Msc Thesis
A Multi-Agent Reinforcement Learning Approach for the Hybrid Flow Shop Problem
Applied to a hospital case

Anne Maria Leonie Tacken (1264952)

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Business Analytics and Operations Research

Tilburg School of Economics and Management
Tilburg University

Supervised by:
dr. J.C. Wagenaar
M. Peters MSc - Pipple

January 20, 2021

1 Management Summary

The Operation Theater (OT) is the most costly resource in the hospital. Therefore it is often the focus point of research. The recent Covid-19 outbreak was characterized by capacity shortages in hospitals. This stresses the importance of also including other resources in the hospital planning. In this thesis a three-stage No-Wait Hybrid Flow Shop Problem (HFSP) formulation is proposed for modelling the OT hospital flow. Next to the OT also the Intensive Care (IC) and the Medium Care (MC) department are included.

Deep Reinforcement Learning (DRL) techniques have proven itself to be very efficient. Multi Agent Reinforcement Learning (MARL) is a particular area of DRL, in which autonomous agents learn to make successful decisions in a joint environment. A literature review is presented on Hospital scheduling and Reinforcement Learning methods for solving scheduling problems. What is noted is that literature and applications of MARL applied to a HFSP and in particular, with a application to hospital scheduling, are scarce. This research aims to fill this gap.

The core of this thesis focuses on building a Multi-Agent Reinforcement Learning Framework for the general HFSP. By means of a case study it is shown that, by applying this framework, solutions with reasonably good objective values are obtained, when compared to a benchmark solution method. The latter uses a state-of-the-art commercial solver. In addition, it is illustrated that this MARL framework can obtain reasonably good results for new similar problem instances instantly. These problem instances are "new" in the sense that they were not used in the training process. Yet, using experience obtained based on the training instances, a good solution for these new instances can be produced practically instantaneously, without going through any additional learning or optimization. This feature is very promising in cases where a new scheduling solution may be required in a very short amount of time.

It is shown that this MARL formulation for the general HFSP can be extended such that it is also applicable to the three-stage No-Wait HFSP which was proposed to model the OT hospital planning. For this purpose, the cost function was adjusted such that the no-wait constraint is enforced. Also for this particular problem, reasonably good solutions are obtained, and the generalization capabilities of the framework are illustrated.

This work presents results which support the claim that the MARL algorithm has potential to be a good solution method for the HFSP. Especially the generalization property has potential benefits in fields where a new solution is needed rapidly. Future work on appropriate action selection methods in the multi agent setting of this research is still required.

2 Acknowledgements

First of all I want to express my gratitude towards dr. Joris Wagenaar for the guidance, assistance, and the many meetings over the course of this thesis. Our meetings always helped me a lot to structure my work and to remain the overview. Every time after we had a (virtual) meeting I felt like I had new energy and ideas and wanted to get back to work right away.

I would also like to acknowledge Maurice, my supervisor from Pipple, who has helped me through this thesis process thanks to our weekly meetings. His critical notes and sharp questions really improved the quality of this work. Likewise, I want to thank all of my colleagues at Pipple. I am grateful for all the help and support I received of various colleagues during this thesis. Even though I spent a total of only three days at the office during this internship, I still felt part of Pipple. Especially, the few real-life Pipple Days during this summer, were a highlight. The enthusiasm of the Pipple colleagues on work-related matters, especially in the field of Reinforcement Learning, really sparked my interest. A special word of thanks for Lennart, who was always available for questions and for sparring and who was able to structure my thoughts when I could not. I also want to thank my colleagues and dear friends Masum and Nina, who have provided valuable feedback and laughter along the way.

I would also like to acknowledge dr. Yasemin Merzifonluoglu for being the second reader of this work, and for taking the time to review this lengthy thesis.

Finally, last but not least, a big thank you to all my family and friends for their support. Especially to Bart, who during these special times of working at home, has been very involved and supported me in any kind of way.

