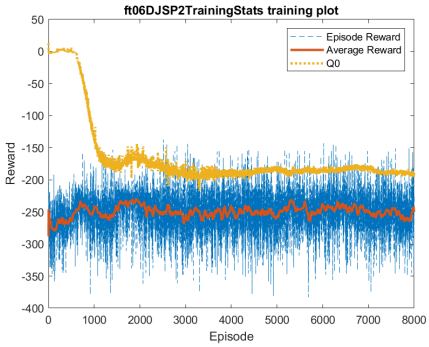
N Dynamic job shop scheduling: reward structures

The reward structures are proposed and tested in this Appendix. First, each reward structure is deployed in the created environment for DJSP. Next, the agents rewards are normalized. Finally, the normalized rewards are deployed with different hyperparameters, for better found rewards and are assessed on the performance and balancing of the objective, to minimize tardiness, maximize the utilization rate and minimize the makespan. Depending on the reward structure, the slack is minimized or maximized.

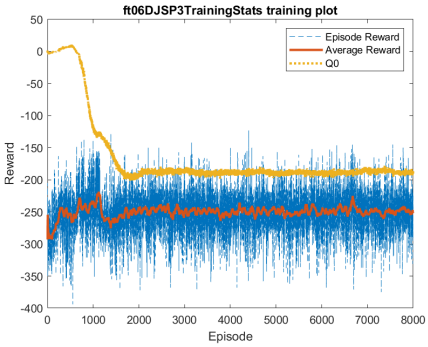
N.1 First reward structure test

For the first test, standard hyperparameter values of Matlab have been used as given in Chapter 8.1, with 3 layers and 256 neurons. The created training graphs are given in Figure 92.

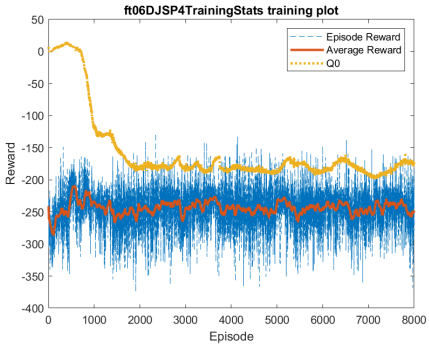
(a) Reward 1



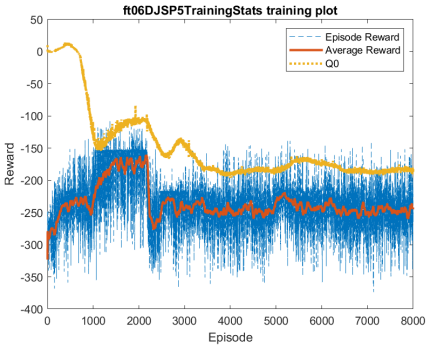
(b) Reward 2



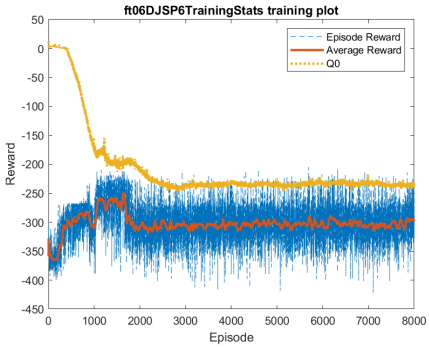
(c) Reward 3



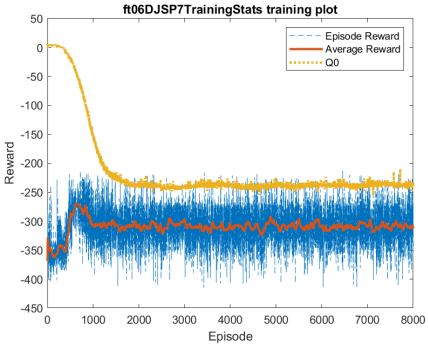
(d) Reward 4



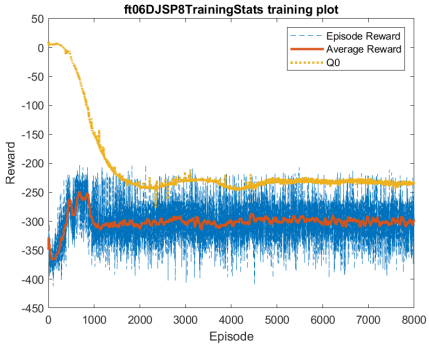
(e) Reward 5



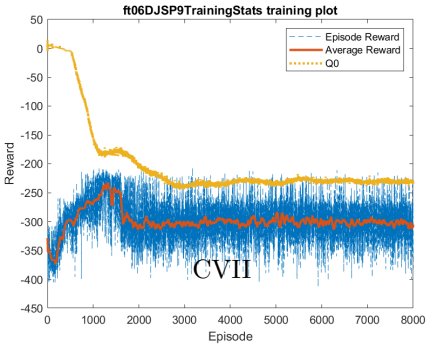
(f) Reward 6



(g) Reward 7



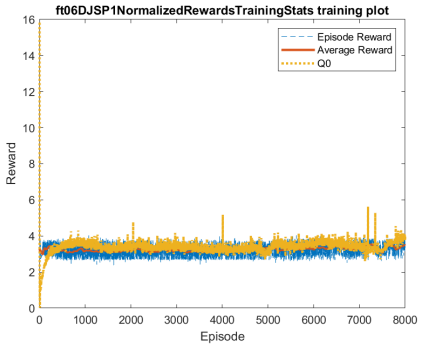
(h) Reward 8



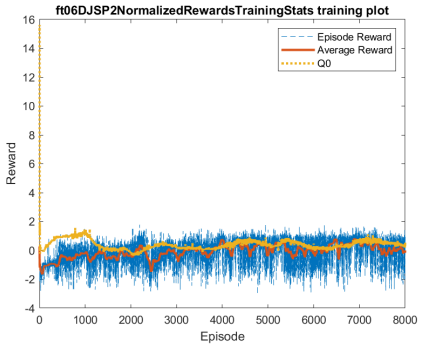
From these figures it is found that they are not yet quite comparable. Hence, the figures are normalized, to ensure that the reward signals given to the agents do not change the neural networks weights and biases too much. Also, the reward structures where the utilization is added to them, the utilization rate probably has little to no influence due to the big difference between the reward gained for the slack and tardiness in comparison to utilization rate.

N.2 Second reward structure test: normalization

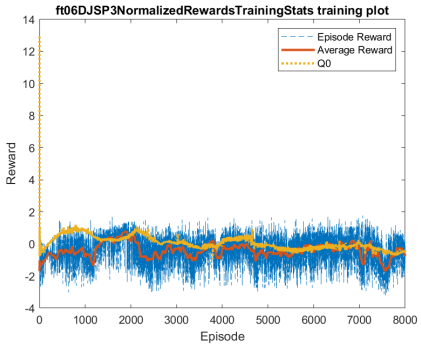
For the normalization, the average utilization reward does not have to be normalized as this is already a reward between 0 and 1. The other eight proposed rewards have been normalized by determining the maximum values of the expected slack, expected tardiness, actual slack and actual tardiness found. Each reward is normalized with these values, to always give a reward between 0 and 1. The new training graphs are given in Figure 93.