Contents

1 Management Summary **1**

2 Acknowledgements **2**

3 List of Abbreviations **7**

4 Introduction **8**

5 Literature Review **10**

5.1 Planning problems in a hospital 10

5.1.1 Different Types of Patients 11

5.1.2 Up- and Downstream Facilities 11

5.1.3 Objectives 12

5.1.4 Type of Scheduling 13

5.1.5 Uncertainty: Deterministic vs Stochastic 14

5.1.6 Mathematical Models 14

5.1.7 Solution Methods 15

5.2 Hybrid Flowshop Scheduling Problem 15

5.2.1 General Introduction HFSP 16

5.2.2 Mathematical Problem Description of HFSP 16

5.2.3 Solving a HFSP to Optimality 18

5.2.4 Naming Scheduling Problems 18

5.3 Reinforcement Learning for Scheduling Problems 19

6 Reinforcement Learning **21**

6.1 Concept of Reinforcement Learning 21

6.1.1 Markov Decision Processes . . . 22
6.1.2 Policy . . . 24
6.1.3 Terminal state . . . 24
6.1.4 Episodic and Continuing Tasks . . . 25
6.1.5 Rewards . . . 25
6.1.6 The Value Function . . . 27
6.1.7 The Bellman Equation . . . 27
6.1.8 Q-values . . . 30
6.1.9 Q-Learning using Q-Tables . . . 31
6.2 Learning in More Complex, Real-Life Settings with Uncertainty . . . 34
6.2.1 Size of the Learning Rate α . . . 36
6.2.2 Action Selection Methods and the Exploration-Exploitation Trade-off . . . 37
6.2.3 Size of the Discount Rate γ . . . 38
6.3 Multi-Agent Reinforcement Learning . . . 38
6.3.1 Joint vs. independent learners . . . 39
6.3.2 Markov Game . . . 39
6.3.3 Challenges and Pitfalls of MARL . . . 40
6.4 Deep Reinforcement Learning . . . 42
6.4.1 Deep Q-learning . . . 43
6.4.2 Multi-Agent DQL . . . 44

7 Multi-Agent Reinforcement Learning Framework for a HFSP 45

7.1 Problem Introduction . . . 45
7.2 Single Agent Setting . . . 47
7.2.1 Agent . . . 47
7.2.2 States . . . 47

7.2.3 Actions . . . 48
7.2.4 Reward system . . . 50
7.2.5 State Transitions . . . 50
7.2.6 Greedy Solution . . . 51
7.2.7 Implementation of Q-learning Algorithm in Case of 1 Learning Agent . . . 52
7.2.8 Learning Under Uncertainty . . . 56
7.3 Framework for the Multi-Agent Setting . . . 58
7.3.1 Learning in a Multi-Agent Setting . . . 60
7.3.2 Agents in Consecutive Departments . . . 61
7.3.3 Agents Within the Same Department . . . 61
7.3.4 4 Learning Agents . . . 63
7.4 Summary . . . 63

8 Hospital planning problem formulated as HFSP **64**
8.1 Hospital Scheduling setting . . . 64
8.1.1 Problem Setting . . . 64
8.2 Formulation of Hospital Flow using a HFSP . . . 66
8.3 Data Generation . . . 69
8.3.1 Standard Scheduling Notation . . . 70

9 Case Study: Solving the HFSP **71**
9.1 Benchmark Method: Solving the HFSP MILP . . . 72
9.2 Implementation of the MARL Solution Approach for the HFSP . . . 73
9.2.1 Deep Q-learning Algorithm . . . 73
9.3 MARL Learning Statistics . . . 75
9.4 Solution Quality and Generalisation Capability . . . 76

9.5 Summary and Discussion . . . 78
9.6 Limitations and Suggestions for Future Research . . . 79

10 Case Study: Solving the Hospital planning problem **81**

10.1 Implementation of the MARL Framework in a Hospital Setting . . . 81
10.1.1 Penalizing Based on Patients in the Queue . . . 81
10.1.2 Adjusting the Release Moment to the Availability in the Subsequent Department . . . 83
10.1.3 Focus on Exploring Feasible Solutions . . . 85
10.2 Implementation and Results . . . 85
10.3 Summary and Discussion . . . 86
10.4 Limitations and Suggestions for Future Research . . . 88

11 Conclusion **89**

11.1 MARL for the HFSP . . . 89
11.2 MARL for a Hospital Scheduling Problem . . . 90
11.3 Recommendations for Future Research . . . 90