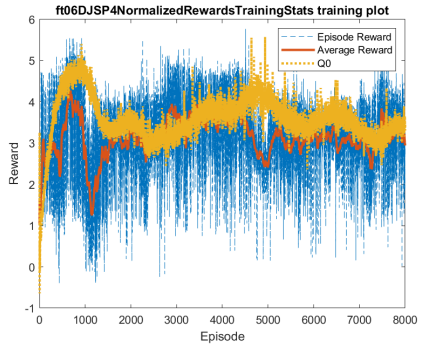
(a) Reward 1



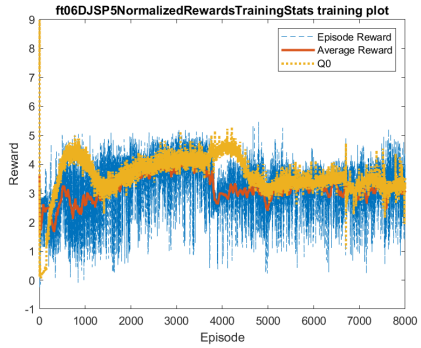
(b) Reward 2



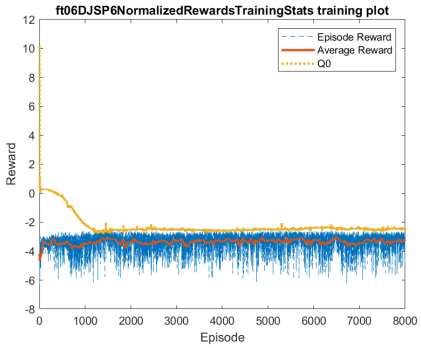
(c) Reward 3



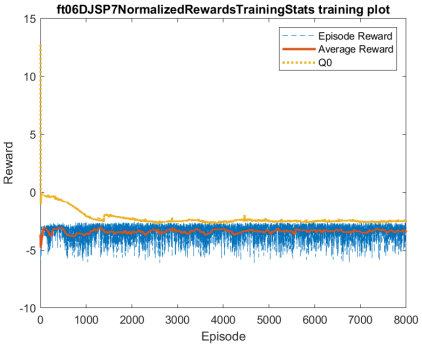
(d) Reward 4



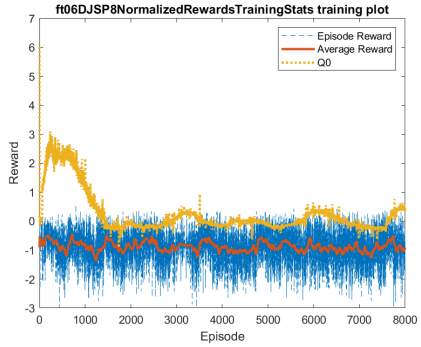
(e) Reward 5



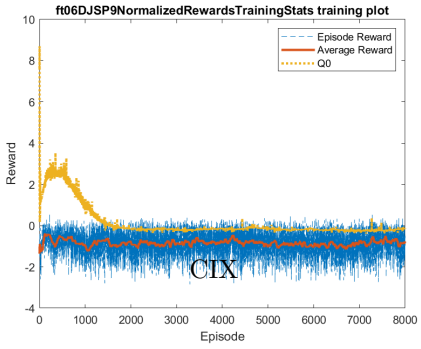
(f) Reward 6



(g) Reward 7



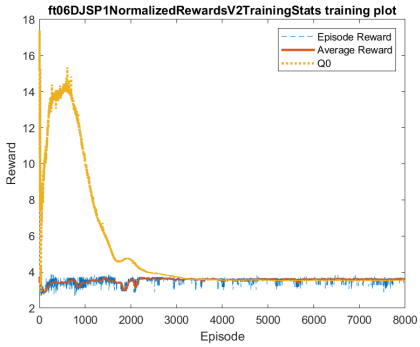
(h) Reward 8



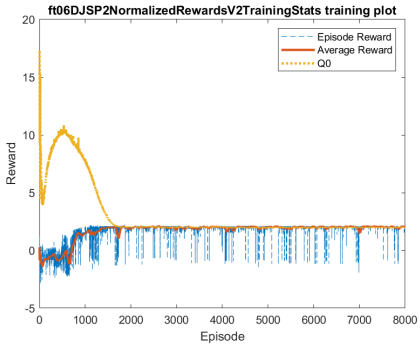
From the graphs it is found that in comparison to the graphs without the normalization, the Q0 estimation improves when normalized. However, the graphs show no real improvement upon the reward gained over training. Hence, a final test will be done where the hyperparameters are changed, to hopefully improve the learning behavior. It is found that each of them create a big oscillation in rewards gained. This is likely due to the relatively high learning rate the standard settings of MATLAB uses. Hence, the learning rate is reduced from 0.01 to 1E-4, to reduce the size of the changes to the weights made for the neural network, hence being more constant in gained rewards and being able to better assess how well each reward performs. Also, the epsilon values are changed to encourage more exploration, with the decay rate changed from 0.01 to 0.0001. When using these values, the episodes needed to converge to the minimum value of 0.005 takes a longer time, hence having more exploration instead of exploitation.

N.3 Final reward structure test: assessing performance

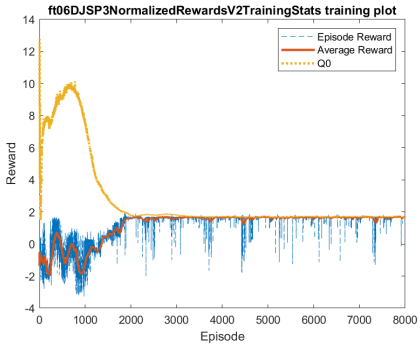
The final test for the reward structure has been done, which shows some promise. The training graphs are given in Figure 94. From these figures, it is found that the training behavior is still not perfect, which is to be expected. However, for most of them, the Q0 value aligns with the found reward and they all show improvement upon the initial reward found.



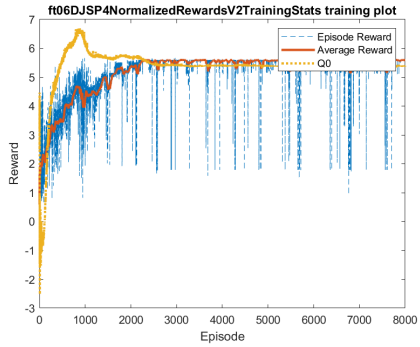
(a) Reward 1



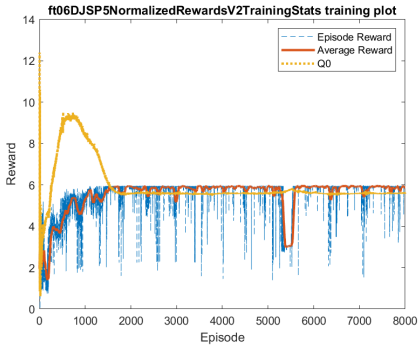
(b) Reward 2



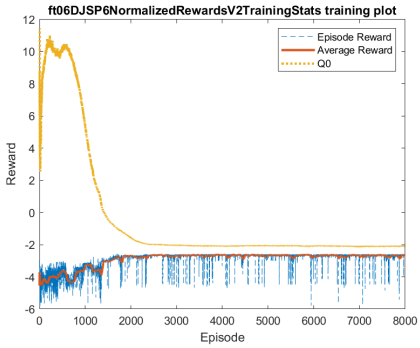
(c) Reward 3



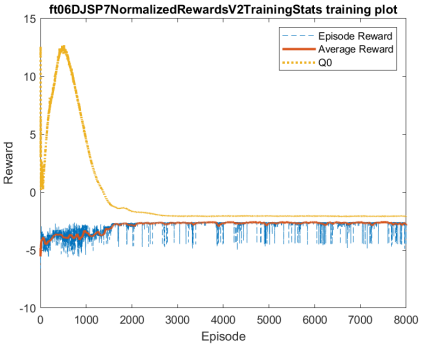
(d) Reward 4



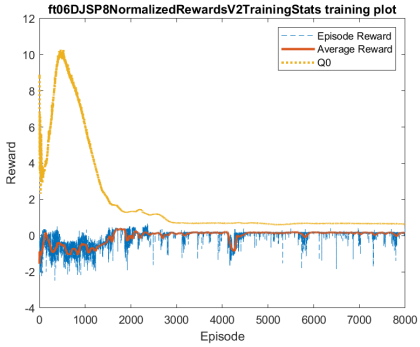
(e) Reward 5



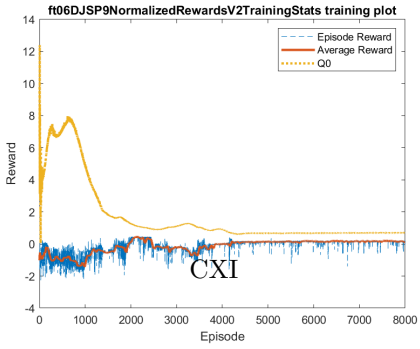
(f) Reward 6



(g) Reward 7



(h) Reward 8



Looking at the found results from these different reward structures, the performance assessment is done based on the found utilization rate, makespan, average slack and average tardiness. First, the assessment is made on how the dispatching rules used perform, these results are given in Table 88. Here, it is found that the best performing dispatching rule is HRPT, which is quite capable of balancing the average tardiness and average slack, while having a low makespan and high utilization rate.

	LRPT	**HRPT**	**SPT**	**LPT**	**HTPT**	**LTPT**	**FIFO**	**LIFO**	**LOR**	**MOR**
makespan	84	**63**	86	68	68	86	93	85	86	89
utilization rate	0.49	**0.63**	0.48	0.58	0.58	0.48	0.45	0.57	0.48	0.45
average slack	0.00	5.50	1.33	6.17	6.17	1.33	3.17	6.17	1.33	3.17
average tardiness	12.50	**6.50**	9.83	9.67	9.67	9.83	13.00	10.33	9.50	14.83

Table 88: Performance of dispatching rules on dynamic job shop scheduling problems

Secondly, the reward structures are assessed on performance in Table 89. Here, it is found that the reward using only the utilization rate still performs best in terms of minimizing both makespan and average tardiness, while having a high utilization rate. While the last 4 reward structures, which are penalized for having slack to achieve just-in-time delivery show that they are indeed capable of minimizing slack, the overall performance leaves to be desired.

Reward structure	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
makespan	**61**	68	69	68	66	75	75	68	68
utilization rate	**0.61**	0.60	0.59	0.60	**0.61**	0.50	0.50	0.59	0.59
average slack	3.83	6.00	5.83	6.00	5.67	1.33	1.33	1.83	1.83
average tardiness	**4.00**	5.67	7.67	5.67	9.67	10.50	10.50	7.33	7.33

Table 89: Performance of agents with different reward structures on dynamic job shop scheduling problems

Concluding, it is found that the best performing reward structure would be the first one, focusing on the change in utilization rate per action. This reward outperforms the dispatching rules, without optimizing the hyperparameters to the best standard. Hence, this reward structure is chosen for the tests in the next Chapter.

O Dynamic job shop scheduling: Hyperparameter tuning

In this appendix chapter, the hyperparameter tuning for the dynamic job shop scheduling is set out. Based on findings in this research, the range for hyperparameters is already converged to some more specific values, to speed up the process.

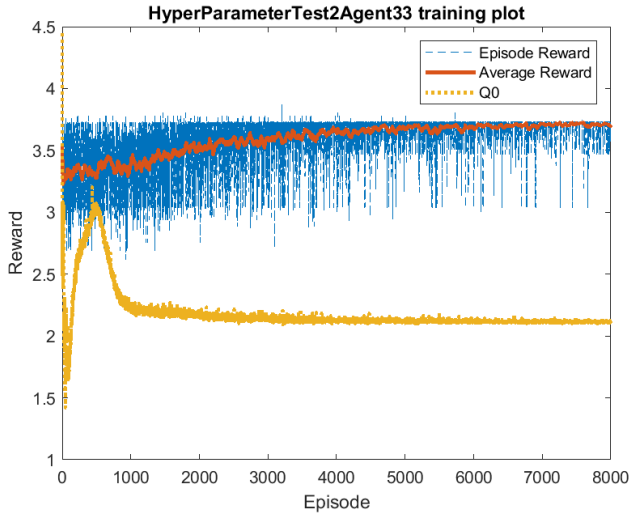
O.1 First hyperparameter test

For the first hyperparameter test, the values used are given in Table 90. These are based on both earlier tests as well as the reward structure tests, hence the range of values is kept low.

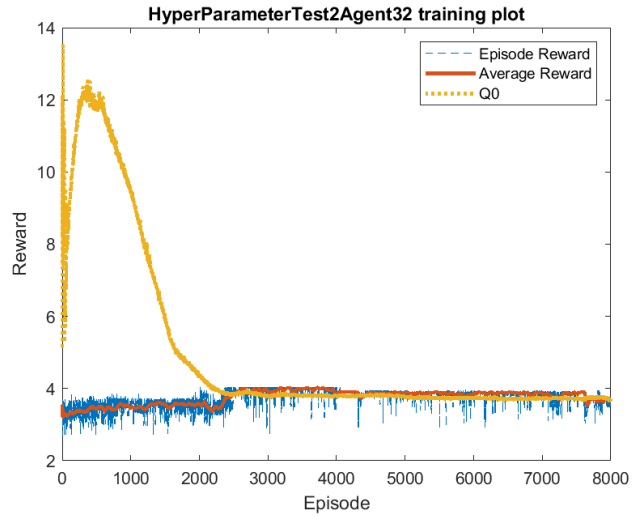
LR	DF	ϵ	ϵ_m	ϵ_d	Neurons	Hidden layers
1e-4 1e-5 1e-6	0.75 0.99	0.75 1	1e-8 1e-7 1e-6	1e-4 1e-5 1e-6	64 128 256	3 to 5

Table 90: Possible values first hyperparameter test for dynamic job shop

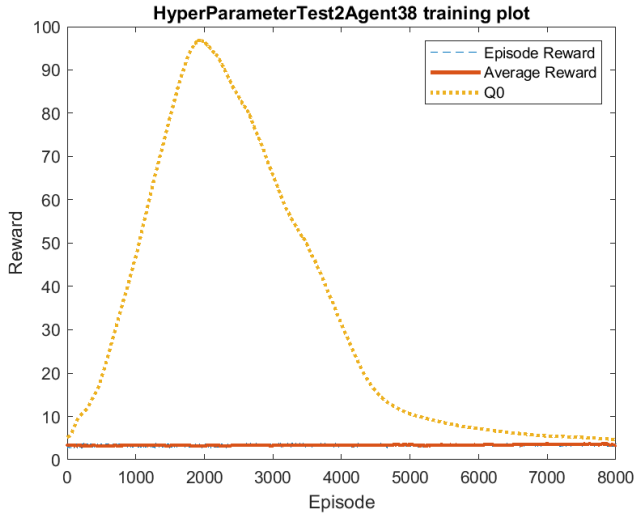
Again, the agents are divided into subcategories based on their learning graph. The categories being Q0 below the found reward, expected, or stable, behavior and Q0 higher than 30. The latter is chosen due to the Q0 being significantly higher than the found reward, which is between 3 and 4. In Figure 95 these three different categories are shown.



(a) low Q0



(b) Stable



(c) Q0 ¿ 30

Figure 95: Representation of different categories in hyperparameter test 1 for dynamic job shop scheduling

From these, the categorized agents are given in Table 91, with the agent values given in Table 92.

Category	Agents
Low Q0	2 3 5 7 12 17 21 23 24 27 29 31 33 34 35 36 37 39 41 44 45 47 48 49 51 52
Stable	9 10 14 16 18 20 22 25 28 32 42 46 50
Q0 ¿ 30	1 4 6 8 11 13 15 19 26 30 38 40 43 54

Table 91: Overview of different categories and agents in test 1 for DJSP

Agent	LR	DF	ϵ	ϵ_d	ϵ_m	Neurons	HL
1	1E-06	0.99	0.75	1E-07	1E-06	64	3
2	1E-05	0.75	1	1E-07	1E-05	128	5
3	1E-06	0.75	1	1E-06	1E-06	128	3
4	1E-06	0.99	0.75	1E-06	1E-05	256	3
5	1E-04	0.75	0.75	1E-07	1E-04	256	3
6	1E-04	0.99	1	1E-07	1E-06	128	5
7	1E-05	0.75	0.75	1E-07	1E-04	256	5
8	1E-04	0.99	1	1E-08	1E-06	64	4
9	1E-04	0.99	1	1E-06	1E-05	256	5
10	1E-06	0.99	1	1E-07	1E-04	64	3
11	1E-05	0.99	1	1E-07	1E-05	64	4
12	1E-06	0.75	1	1E-08	1E-06	64	4
13	1E-06	0.99	0.75	1E-06	1E-06	64	3
14	1E-04	0.99	1	1E-06	1E-06	64	5
15	1E-06	0.99	1	1E-07	1E-05	64	5
16	1E-05	0.99	1	1E-08	1E-04	64	3
17	1E-05	0.75	1	1E-06	1E-05	128	5
18	1E-05	0.99	1	1E-06	1E-05	128	3
19	1E-06	0.99	1	1E-06	1E-05	64	4
20	1E-04	0.99	0.75	1E-06	1E-06	256	5
21	1E-04	0.75	1	1E-08	1E-05	256	3
22	1E-04	0.99	0.75	1E-06	1E-06	64	4
23	1E-06	0.75	0.75	1E-08	1E-06	64	3
24	1E-05	0.75	0.75	1E-07	1E-06	64	5
25	1E-05	0.99	1	1E-06	1E-05	64	5
26	1E-06	0.99	1	1E-06	1E-05	256	3
27	1E-05	0.75	1	1E-07	1E-06	64	4
28	1E-04	0.99	1	1E-07	1E-05	256	3
29	1E-05	0.75	0.75	1E-08	1E-05	256	5
30	1E-04	0.99	1	1E-07	1E-06	128	3
31	1E-04	0.75	1	1E-08	1E-06	128	4
32	1E-04	0.99	0.75	1E-08	1E-04	128	3
33	1E-05	0.75	1	1E-08	1E-05	256	5
34	1E-06	0.75	0.75	1E-06	1E-06	128	3
35	1E-05	0.75	0.75	1E-08	1E-06	64	5
36	1E-06	0.75	0.75	1E-06	1E-05	256	4
37	1E-06	0.75	1	1E-06	1E-06	256	4
38	1E-06	0.99	1	1E-08	1E-05	64	4
39	1E-04	0.75	1	1E-08	1E-06	64	3
40	1E-06	0.99	0.75	1E-08	1E-06	256	4
41	1E-06	0.75	0.75	1E-07	1E-06	64	4
42	1E-04	0.99	0.75	1E-06	1E-04	256	4
43	1E-04	0.99	1	1E-07	1E-05	64	3
44	1E-06	0.75	0.75	1E-08	1E-04	64	4
45	1E-06	0.75	1	1E-08	1E-06	256	3

46	1E-04	0.99	0.75	1E-06	1E-05	128	5
47	1E-06	0.75	1	1E-07	1E-05	64	3
48	1E-04	0.75	0.75	1E-06	1E-04	128	3
49	1E-05	0.75	1	1E-07	1E-05	64	5
50	1E-04	0.99	0.75	1E-08	1E-05	64	5
51	1E-05	0.75	0.75	1E-06	1E-05	256	5
52	1E-06	0.75	1	1E-06	1E-04	128	5
53	1E-04	0.75	0.75	1E-07	1E-05	256	5
54	1E-06	0.99	1	1E-08	1E-06	256	5

Table 92: All settings for the agents created in test 1 for DJSP

From the categorization of these agents, it is found that the discount factor of 0.75 results in the Q0 always being below the average found reward. Also, the high Q0 values are a result of the learning rate being 1E-6. Further assessing the learning behaviors it is also found that there are no agents showing a good balance between exploration and exploitation, thus the decay values for ϵ have been changed. The ϵ value is set to 1 to ensure exploration early on. Finally, the agents with 64 neurons show worse results in terms of behavior than those with 128 and 256.