A Data Tables **92**

B Regular Q-Learning Algorithm **93**

C Patient Planning 1 learning agent **94**

D Q-Table for 1 learning agent under uncertain state transitions **95**

E Pseudocode for Multi-Agent Deep Q-Learning algorithm **96**

References **98**

3 List of Abbreviations

OT Operation Theater

RL Reinforcement Learning

IC Intensive Care

MC Medium Care

HFSP Hybrid Flowshop Scheduling Problem

PACU Post-anesthesia care unit

MDP Markov Decision Process **ML** Machine Learning

MAS Multi-Agent System

NN Neural Network

FNN Feed-forward Neural Network

MC Monte Carlo

TD Temporal Difference

MAS Multi-Agent System

4 Introduction

The operation theatre (OT) is the most costly resource in a hospital, and accounts for 40% of all hospital costs. Since the OT is the hospital's largest cost center, it has a big influence on the performance of the hospital as a whole. This makes the OT the focus point of many studies. In particular, the focus lies on effective and adequate scheduling procedures for the OT (Cardoen, Demeulemeester, & Beliën, 2010). The recent Covid-19 outbreak was characterized by worldwide capacity shortages in hospitals. Bottlenecks were not the OT department but instead the IC capacity and the availability of nursing hours. This stresses the importance of including not only the OT department but also other departments in hospital scheduling research. Samudra et al. (2016) note that a way to avoid cancellations of surgeries is to level the utilization of units up- and downstream of the OT. "For example, an overutilized Post-anesthesia care unit (PACU) can block the OT, therefore prohibiting patients who have already completed surgery from leaving it. A blocked OT will impact succeeding elective surgeries, as they are either delayed or canceled. This situation can be avoided if the OT schedule is constructed in a way that the utilization of the units connected to the OT are leveled." Existing work did model the OT scheduling problem as a two stage Hybrid Flow Shop Problem (HFSP). Where the OT was stage 1 and the PACU was stage 2. In this work, we wish to model the full OT hospital flow, by including also the IC and MC department. Therefore, we propose a three-stage no-wait HFSP, where next to the OT, also the IC and MC department are included. We want to stress the addition of the no-wait property because obviously a patient cannot wait in a queue before moving to the next department.

Deep Reinforcement Learning (DRL) methods have shown many successful implementations in recent years. DRL is a combination of Reinforcement Learning (RL) and deep learning techniques. Very recently DRL agents have learnt to play difficult games like Dota and Starcraft. It has been shown, that for all of these games, the RL agent has been able to achieve superhuman performance. RL is based on the natural learning mechanism of animals. In RL, an agent is not told how to behave but instead he will discover what behaviour is desirable. The goal of the agent is to learn how to behave by interacting with the environment and by maximizing a numerical reward structure. The main advantage of DRL techniques is its ability to generalize from experience, with its ability to scale to problems with high-dimensional state and action spaces. This implies that when faced with a new instance of the problem, the agent can use past experience to determine the best behaviour in that setting instantly and without the need of optimizing.

A colony of ants have an innate ability to communicate, coordinate and collaborate effortlessly. The potential of single agent DRL systems has already been shown, now imagine the possibilities if a set of agents could do the same. Multi-Agent Reinforcement Learning (MARL) is the field of study which is concerned with developing Deep Reinforcement Learning Techniques and algorithms that enable a set of autonomous agents to make successful decisions in a shared environment (Kouridi, 2020).

The main research questions which we aim to answer in this thesis are:

- Can we design a MARL algorithm which can provide solutions for a general HFSP?
- Can the MARL algorithm be utilized for solving the hospital scheduling problem?

By answering these research questions we get to the main contributions of this thesis. This main contribution is the design of a MARL framework for the general HFSP. By means of a case study, we show that by applying this framework, solutions to problem instances that are within a reasonable range of benchmark solutions are obtained, where the latter use a state-of-the-art commercial solver. Moreover, we illustrate that reasonably good solution for new problem instances can be generated instantly, using the MARL approach. These instances were not used in the learning process. This feature is promising in cases where good solutions are required in a very short amount of time.

In addition, a Three-Stage No-Wait HFSP is proposed for the hospital OT scheduling problem. It is shown how this MARL framework can be extended such that it can comply with this No-wait constraint, such that framework can also be applied to this special case of the HFSP. This constraint is enforced through adjusting a cost function. For this end, different cost functions will be discussed, and an advice is formed based on a case study.

This thesis was conducted in cooperation with Pipple, a consultancy firm specialized in the field of data science solutions. One of their customer areas is the health care sector. The frequent occurrence of HFSP in practice, combined with Pipple's interest in exploring innovations and promising new techniques, like Reinforcement Learning, constituted the motivation behind this research.

The outline of this work is as follows: Chapter 5 presents existing literature on hospital scheduling problems and on reinforcement learning techniques applied to scheduling problems. Additionally, it introduces the hybrid flow shop problem. Chapter 6 aims to provide the reader with a theoretical background on reinforcement learning. In Chapter 7 we propose our MARL framework for a general HFSP. In Chapter 8 a HFSP formulation for the hospital scheduling flow is provided. Chapter 9 presents results on the performance of the proposed MARL framework on the general HFSP, where the no-wait constraint is disregarded. Finally, in Chapter 10 the MARL approach is applied to the hospital scheduling problem. To conclude, Chapter 11 presents results to the research questions and contains recommendations for future research.

5 Literature Review

The purpose of this chapter is to review existing literature on hospital scheduling problems, hybrid flow shop problems (HFSP) and on reinforcement learning methods applied to scheduling problems. While reviewing existing literature, a literature gap will be identified with regards to the formulation of the hospital scheduling flow as a hybrid flow shop. Additionally, we will see that there barely exists literature on the use of RL techniques, in particular MARL techniques, for hybrid flow shop scheduling. Later in this thesis, we will attempt to fill this gap by proposing a three-stage HFSP to model the OT hospital flow. Then, we wish to research whether MARL techniques can be used to solve a general HFSP model, and in particular, a hospital OT flow modeled as a HFSP.