O.2 Second hyperparameter test

From the previous findings, the new hyperparameters are chosen and given in Table 93. These combinations give us 48 possibilities, which will all be tested in this hyperparameter test.

LR	DF	ϵ	ϵ_m	ϵ_d	Neurons	Hidden layers
1e-4 1e-5	0.99	1	1e-6	1e-3 5e-4 1e-4 5e-5	128 256	3 to 5

Table 93: Possible values second hyperparameter test for dynamic job shop

The combination of hyperparameters are given in Table 94.

agent	LR	DF	ϵ	ϵ_m	ϵ_d	neurons	layers
1	1E-05	0.99	1	1E-06	5E-05	128	3
2	1E-04	0.99	1	1E-06	1E-03	256	4
3	1E-04	0.99	1	1E-06	5E-05	256	4
4	1E-05	0.99	1	1E-06	5E-04	256	5
5	1E-04	0.99	1	1E-06	5E-04	128	5
6	1E-05	0.99	1	1E-06	1E-04	128	4
7	1E-04	0.99	1	1E-06	1E-04	256	5
8	1E-04	0.99	1	1E-06	1E-04	128	3
9	1E-05	0.99	1	1E-06	1E-03	256	3
10	1E-05	0.99	1	1E-06	5E-05	256	4
11	1E-05	0.99	1	1E-06	1E-03	128	5
12	1E-05	0.99	1	1E-06	1E-03	256	4
13	1E-04	0.99	1	1E-06	1E-03	128	3
14	1E-05	0.99	1	1E-06	5E-04	128	4
15	1E-05	0.99	1	1E-06	1E-04	128	3
16	1E-05	0.99	1	1E-06	1E-03	256	5
17	1E-04	0.99	1	1E-06	1E-03	128	5
18	1E-05	0.99	1	1E-06	5E-04	128	3
19	1E-04	0.99	1	1E-06	5E-04	256	4
20	1E-05	0.99	1	1E-06	1E-04	256	3
21	1E-04	0.99	1	1E-06	1E-04	128	4
22	1E-04	0.99	1	1E-06	5E-05	128	3
23	1E-05	0.99	1	1E-06	5E-04	256	3
24	1E-04	0.99	1	1E-06	1E-03	128	4

25	1E-04	0.99	1	1E-06	5E-05	256	3
26	1E-04	0.99	1	1E-06	1E-04	128	5
27	1E-04	0.99	1	1E-06	5E-04	256	3
28	1E-04	0.99	1	1E-06	5E-04	128	4
29	1E-04	0.99	1	1E-06	5E-05	128	5
30	1E-04	0.99	1	1E-06	1E-03	256	5
31	1E-05	0.99	1	1E-06	5E-04	128	5
32	1E-04	0.99	1	1E-06	1E-04	256	3
33	1E-05	0.99	1	1E-06	1E-03	128	3
34	1E-04	0.99	1	1E-06	1E-03	256	3
35	1E-04	0.99	1	1E-06	5E-05	256	5
36	1E-05	0.99	1	1E-06	5E-05	128	5
37	1E-05	0.99	1	1E-06	1E-04	256	4
38	1E-04	0.99	1	1E-06	5E-05	128	4
39	1E-05	0.99	1	1E-06	1E-03	128	4
40	1E-04	0.99	1	1E-06	1E-04	256	4
41	1E-05	0.99	1	1E-06	5E-05	256	3
42	1E-05	0.99	1	1E-06	1E-04	256	5
43	1E-04	0.99	1	1E-06	5E-04	128	3
44	1E-05	0.99	1	1E-06	5E-04	256	4
45	1E-05	0.99	1	1E-06	1E-04	128	5
46	1E-04	0.99	1	1E-06	5E-04	256	5
47	1E-05	0.99	1	1E-06	5E-05	256	5
48	1E-05	0.99	1	1E-06	5E-05	128	4

Table 94: Hyperparameter settings per agent

For this iteration of the hyperparameter tests, each agent will be tested in terms of performance. After finding the best performers, the learning behavior is assessed as well. In Figure 96 this comparison made. Also, in Table 95 values are also shown.

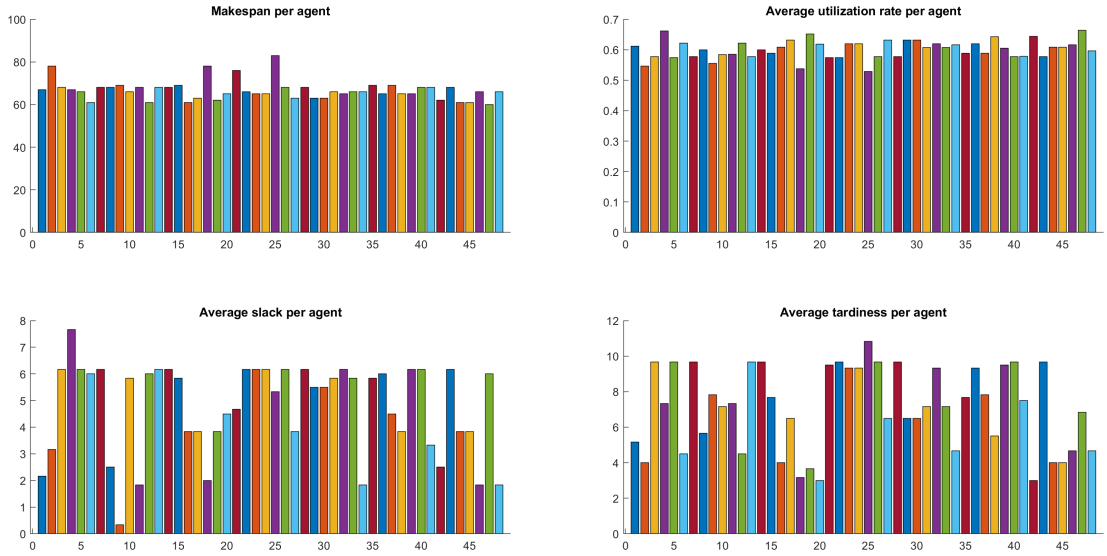


Figure 96: Comparison of agents in bar graphs

Agent	max S	max T	avg S	avg T	Ut rate	makespan	reward
1	7	17	2.166667	5.166667	0.612072	67	3.672434

2	10	13	3.166667	4	0.545893	78	3.275355
3	24	34	6.166667	9.666667	0.577031	68	3.462187
4	23	16	7.666667	7.333333	0.661553	67	3.969317
5	24	31	6.166667	9.666667	0.573948	66	3.443687
6	23	17	6	4.5	0.621723	61	3.730337
7	24	34	6.166667	9.666667	0.577031	68	3.462187
8	11	17	2.5	5.666667	0.599446	68	3.596673
9	2	26	0.333333	7.833333	0.554955	69	3.329733
10	24	18	5.833333	7.166667	0.58417	66	3.50502
11	8	18	1.833333	7.333333	0.585073	68	3.51044
12	23	17	6	4.5	0.621723	61	3.730337
13	24	34	6.166667	9.666667	0.577031	68	3.462187
14	24	34	6.166667	9.666667	0.599699	68	3.598194
15	24	26	5.833333	7.666667	0.588467	69	3.530804
16	10	17	3.833333	4	0.608385	61	3.650311
17	13	23	3.833333	6.5	0.631752	63	3.79051
18	10	13	2	3.166667	0.537215	78	3.223293
19	20	10	3.833333	3.666667	0.651755	62	3.910532
20	18	10	4.5	3	0.618803	65	3.712821
21	15	33	4.666667	9.5	0.573572	76	3.441431
22	24	31	6.166667	9.666667	0.573948	66	3.443687
23	24	31	6.166667	9.333333	0.619438	65	3.716631
24	24	31	6.166667	9.333333	0.619438	65	3.716631
25	24	23	5.333333	10.83333	0.528382	83	3.170292
26	24	34	6.166667	9.666667	0.577031	68	3.462187
27	13	23	3.833333	6.5	0.631752	63	3.79051
28	24	34	6.166667	9.666667	0.577031	68	3.462187
29	23	23	5.5	6.5	0.631752	63	3.79051
30	23	23	5.5	6.5	0.631752	63	3.79051
31	24	18	5.833333	7.166667	0.606838	66	3.641027
32	24	31	6.166667	9.333333	0.619438	65	3.716631
33	24	18	5.833333	7.166667	0.606838	66	3.641027
34	8	10	1.833333	4.666667	0.615691	66	3.694144
35	24	26	5.833333	7.666667	0.588467	69	3.530804
36	23	31	6	9.333333	0.619438	65	3.716631
37	24	26	4.5	7.833333	0.588467	69	3.530804
38	20	10	3.833333	5.5	0.643373	65	3.860237
39	24	31	6.166667	9.5	0.605461	65	3.632765
40	24	34	6.166667	9.666667	0.577031	68	3.462187
41	11	18	3.333333	7.5	0.578324	68	3.469943
42	12	6	2.5	3	0.643788	62	3.862725
43	24	34	6.166667	9.666667	0.577031	68	3.462187
44	10	17	3.833333	4	0.608385	61	3.650311
45	10	17	3.833333	4	0.608385	61	3.650311
46	8	10	1.833333	4.666667	0.615691	66	3.694144
47	20	24	6	6.833333	0.664217	60	3.985299
48	8	10	1.833333	4.666667	0.596276	66	3.577657

Table 95: Performance of each agent

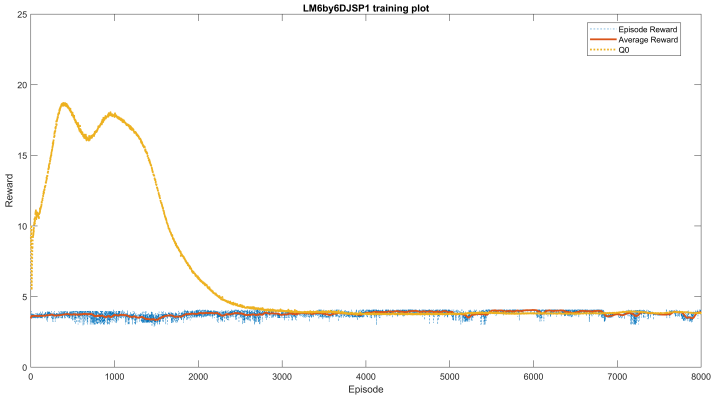
To completely compare the agents, the best found agents are compared to the benchmark found in the previous appendix in Table 96. Here, it is found that while all agents perform comparable or better to the benchmark, agent 47 performs best and together with agent 19 finds the highest reward. Hence, agent 47 is chosen for the hyperparameters.

	makespan	ut rate	avg S	avg T
benchmark	61	0.61	3.83	4.00
6	61	0.62	6.00	4.50
12	61	0.62	6.00	4.50
16	61	0.61	3.83	4.00
17	63	0.63	3.83	6.50
19	62	0.65	3.83	3.67
20	65	0.62	4.50	3.00
27	63	0.63	3.83	6.50
29	63	0.63	5.50	6.50
30	63	0.63	5.50	6.50
38	65	0.64	3.83	5.50
42	62	0.64	2.50	3.00
44	61	0.61	3.83	4.00
45	61	0.61	3.83	4.00
47	60	0.66	6.00	6.83

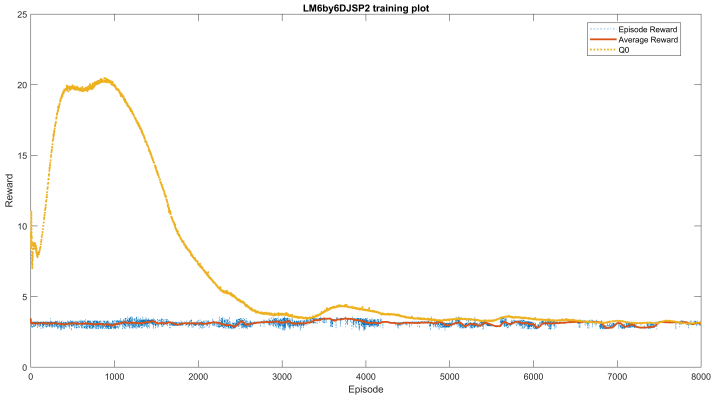
Table 96: Best agents found for second iteration of hyperparameter tuning

P Training on multiple instances: dynamic job shop scheduling

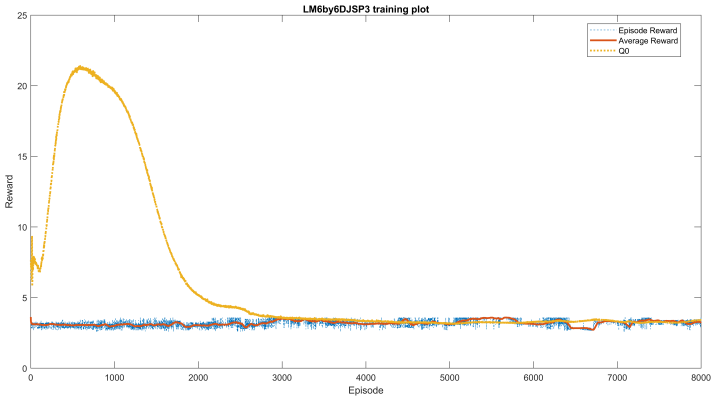
As the hyperparameters have been chosen, now multiple agents are trained on a different number of problems. These are trained on 1, 5, 10, 25 and 50 problems respectively. The training graphs are given in Figure 97 and 98.



(a) 1 problem

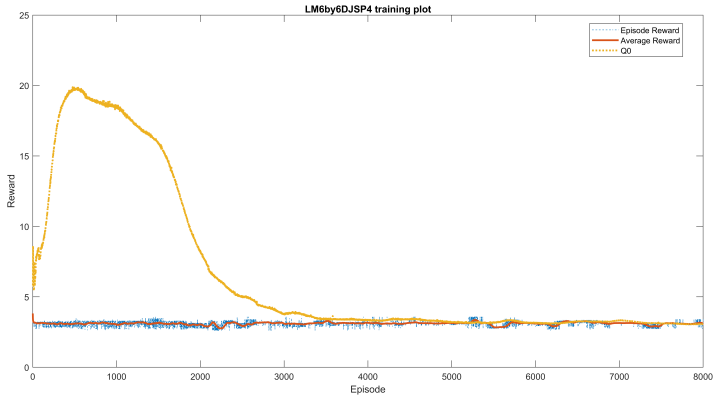


(b) 5 problems

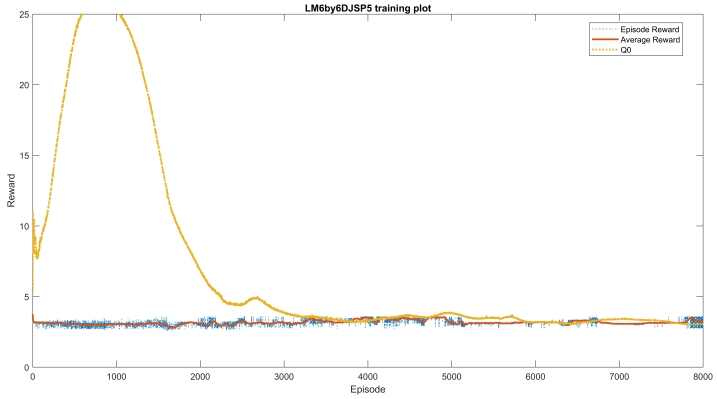


(c) 10 problems

Figure 97: Training graph of agents 1, 2 and 3



(a) 25 problem



(b) 50 problems

Figure 98: Training graph of agents 4 and 5

In Table 97 the results are shown per agent for the problems they were trained on. From these values, it is found that the agents show comparable results on all aspects.