First in Section 5.1 a general introduction about planning problems in a hospital will be provided. Most existing work focuses on the planning of the Operation Theater (OT). Scheduling problems involving the OT involve lots of the different aspects. In this thesis we will also consider an OT scheduling setting. Therefore, aspects which are relevant and should be taken into account when conducting research on OT are introduced and elaborated upon in the remainder of section 5.1. Subsequently Section 5.2 will present literature and the mathematical formulation on the HFSP. Additionally, 5.2.4 introduces the naming convention which is used to classify scheduling problems. Finally, Section 5.3 discusses related work on reinforcement learning for scheduling problems.

5.1 Planning problems in a hospital

An operating hospital is the result of various resource schedules. Planning of the patients, the medical specialists, the nurses and all required material should be adjusted to each other and must all come together in order to have a well-functioning hospital. This makes a hospital a complex and interesting environment with many stakeholders. Looking at the existing literature on hospital planning, a statement that is commonly found is that "the Operation Theater (OT) is the most costly resource in the hospital", that accounts for 40% of hospital costs (Macario, Vitez, Dunn, & McDonald, 1995). This makes the OT the focus point of many studies. In particular, the focus lies on effective scheduling and planning of the OT. The OT is the facility in a hospital where surgical operations are carried out.

This section will elaborate upon the different aspects of OT planning. For this end, the framework presented by Samudra et al. (2016) for classifying existing OT literature will be used to introduce the relevant aspects. These relevant features are: The different types of patients; The up- and downstream facilities; the Objectives; the type of scheduling; Uncertainty; and the mathematical models and solution methods used. When conducting research in the field of OT scheduling, the researcher should be aware of these seven fields and clearly report the choices and assumptions made. In the remainder of this section we will elaborate upon the different aspects of OT planning. By defining the use case of this thesis by means of a similar framework, it will be clear where this research fits within the existing literature. Also, it will make sure that every relevant field is included and thought about. Readers interested in an detailed and schematic overview of existing literature on hospital planning are referred to Samudra et al. (2016).

5.1.1 Different Types of Patients

There are two different types of patients: elective patients and non-elective patients. Elective patients are patients whose surgery can be planned in advance. These patients are usually on a waiting list for a specific surgery. The amount and type of patients waiting for a certain treatment is thus known upfront. Therefore the surgeries of these elective patients can be scheduled. Non-elective patients are patients that have to be scheduled on very short notice after their arrival. Amongst non-elective patients, two different types can be distinguished: emergency patients and urgent patients. Urgent patients are patients whose surgery can be delayed for a short period (i.e. days). Next there are emergency patients, who have to be scheduled right away. Studies deal with non-elective patients in different ways.

Most hospitals deal with emergency patients by reserving some OT capacity for emergency patients (van Essen, Hans, Hurink, & Oversberg, 2012). Some hospitals reserve one or multiple OTs specifically for emergency patients, others schedule the emergency patients in an OT for elective patients, sometimes leading to cancellations of elective patients. There are also hospitals that use a combination of the two. Then there is another distinction that can be made: both elective and non-elective patients can be divided in inpatients and outpatients. Inpatients are patients that are hospitalized and have to stay overnight. Whereas outpatients are patients that typically enter and leave the hospital on the same day. The arrival time of outpatients is uncertain too since they are not yet present in the hospital, and could be late for their operation. However, this type of uncertainty is not as uncertain as the arrival of non-elective patients.

Van Riet and Demeulemeester (2015) discuss the trade-offs in OT planning for elective and non-elective patients. They do so by presenting a review of existing literature on OT planning where both emergency and elective patients are involved. The reviewed papers differ in terms of the amount of capacity that is reserved for the different patient types. The main trade-offs are in terms of overtime, utilization, schedule disruptions and waiting time for elective and emergency patients. It is argued that the research setting and assumptions vary widely in the existing literature which makes it difficult to compare existing papers. Van Riet and Demeulemeester (2015) conclude that there is no complete answer yet to the question which policy choice, for incorporating non-electives in the OT schedule, is most appropriate.

5.1.2 Up- and Downstream Facilities

The OT might be the most costly aspect in an hospital planning. However, an hospital exists of many other vital facilities which are extremely important for a well functioning hospital planning. The OT planning and scheduling influences many other departments throughout the hospital. Dellaert and Jeunet (2010) argued that the inclusion of up- and downstream facilities in one's model is crucial. The authors mention situations where surgeries have to be cancelled because of shortages in resources different from the OT. Therefor it does not suffice to only focus on the OT. Hence the authors included bed capacity and nursing hours at the IC and MC in their model. When talking about up- and downstream facilities in the remainder of this thesis, then these refer to the MC and IC care which a patients might require before and after the OT.

The importance of paying attention not only to the OT capacity but to these other resources was stressed by the recent COVID-19 outbreak. During the first Covid-19 wave in the Netherlands, the IC capacity was the main bottleneck. Where in the second wave, the nursing capacity is the new main bottleneck (Visser, 2020). Some OT planning researches include up- and down stream facilities in their model, whereas others only focus on the OT planning itself. This illustrates that including up- and downstream facilities in the model makes way for investigating the combined performance, instead of just focusing on one element. When other vital facilities and resources are ignored in a research, it is very likely that improved performance of the OT leads to a decline in the performance of other facilities. Therefore, results presented about the OT performance may not represent the overall performance of the complete hospital accurately. This stresses the importance of including these facilities in one's research.

5.1.3 Objectives

What is the goal when making an OT schedule? What makes a schedule a good one? These are questions that every researcher should answer before attempting to optimize a hospital planning.

There are various performance measures that are used in literature by researchers when constructing and evaluating a model. In essence all scheduling models focus on maximizing the efficiency of the OT and/or on minimizing the costs of the resources (Zhu, Fan, Yang, Pei, & Pardalos, 2019). Here we will shortly elaborate upon some performance measures.

- Waiting time: There are multiple types of waiting time. On the one hand we have the time that patients spent on a waiting list before a surgery is performed. And on the other hand there is the waiting time within the hospital on the day of the surgery itself.
- Utilization of resources and under/over-time: This is a measurement of how efficient resources are being used. Usually expressed in how much percent of the available hours the resource is being used. Note that it is possible for a resource to run overtime while being underutilized. Utilization refers to the workload of a resource, whereas undertime and overtime include a timing aspect. (Samudra et al., 2016) group underutilization with undertime and similarly overutilization with overtime because in literature it is often unclear which of the two views is applied. Minimizing overtime is a popular objective in literature. Logically because overtime of the OT leads to dissatisfaction amongst surgical staff and high costs for the hospital as overtime hours are usually more costly. Also, overtime in the OT might lead to disruptions in other schedules: for example of downstream facilities, but also to cancellations of surgeries, in turn leading to dissatisfaction amongst patients.
- Financial measures: There are also authors that have a sole focus on financial objectives. Stanciu, Vargas, and May (2010) even go so far, that the considered objective is to maximize the expected revenues of the surgical unit. Here a patient has a higher priority if the expected revenue per unit surgery time is highest.
- Makespan: The makespan usually defines the time span between the entrance of the first patient and the finishing time of the last patient. Minimizing the make span often results in

a dense schedule. Deviations from the schedule result in complications, making the schedule infeasible and requiring big adjustments.

- Total completion time: This objective is frequently used in literature. The objective is to minimize the total completion time of all jobs combined. As opposed to minimizing the make span, using this objective means that there is an incentive to finish individual jobs as soon as possible.

Other performance measures used in literature are: leveling, idle time, throughput, preferences, and patient deferral (Samudra et al., 2016). Some studies use a multi-objective, and use combinations of the aforementioned performance measures.
An interesting and unique objective is studied by van Essen et al. (2012). Their goal is to minimize the waiting time for emergency patients. They do so by scheduling elective patients as evenly as possible over the different OTs over the day. This is important since the operation of an emergency patient can only start when an OT is either idle, or when an operation is finished. As it is not possible to interrupt an ongoing operation, the aim of van Essen et al. (2012) is to distribute the finishing times of scheduled surgeries as much over the day as possible. The objective thus focuses on a desirable situation for emergency patients, but this is achieved by the scheduling of elective patients.

An important note made by Samudra et al. (2016) is that research is lacking on how certain modelling assumptions (e.g. neglecting emergency patients) influence the particular performance measures. Such research would be extremely valuable for researchers such that they know what factors they have to include in their model.

5.1.4 Type of Scheduling

First the decision level should be determined: for who are the decisions that are made? Here there can be distinguished between: discipline level, surgeon level, patient level. When decisions are made on a discipline level, the decisions are made for a department or a medical specialty as a whole. At the surgeon level, scheduling decisions can be done for an individual surgeon or for groups of surgeons. Most papers however focus on the patient level, where scheduling decisions are made on the basis of individual patients or patient type.

Samudra et al. (2016) point out that there are two types of patient scheduling, Advance scheduling and Allocation scheduling. In advance scheduling a patient is assigned to a surgery date. This type we will call the patient-to-date scheduling. For allocation scheduling, a given surgery date for a patient is assumed. In the allocation scheduling phase the specific OT, the starting time and/or the order in which patients are treated on a specific day are then determined. This type of scheduling will be called patient-to-room-and-time scheduling.