dynamic	agent 1				agent 2				agent 3				agent 4				agent 5			
problem	**C**	**U**	**S**	**T**	**C**	**U**	**S**	**T**	**C**	**U**	**S**	**T**	**C**	**U**	**S**	**T**	**C**	**U**	**S**	**T**
1	67	0.65	5.83	1.00	82	0.60	5.00	4.83	76	0.58	4.50	2.17	72	0.62	6.33	3.67	68	0.62	4.00	3.67
2	84	0.49	5.83	3.33	84	0.50	6.67	7.33	83	0.53	5.83	6.00	83	0.52	4.50	6.67	68	0.60	5.50	1.17
3	81	0.54	4.33	2.83	63	0.59	4.83	4.00	66	0.59	2.17	3.17	80	0.52	4.83	7.83	64	0.63	6.67	4.33
4	73	0.63	7.33	4.17	84	0.57	2.67	10.17	87	0.54	5.17	6.67	77	0.62	2.67	9.00	82	0.59	5.00	9.33
5	80	0.52	6.33	12.00	77	0.54	6.67	6.50	81	0.52	6.33	10.33	86	0.52	6.33	12.33	85	0.56	6.67	10.83
6	91	0.49	11.67	3.33	81	0.52	12.17	1.83	78	0.54	12.83	1.17	100	0.45	8.50	7.50	85	0.51	12.17	2.33
7	86	0.50	4.33	10.67	93	0.56	6.50	8.33	84	0.50	6.33	10.50	73	0.58	7.00	7.83	93	0.56	6.50	8.33
8	76	0.51	6.67	10.50	78	0.54	6.17	6.50	81	0.51	4.00	6.50	72	0.55	9.17	9.33	78	0.54	5.67	6.50
9	79	0.52	5.83	8.83	72	0.57	8.50	7.00	79	0.51	6.17	11.17	78	0.54	5.83	6.67	84	0.58	8.33	5.50
10	85	0.49	4.67	4.00	87	0.49	6.00	10.00	87	0.52	6.50	7.33	87	0.49	5.17	7.33	81	0.50	6.17	9.17
11	63	0.60	6.50	3.50	66	0.58	7.00	4.17	66	0.59	7.50	2.83	69	0.58	7.50	6.00	66	0.58	7.00	4.17
12	75	0.51	9.50	5.00	67	0.56	10.50	3.17	67	0.56	10.50	3.17	79	0.54	9.17	5.17	68	0.56	10.50	3.50
13	58	0.55	9.67	3.67	71	0.51	7.67	5.50	58	0.57	6.17	2.17	66	0.53	7.83	3.00	64	0.52	5.33	2.33
14	97	0.44	4.00	8.67	87	0.48	3.67	8.67	82	0.54	8.33	9.67	88	0.48	5.33	10.33	93	0.49	2.83	12.50
15	86	0.56	7.17	2.17	86	0.56	6.00	1.83	82	0.61	7.50	4.17	82	0.61	7.50	4.17	81	0.56	7.50	4.17
16	78	0.49	5.83	1.17	78	0.49	6.67	1.50	79	0.48	8.00	2.83	72	0.53	9.33	6.17	79	0.48	8.00	2.83
17	68	0.54	9.00	4.00	75	0.54	9.83	2.83	81	0.47	9.00	7.83	75	0.54	9.83	2.83	75	0.54	9.83	2.83
18	76	0.56	7.83	2.50	76	0.56	7.83	2.50	76	0.55	8.00	4.17	68	0.62	12.00	3.00	78	0.59	9.00	2.00
19	89	0.45	6.17	6.00	82	0.48	4.67	5.00	82	0.48	4.50	4.50	82	0.48	5.17	7.17	77	0.51	4.17	2.83
20	85	0.49	2.67	7.50	80	0.53	2.00	6.00	66	0.64	6.83	2.17	81	0.52	6.50	8.50	66	0.64	6.83	2.17
21	71	0.52	2.50	3.50	77	0.51	8.83	6.33	88	0.49	9.50	6.67	67	0.51	5.67	2.33	70	0.53	4.67	3.17
22	75	0.60	7.50	2.67	75	0.54	5.50	5.50	69	0.54	5.67	7.17	89	0.49	6.83	7.00	76	0.56	9.67	10.33
23	89	0.45	5.33	8.67	66	0.52	6.17	7.17	72	0.51	5.83	8.00	89	0.46	5.33	8.67	80	0.47	5.67	8.17
24	74	0.54	9.00	5.17	64	0.56	5.00	3.33	69	0.53	10.83	5.33	75	0.51	11.33	6.83	71	0.53	2.33	3.17
25	68	0.58	8.50	4.33	63	0.61	7.50	2.83	70	0.57	7.67	4.50	66	0.59	7.33	4.50	82	0.58	10.17	1.83
26	78	0.51	4.67	4.33	68	0.57	6.00	2.67	81	0.48	4.83	6.17	81	0.49	4.83	5.67	69	0.57	6.33	9.17
27	68	0.64	10.00	2.83	61	0.64	9.00	0.50	63	0.64	11.00	3.17	64	0.61	8.33	1.33	73	0.60	8.83	1.83
28	89	0.46	3.83	8.00	88	0.46	4.33	7.33	77	0.51	4.00	3.67	80	0.48	4.17	6.00	81	0.56	8.83	5.00
29	86	0.46	5.83	9.50	81	0.47	5.83	9.33	87	0.47	5.83	9.00	78	0.48	5.50	11.00	85	0.46	2.67	9.00
30	78	0.48	7.50	3.67	84	0.47	8.67	4.67	79	0.52	4.33	6.17	71	0.52	1.83	6.00	72	0.58	3.33	4.17
31	77	0.57	3.50	8.67	80	0.63	7.00	4.83	83	0.61	5.33	7.67	94	0.55	3.50	11.83	93	0.58	4.50	9.50
32	62	0.57	6.83	2.17	64	0.52	6.00	2.00	64	0.54	9.67	1.83	64	0.59	7.83	4.00	62	0.56	6.67	5.67
33	72	0.54	8.17	5.67	70	0.54	6.67	5.17	72	0.54	8.50	6.00	69	0.57	7.33	4.67	64	0.56	5.50	1.50
34	81	0.52	7.17	11.17	83	0.51	4.83	10.17	73	0.54	7.33	8.33	73	0.54	8.00	8.33	90	0.49	4.00	14.17
35	65	0.59	10.67	0.33	60	0.64	12.00	0.00	70	0.57	9.67	0.33	67	0.57	9.83	0.33	64	0.62	9.17	0.00
36	77	0.48	3.67	6.83	90	0.42	5.50	8.00	85	0.45	8.50	5.83	81	0.43	5.67	7.17	68	0.58	10.50	2.50
37	67	0.64	8.83	3.83	67	0.54	9.00	4.33	74	0.57	8.83	4.67	71	0.60	7.33	4.50	64	0.63	7.67	4.50
38	70	0.54	2.67	5.67	70	0.53	9.33	4.83	67	0.54	8.83	2.33	85	0.43	6.83	5.00	75	0.50	3.83	7.83
39	74	0.52	10.00	1.17	72	0.52	10.00	3.00	74	0.52	10.00	1.17	74	0.52	10.00	1.17	69	0.53	10.17	0.50
40	73	0.48	8.17	3.83	75	0.47	6.83	4.17	75	0.47	7.67	4.17	75	0.47	6.83	4.17	74	0.50	3.33	4.83
41	93	0.48	5.33	7.17	98	0.45	9.17	9.83	92	0.49	5.50	6.67	80	0.51	4.50	5.67	92	0.48	9.17	8.00
42	69	0.51	4.67	3.33	76	0.53	4.50	5.17	68	0.51	4.67	3.50	73	0.50	7.00	5.50	71	0.53	5.33	3.67
43	87	0.49	8.67	5.83	81	0.53	10.67	2.83	79	0.44	10.00	4.17	75	0.49	7.83	3.00	73	0.51	10.50	1.50
44	80	0.45	6.33	9.83	85	0.46	5.17	5.00	82	0.48	4.33	3.00	85	0.46	5.17	5.00	89	0.45	5.17	4.67
45	77	0.60	8.33	7.50	73	0.62	6.00	6.17	72	0.64	8.17	6.00	77	0.61	7.50	5.83	69	0.63	5.17	4.17
46	92	0.56	6.67	5.33	94	0.59	5.83	4.00	98	0.57	10.67	6.00	80	0.59	9.33	1.50	78	0.60	7.83	4.83
47	66	0.48	7.83	4.17	66	0.53	8.67	4.00	66	0.53	8.67	4.00	66	0.53	8.67	4.00	66	0.50	5.83	3.17
48	64	0.53	9.00	3.67	64	0.53	9.00	3.67	75	0.51	7.83	4.00	66	0.52	8.50	5.50	85	0.45	8.00	7.67
49	96	0.47	11.50	3.00	98	0.46	11.17	2.33	90	0.47	5.50	2.33	83	0.53	10.83	1.67	98	0.48	10.67	2.67
50	88	0.49	7.33	7.83	83	0.50	4.00	7.00	81	0.51	4.17	8.67	83	0.52	4.00	7.00	86	0.53	2.83	4.83
Average	77.66	0.53	6.82	5.29	76.9	0.53	6.98	5.08	76.72	0.53	7.19	5.18	77.02	0.53	6.96	5.79	76.48	0.55	6.72	5.05

Table 97: results of agents on generated problems

Next, problems with the same values but not specifically trained on were tested. Results are given in Table 98.