These different types are important to keep in mind such that one can clearly specify the type of decision that needs to be made.

Some papers such as Fei, Meskens, and Chu (2010) perform a so called two-phase patient centered scheduling. Fei et al. (2010)'s goal is to design a weekly surgery schedule. To do so, first a weekly

patient-to-date schedule problem is solved. Then secondly, a patient-to-room-and-time scheduling is devised, where the sequence of all patients that are to be operated on that day are determined. This is a common scheduling fashion, as the assignment of a specific day can be more easily planned ahead than the exact time. The latter is usually specified only short before the surgery takes place. Here thus both the patient-to-date and a patient-to-room-and-time scheduling is performed, this is not necessarily the case for all papers. Some focus only on just one of the two.

5.1.5 Uncertainty: Deterministic vs Stochastic

It are not only the many different stakeholders that cause the planning of the OT to be a difficult process, Van Riet and Demeulemeester (2015) discuss that the various sources of variability cause the main complexity. This variability greatly influences the trade-off between hospital costs and patient waiting times, which are important and commonly used performance measures. Examples of sources of variability are: surgery duration, arrival of emergency patients, arrival of outpatients. Deterministic scheduling approaches ignore uncertainty and all information is assumed to be known beforehand: the number of patients, the type of patients, the OT duration, the needed recovery time in the IC and the MC. Stochastic scheduling approaches on the other hand incorporate elements of uncertainty in their modeling. Meaning that the information is not complete and will only become available once this patient is being treated.

5.1.6 Mathematical Models

Several Mathematical Models have been used in literature to formalize the OT scheduling problem. Zhu et al. (2019) present a literature review in which they distinguish between the Bin-Packing Model, the Flowshop Problem, the stochastic model and a bi-criteria model. The distinction between deterministic and stochastic problems was already discussed in the previous subsection. Moreover, in the subsection 5.1.3 it has been discussed that there are studies that consider multiple or combinations of multiple objectives. These are so-called multi-criteria models. Hence what remains is to introduce the Bin-Packing and the Flowshop problem. This will be done in the upcoming subsection with special attention for the flowshop problem as this will be important for the remainder of this thesis.

Bin-Packing Problem
In this setting the OTs are the bins and the patients represent the items that must be packed. (Fei et al., 2010) argue that, under certain hypotheses, the first phase of the two-phase scheduling (Section 5.1.4) can be regarded as a resource-constrained bin-packing problem.

Flowshop Scheduling Problem
Patients that undergo a surgery, go through several processes. Because of this property, there are studies that have described the surgery process using a flow shop.

An example of authors that have modeled the flow of patients using a flow shop is the study of Wang, Tang, Pan, and Yan (2015). Here the daily flow of patients is modeled using a two-stage no-wait hybrid flowshop. A daily schedule is studied where the authors have included multiple

OTs as department 1 and the post-anesthesia care unit (PACU) with multiple recovery beds as a department 2. The decision goal of Wang et al. (2015) is to determine what is the optimal number of open OT in department 1 and what are the optimal number of recovery beds in department 2, so as to minimize the costs of overtime and undertime hours. (Wang et al., 2015) propose a hybrid solution method by combining a new heuristic with a discrete Particle Swarm optimization algorithm to solve this problem. No-wait means that there is no waiting time between department 1 and department 2. Implying that as soon as a patient is finished in the operating phase, he is immediately transferred to a recovery bed.

The modelling of the daily patient scheduling problem by means of a no-wait hybrid flow-shop problem has also been addressed by Fei et al. (2010) and Souki (2011).

5.1.7 Solution Methods

There are many solution methods and techniques that have been used to solve the OT scheduling problem over the years: "analytical procedures, mathematical programming, dedicated branch-and-bound, scenario/sensitivity analysis, simulation and various heuristics" (Samudra et al., 2016). For most real life applications, mathematical programming formulations are too difficult to solve within a reasonable time limit. That is why heuristics come into play and are frequently used. Examples of heuristics that have been widely studied in the OT planning are: Simulated annealing, the Genetic Algorithm and Tabu search. Some studies combine several of the abovementioned methods, like Lamiri, Grimaud, and Xie (2009) who consider several stochastic optimization methods to plan elective surgeries. They present a solution method combining Monte Carlo sampling and mixed integer programming. They also test several heuristic methods from which the most efficient one proved to be Tabu search.

What is noteworthy is that the vast majority of articles that have been published do not report anything on actual implementation.

Reinforcement Learning has only been recently introduced in the domain of applied Operations Research. To the best of our knowledge, there are no existing studies that uses Reinforcement Learning (RL) methods to solve the OT planning problem. Therefore this will be the topic of this research. Reinforcement Learning will be extensively discussed in Chapter 6.