dynamic	agent 1				agent 2				agent 3				agent 4				agent 5			
problem	C	U	S	T	C	U	S	T	C	U	S	T	C	U	S	T	C	U	S	T
1	71	0.48	4.83	5.67	58	0.61	6.00	4.83	82	0.48	4.67	6.50	73	0.48	5.33	7.83	76	0.53	3.83	6.83
2	76	0.47	5.50	3.50	76	0.47	5.50	3.50	76	0.52	5.00	0.00	80	0.45	5.50	1.00	76	0.52	5.00	0.00
3	80	0.52	3.50	4.17	79	0.51	3.67	4.83	79	0.51	3.67	4.83	79	0.51	3.67	4.83	82	0.50	3.00	5.17
4	84	0.51	6.17	6.17	88	0.50	7.67	16.00	84	0.50	7.67	12.17	87	0.48	8.17	12.50	85	0.51	6.67	9.83
5	91	0.52	4.50	11.17	85	0.53	4.50	8.50	91	0.51	5.83	10.83	90	0.52	6.33	9.00	80	0.52	6.17	10.67
6	89	0.55	9.50	4.50	96	0.49	9.33	6.17	82	0.56	9.50	3.83	82	0.53	9.33	8.33	88	0.53	6.00	4.67
7	60	0.57	5.67	8.00	60	0.59	5.67	8.33	59	0.58	6.17	6.17	69	0.51	6.83	10.17	74	0.55	3.83	3.50
8	58	0.56	6.33	2.83	63	0.52	6.00	5.17	61	0.54	6.67	4.00	61	0.54	6.67	4.00	60	0.55	7.17	3.83
9	69	0.52	8.17	5.50	70	0.50	8.17	4.83	84	0.46	8.00	8.17	74	0.51	8.67	5.67	74	0.47	6.83	5.83
10	93	0.50	2.33	8.00	80	0.54	3.83	7.33	76	0.55	4.33	8.33	82	0.52	4.33	7.00	95	0.46	3.83	15.33
11	79	0.58	8.17	5.00	79	0.59	5.67	4.67	79	0.58	8.17	5.00	79	0.58	8.17	5.00	82	0.57	8.33	2.00
12	88	0.51	11.33	4.50	81	0.57	12.50	2.33	86	0.53	9.67	3.17	76	0.55	8.50	2.83	70	0.62	6.17	2.50
13	91	0.47	5.50	9.83	72	0.57	2.50	3.67	72	0.54	3.33	4.67	63	0.62	5.50	1.17	63	0.60	3.67	1.67
14	66	0.50	11.50	6.17	56	0.56	10.33	2.00	64	0.51	8.17	2.50	57	0.51	5.50	4.17	55	0.53	5.33	1.00
15	91	0.48	5.33	6.67	79	0.57	8.67	3.17	72	0.58	8.83	3.17	79	0.50	5.33	5.83	85	0.53	7.00	2.50
16	64	0.58	12.67	0.50	64	0.58	12.67	0.50	73	0.49	9.00	4.67	61	0.61	12.00	5.67	66	0.57	12.50	0.83
17	77	0.54	12.00	9.83	90	0.46	7.00	6.50	77	0.55	12.00	7.33	73	0.52	8.67	2.00	88	0.44	8.17	7.17
18	82	0.48	8.17	9.33	89	0.41	9.33	14.83	82	0.48	8.17	9.67	74	0.52	6.83	5.17	78	0.55	2.83	7.83
19	82	0.53	11.83	4.17	86	0.49	9.17	1.67	65	0.60	11.83	1.33	75	0.55	12.67	7.00	74	0.53	4.67	0.50
20	66	0.64	6.83	2.67	68	0.62	5.50	2.33	68	0.62	5.33	4.67	73	0.57	6.33	7.83	68	0.62	5.33	4.67
21	78	0.50	7.67	2.00	82	0.49	5.67	2.67	79	0.48	7.33	3.33	78	0.50	7.67	2.00	85	0.46	8.67	2.83
22	64	0.52	8.83	3.17	56	0.56	10.00	4.33	70	0.56	8.33	0.67	61	0.53	8.00	6.00	62	0.59	10.33	1.17
23	77	0.47	4.17	11.17	77	0.47	4.17	11.17	75	0.47	4.17	10.17	81	0.47	4.17	11.50	72	0.56	5.17	4.00
24	63	0.50	11.33	2.17	68	0.47	12.17	4.67	77	0.47	9.33	2.17	70	0.46	11.83	4.33	62	0.50	11.33	3.67
25	77	0.59	7.33	3.67	91	0.53	3.17	10.00	92	0.53	6.50	8.00	85	0.53	5.33	10.33	99	0.50	4.83	12.67
26	76	0.51	5.17	3.50	70	0.56	5.50	6.00	71	0.53	5.17	2.33	75	0.48	3.67	5.67	71	0.59	5.83	1.67
27	78	0.48	10.50	6.83	71	0.51	9.17	6.00	79	0.49	10.33	6.00	78	0.47	10.67	6.83	73	0.54	7.67	3.00
28	84	0.50	5.67	3.17	83	0.49	5.67	3.17	79	0.54	7.00	2.33	78	0.50	6.50	4.17	75	0.56	6.00	2.00
29	70	0.53	4.17	4.17	74	0.50	3.83	2.67	84	0.45	4.33	4.33	74	0.48	5.00	4.67	73	0.50	5.50	3.00
30	76	0.47	5.17	1.67	59	0.54	5.67	0.33	59	0.54	5.67	0.33	59	0.52	4.83	6.33	58	0.58	4.00	4.17
31	67	0.54	8.83	2.83	77	0.49	5.83	5.00	65	0.54	7.67	0.67	67	0.54	8.33	4.17	71	0.54	6.83	6.00
32	59	0.60	5.83	1.00	59	0.60	5.83	1.00	71	0.54	7.67	3.00	77	0.53	9.83	4.67	53	0.65	7.33	1.17
33	87	0.50	4.00	7.17	75	0.52	5.83	7.17	99	0.47	4.67	5.00	85	0.47	8.00	10.33	77	0.50	2.83	6.17
34	76	0.50	4.00	6.00	73	0.50	7.83	9.50	87	0.44	2.17	9.17	88	0.44	5.83	11.50	79	0.49	4.17	6.83
35	89	0.49	4.67	3.33	78	0.52	5.00	4.17	86	0.51	6.33	1.50	95	0.44	6.00	8.33	81	0.57	9.50	6.83
36	88	0.48	7.00	5.17	75	0.51	6.17	5.83	86	0.50	6.00	6.33	75	0.51	6.17	5.83	81	0.50	8.67	4.67
37	82	0.50	4.83	5.67	67	0.58	9.50	0.67	70	0.53	10.50	4.50	81	0.52	9.33	4.67	85	0.49	9.00	5.83
38	81	0.50	10.67	8.83	84	0.46	8.33	9.67	84	0.53	9.17	4.67	73	0.56	8.83	2.50	81	0.48	7.67	6.83
39	82	0.46	10.17	7.00	89	0.43	10.17	7.67	70	0.58	10.83	2.00	88	0.43	10.17	7.33	74	0.56	8.17	4.33
40	79	0.48	6.17	5.00	74	0.49	7.67	2.67	77	0.48	6.17	3.17	74	0.49	7.67	2.67	74	0.49	6.33	2.67
41	81	0.46	5.50	7.83	93	0.48	7.83	7.33	78	0.52	5.67	6.67	93	0.47	5.33	8.50	84	0.50	5.83	3.67
42	71	0.51	8.00	2.67	63	0.63	6.33	0.33	62	0.56	7.67	1.00	71	0.53	9.00	2.67	60	0.65	7.50	0.00
43	76	0.44	13.17	1.67	77	0.46	12.83	2.50	76	0.44	10.50	1.67	76	0.44	13.17	1.67	71	0.47	11.83	0.83
44	100	0.51	2.33	6.17	92	0.55	4.33	11.50	96	0.52	7.83	4.67	90	0.53	4.50	6.83	105	0.50	1.33	8.33
45	50	0.51	10.83	2.67	50	0.52	10.33	1.83	54	0.49	8.17	1.17	52	0.49	8.50	2.00	60	0.47	9.33	2.67
46	73	0.56	8.33	3.67	77	0.56	10.83	0.50	75	0.56	6.33	3.83	72	0.56	8.50	5.00	77	0.54	4.67	4.33
47	88	0.47	4.50	11.83	82	0.43	5.67	14.50	90	0.42	5.67	14.00	82	0.44	4.50	12.83	95	0.42	4.17	18.33
48	70	0.61	14.17	2.33	82	0.59	10.67	6.33	71	0.64	12.17	1.50	70	0.60	12.67	3.83	75	0.60	10.83	4.33
49	80	0.56	9.17	0.00	91	0.47	6.83	5.50	72	0.52	7.50	6.67	65	0.56	11.50	3.17	86	0.53	11.00	0.83
50	78	0.51	4.00	7.17	76	0.46	4.50	8.67	81	0.52	4.67	6.33	84	0.44	2.67	7.00	78	0.57	1.33	5.17
Average	77.14	0.52	7.32	5.15	75.68	0.52	7.22	5.49	76.54	0.52	7.23	4.76	75.68	0.51	7.45	5.89	75.92	0.53	6.48	4.69

Table 98: Results of similar problems not trained on

Finally, problems with different processing times, again U[20,99] were tested. Results are given in Table 99.