5.2 Hybrid Flowshop Scheduling Problem

Section 5.1.6 briefly mentioned existing literature that model the OT scheduling as a no-wait hybrid-flowshop problem (HFSP). As mentioned, later in this work (8 we will propose a HFSP formulation for the hospital OT flow. This model will, next to the OT, incorporate both the IC and MC department. For this purpose, the upcoming section will introduce the concept and mathematical formulation of the HFSP. It also includes references to related literature for readers that wish to learn more about the HFSP.

5.2.1 General Introduction HFSP

A HFSP consists of a series of production departments. Each department consists of multiple machines operating parallel to each other and there must be at least one department that has more than one machine. There are N jobs which all need to be processed in S departments. Each department $s \in S$ has m_s machines that work parallel to each other. Each job must visit the departments in the same order. A job might skip any number of departments, provided that it is processed in at least one of them (Ruiz & Vázquez-Rodríguez, 2010). A job will only visit each department once. There are several variants of the HFSP but all of them have to satisfy the following criteria (Emmons & Vairaktarakis, 2013):

1. The number of departments should be at least two. $S \geq 2$
2. All departments must have at least one machine. Additionally, there is at least one department that has more than one machine. It should hold that $m_s \geq 1 \quad \forall s \in S$ and $\exists\, k \in S$ s.t. $m_k > 1$.
3. All jobs are processed in the same production flow: $S_1, \ldots, S_m$. A job might skip a department provided that it is processed in at least one of the departments.

The goal is then to find a schedule for which a specified objective is optimized. There exists numerous variants of the HFSP. These variants can differ in terms of the structure of the flowshop but also in terms of the objective and in terms of the decision variables. Therefore when working with HFSPs it is important to clearly specify the HFSP that is considered. For this a naming convention for scheduling problems will be introduced in Section 5.2.4.

5.2.2 Mathematical Problem Description of HFSP

Here a mathematical formulation of the general HFSP will be provided. There is a set of N jobs that need to be processed. The objective used in this formulation is the minimization of the total completion time, which is equal to the sum of the completion times of the individual jobs. Note that this objective can be replaced by another objective depending on the problem setting and the desired goal.

The following assumptions are made:

- All jobs are available at time zero
- All jobs are treated in every department
- Set-up times are included in the processing times
- Processing interruptions (preemptions) are not allowed
- Processing times are deterministic and known upfront

- All machines within one department are the same: we are dealing with parallel machines. This implies that the treatment time for a job within a department is the same for each machine

Sets
P: set of jobs to be processed, $P = \{1, ..., N\}$, N total number of jobs
S: set of departments that a job has to be processed, $S = \{1, ..., T\}$, with T total number of departments
M_s: set of machines in department $s \in S$

Hence $|M_s|$ thus represents the capacity of department $s \in S$, i.e. the number of machines in a department s.

Parameters
d_{is}: processing time of job i in department s

Decision Variables
$$Y_{isl} = \begin{cases} 1 & \text{if job } i \in P \text{ is scheduled on machine } l \in M_s \text{ in department } s \in S \\ 0 & \text{otherwise} \end{cases}$$

$$X_{ijs} = \begin{cases} 1 & \text{if job } i \in P \text{ precedes job } j \in P \text{ in department } s \in S \\ 0 & \text{otherwise} \end{cases}$$

$C_{i,s}$: completion time of job $i \in P$ at department $s \in S$, and let $C_{i,0} = 0$

Let $Q \geq \sum_{i,s} |M_s| d_{is}$ be an arbitrarily large number. Then the model can be formulated as follows:

$$\min \sum_{i \in P} C_{i,T} \tag{1}$$
$$st. \sum_{l \in M_s} Y_{isl} = 1 \qquad \forall s \in S, i \in P \tag{2}$$
$$C_{is} - C_{i,s-1} \geq d_{is} \qquad \forall i \in P, s \in S \setminus \{1\} \tag{3}$$
$$Q(2 - Y_{isl} - Y_{jsl} + X_{ijs}) + C_{is} - C_{js} \geq d_{is} \qquad \forall i, j \in P, \forall l \in M_s, \forall s \in S \tag{4}$$
$$Q(3 - Y_{isl} - Y_{jsl} - X_{ijs}) + C_{js} - C_{is} \geq d_{js} \qquad \forall i, j \in P, \forall l \in M_s, \forall s \in S \tag{5}$$
$$Y_{isl} \in \{0, 1\} \qquad \forall i \in P, s \in S, l \in M_s \tag{6}$$
$$X_{ijs} \in \{0, 1\} \qquad \forall i, j \in P, s \in S \tag{7}$$
$$C_{i,s} \geq 0 \qquad \forall i \in P, s \in S \tag{8}$$