dynamic	agent 1				agent 2				agent 3				agent 4				agent 5			
problem	C	U	S	T	C	U	S	T	C	U	S	T	C	U	S	T	C	U	S	T
1	661	0.62	1.33	167.00	683	0.66	1.33	145.50	823	0.57	1.33	172.50	799	0.65	7.50	175.00	656	0.68	1.33	136.00
2	906	0.48	13.17	201.33	928	0.52	13.17	223.33	675	0.63	1.17	162.00	722	0.53	13.17	214.00	675	0.62	1.33	191.83
3	597	0.72	1.83	58.17	697	0.64	1.67	144.17	670	0.64	5.17	123.17	699	0.65	11.33	129.33	607	0.68	7.50	89.83
4	842	0.54	14.67	219.50	680	0.76	2.67	173.17	788	0.68	4.00	152.50	984	0.49	14.67	247.00	838	0.58	0.00	163.17
5	721	0.63	11.67	137.67	685	0.63	11.67	144.17	893	0.55	11.67	189.00	747	0.60	11.67	173.50	685	0.60	0.00	201.50
6	819	0.58	4.00	197.17	671	0.63	4.00	136.17	764	0.61	4.00	153.83	890	0.53	15.00	172.33	693	0.61	4.00	133.00
7	935	0.47	9.50	221.67	926	0.58	2.67	201.33	823	0.50	9.50	225.50	792	0.58	9.50	167.17	783	0.56	0.00	261.50
8	822	0.52	6.50	165.33	845	0.49	12.83	158.67	824	0.51	12.83	166.50	824	0.51	12.83	166.50	845	0.49	12.83	158.67
9	832	0.55	13.83	246.00	713	0.60	5.33	200.83	904	0.52	13.83	241.17	847	0.54	13.83	241.83	897	0.55	4.00	214.17
10	827	0.54	17.00	172.33	827	0.54	17.00	188.67	618	0.65	0.00	131.67	812	0.52	17.00	186.17	838	0.50	12.17	207.83
11	680	0.61	6.67	151.50	732	0.60	1.33	185.83	814	0.56	6.67	198.33	794	0.54	10.33	179.50	758	0.59	1.33	195.33
12	710	0.59	9.83	158.67	708	0.59	9.83	138.17	825	0.56	9.83	148.67	708	0.59	9.83	138.17	759	0.57	4.00	146.83
13	653	0.58	18.83	159.83	643	0.58	13.33	132.83	820	0.51	18.83	227.83	759	0.52	18.83	181.50	720	0.56	2.67	197.00
14	796	0.55	13.83	228.83	877	0.55	13.83	210.50	902	0.49	4.00	229.17	809	0.54	13.83	220.83	882	0.51	13.83	264.17
15	796	0.64	1.33	224.00	885	0.53	0.00	222.00	808	0.62	1.33	204.17	843	0.62	17.67	199.50	714	0.65	0.00	204.33
16	693	0.58	6.00	122.50	737	0.54	6.00	150.33	815	0.54	4.00	190.17	701	0.58	6.00	146.67	670	0.59	6.00	139.33
17	697	0.58	12.67	164.83	697	0.57	12.67	161.67	697	0.55	12.67	192.00	803	0.51	14.17	184.33	760	0.56	1.33	186.17
18	894	0.55	4.00	165.83	777	0.61	5.67	135.33	790	0.60	4.00	165.50	777	0.61	5.67	135.33	843	0.59	5.67	145.83
19	864	0.51	10.50	253.00	860	0.50	10.50	250.17	680	0.66	10.50	165.50	863	0.51	10.50	247.83	797	0.52	10.50	241.50
20	917	0.50	6.67	233.67	837	0.51	6.67	236.00	791	0.55	4.00	192.50	917	0.50	6.67	224.33	717	0.59	1.33	146.50
21	639	0.60	6.17	171.83	639	0.59	6.17	176.50	745	0.56	0.00	185.00	659	0.58	6.17	169.50	690	0.56	6.17	187.50
22	759	0.64	16.33	158.50	759	0.63	16.33	176.17	703	0.58	13.83	188.67	798	0.60	16.33	167.00	759	0.61	16.33	176.17
23	793	0.58	16.67	168.67	623	0.62	16.67	149.00	638	0.61	16.67	178.67	663	0.63	16.67	142.17	674	0.65	16.67	159.33
24	707	0.62	14.50	141.67	652	0.60	14.50	141.17	752	0.60	4.00	149.00	712	0.61	4.50	169.17	707	0.62	14.50	141.67
25	756	0.55	15.83	198.50	771	0.52	11.83	245.33	752	0.57	11.00	203.17	870	0.52	11.83	229.17	840	0.55	6.33	174.33
26	651	0.57	4.00	139.33	598	0.59	2.67	149.83	773	0.50	0.00	234.83	671	0.57	5.33	145.17	645	0.57	2.67	137.83
27	719	0.66	7.67	149.83	691	0.64	8.00	189.50	629	0.68	7.67	135.83	697	0.60	17.67	211.83	713	0.64	0.00	198.17
28	814	0.56	15.67	193.00	763	0.57	15.67	211.67	802	0.58	1.33	180.33	747	0.58	15.67	179.67	680	0.68	0.33	131.67
29	649	0.62	10.33	158.17	775	0.57	2.67	179.50	747	0.60	10.33	131.83	657	0.61	10.33	162.17	657	0.61	10.33	162.17
30	820	0.55	16.50	243.50	704	0.56	19.17	154.50	696	0.58	2.67	142.83	751	0.57	2.67	156.00	688	0.62	0.00	248.17
31	851	0.59	0.00	273.50	830	0.62	0.00	268.33	913	0.55	4.00	264.17	924	0.57	13.00	211.33	824	0.65	4.17	239.17
32	655	0.59	19.17	158.50	655	0.59	19.17	150.50	652	0.60	21.83	159.50	667	0.61	19.33	129.50	655	0.59	19.17	150.50
33	672	0.64	18.83	164.33	698	0.60	18.67	163.83	674	0.62	16.83	152.17	675	0.64	18.83	158.83	701	0.62	17.67	190.17
34	738	0.59	2.17	171.83	831	0.56	2.17	216.33	738	0.59	2.17	168.67	895	0.54	9.00	268.33	814	0.57	0.00	207.17
35	931	0.47	1.83	213.33	802	0.54	4.00	182.50	880	0.51	1.83	187.83	772	0.55	14.67	171.83	669	0.61	1.83	140.50
36	835	0.47	7.00	250.33	837	0.49	7.00	236.50	918	0.47	4.00	207.33	906	0.45	4.00	235.33	750	0.55	2.00	164.33
37	649	0.63	16.00	151.67	749	0.64	16.00	134.33	719	0.56	4.00	161.50	749	0.64	16.00	134.33	593	0.68	8.67	112.83
38	676	0.63	0.00	171.83	632	0.64	0.00	147.33	689	0.62	0.00	184.17	824	0.52	16.33	209.33	637	0.64	0.00	147.50
39	677	0.63	21.00	114.00	714	0.58	21.00	113.50	714	0.56	18.33	167.67	714	0.56	21.00	144.33	631	0.65	6.67	121.83
40	632	0.59	12.83	142.17	594	0.61	12.83	168.50	643	0.56	0.00	183.00	697	0.53	14.17	174.33	674	0.62	1.33	146.67
41	777	0.54	17.17	195.33	935	0.49	17.17	217.83	897	0.49	4.00	208.67	807	0.53	17.17	237.00	855	0.60	4.00	165.83
42	817	0.51	11.17	218.67	765	0.53	11.17	190.33	811	0.51	11.17	204.50	794	0.52	4.00	184.17	704	0.55	11.17	204.83
43	791	0.57	14.33	182.17	682	0.63	4.00	161.00	616	0.59	14.33	150.50	687	0.62	14.33	144.33	723	0.52	0.00	213.33
44	781	0.54	0.00	172.83	805	0.54	1.33	177.17	752	0.55	1.33	172.33	802	0.53	7.00	158.17	762	0.57	0.00	161.83
45	737	0.62	17.33	203.00	752	0.64	17.50	159.83	752	0.64	17.50	173.33	690	0.67	17.50	149.33	786	0.64	0.83	183.00
46	886	0.62	4.00	174.67	850	0.63	4.00	204.50	816	0.66	20.33	181.83	793	0.67	20.33	169.67	861	0.65	4.00	188.83
47	676	0.53	9.83	163.83	763	0.52	9.83	209.50	765	0.53	9.83	168.83	770	0.50	9.83	187.83	649	0.55	1.33	151.50
48	708	0.57	4.00	156.83	773	0.51	4.00	204.50	708	0.57	4.00	156.83	752	0.53	8.50	196.83	646	0.57	4.00	184.50
49	965	0.51	18.33	218.33	830	0.48	18.33	216.00	696	0.59	18.33	153.50	903	0.54	18.33	160.17	696	0.63	18.33	152.33
50	675	0.64	9.67	132.00	748	0.57	4.00	247.33	833	0.54	4.00	286.83	712	0.64	9.67	164.00	737	0.62	0.00	225.17
Average	761.96	0.58	10.24	180.02	752.56	0.58	9.16	181.63	762.94	0.57	7.69	181.10	776.96	0.57	12.40	181.03	731.14	0.60	5.37	175.87

Table 99: Results of similar problems with processing time distribution U[20,99]