The objective function of the model is to minimize the total completion time of all jobs in the

system, as modelled in (1). Constraint (2) means that a job will only be assigned to one capacity space at each department. Constraint (3) restricts the starting time of an operation at a department s to be greater or equal than the finishing time of the previous department. Constraints (4) and (5) make sure that no two jobs are being assigned to the same capacity space at the same time. To see how these constraints achieve this goal, consider the following situation. If two jobs i and j are both assigned to the same machine l in stage s and they would erroneously be scheduled at the same time, then X_{ijs} would be equal to zero, since i does not precede j in stage s. However, in this situation it holds that $(2 - Y_{isl} - Y_{jsl} + X_{ijs}) = 0$, and thus to satisfy Constraint (4), it should hold that $C_{is} \geq C_{js} + d_{is}$. This implies that the completion of job i in stage s must be larger than the completion time of job j plus the processing time of job i at stage s. Hence, this implies that job j must be processed before job i in stage s, making the illustrated situation infeasible. In this situation, where jobs j precedes i, the value of $Q(3 - Y_{isl} - Y_{jsl} - X_{ijs})$ =1, hence in this setting Equation 5 also holds because Q was chosen to be a large enough number. Likewise, in the setting where job i does precede j, it holds that X_{ijs} =1. Hence we have that $(2 - Y_{isl} - Y_{jsl} + X_{ijs}) \neq 0$ and thus constraint (4) holds. In this setting, the equation $(3 - Y_{isl} - Y_{jsl} - X_{ijs})$ equals 0, therefore requiring that $C_{js} - C_{is} \geq d_{js}$ holds. Through this it is achieved that the processing of job j does not start before the completion time of job i. In the setting where the jobs are not both assigned to the same machine in a department s, both $(2 - Y_{isl} - Y_{jsl} + X_{ijs})$ and $(3 - Y_{isl} - Y_{jsl} - X_{ijs})$ are not equal to zero, hence Equation (4) and (5) both hold due to the value of Q is chosen larger than any difference between completion times. So together Constraint (4) and (5) ensure that no two jobs are being assigned to the same machine at the same time.

Finally, constraint (6), (7) and (8) define the domain of the decision variables Y_{isl}, X_{ijs} and $C_{i,s}$.

Section 5.1.6 referred to the two-stage no-wait HFSP that was multiple times discussed in existing literature. The no-wait HFSP differs from the general HFSP formulation provided in this section in terms of the no-wait property. In the no-wait HFSP, there can be no waiting time for jobs in between department 1 and department 2. In the formulation provided above, there can be waiting time and hence a queue between two consecutive departments. To ensure the no wait property, Equation (3) should contain an equality sign.

5.2.3 Solving a HFSP to Optimality

The HFSP arises in many real life situations, like manufacturing problems. In many cases, even in its simplest form, the HFSP is NP-hard (Ruiz & Vázquez-Rodríguez, 2010). Because of its occurrence in many real life situations and this complexity, the HFSP has received a lot of attention in literature. Many solution approaches have been proposed. For an overview of the proposed exact, heuristics and meta-heuristic methods which have been discussed, the reader is referred to Ruiz and Vázquez-Rodríguez (2010). Reinforcement learning techniques are not among these reviewed methods.

5.2.4 Naming Scheduling Problems

Graham, Lawler, Lenstra, and Kan (1979) have proposed a classification scheme for scheduling problems which is adopted by many researchers. This classification scheme classifies scheduling

problems in terms of three fields: $\alpha|\beta|\gamma$. The parameter α describes the structure of the flow shop: α is composed of $\alpha_1, \ldots, \alpha_4$. The parameter α_1 indicates the general configuration of the shop. In the case of a HFSP we have $\alpha_1 = FH$, where FH is the notation for a hybrid flowshop. The value α_2 denotes the number of departments in the flowshop, and $\alpha_3\alpha_4$ together describe the properties of the machines per department as follows: the notation $(\alpha_3\alpha_4)^k$ indicates that in department k there are α_4 machines of type α_3. In this notation, α_3 takes a value in the set $\{\emptyset, P, Q, R\}$. Here P indicates that there are identical parallel machines, meaning that all machines process jobs with the same speed. A value equal to Q means that we use uniform parallel machines that have machine-dependent speeds. Lastly, the value R indicates unrelated parallel machines are used, which means that the machines have job and machine dependent speeds. The parameter β describes the shop specific set of constraints and assumptions, and lastly γ clarifies what objective is considered. For an overview of possible constraints, assumptions, and objectives one is referred to Graham et al. (1979).

5.3 Reinforcement Learning for Scheduling Problems

In this section related work about Reinforcement Learning (RL) for scheduling problems will be presented.

Gabel and Riedmiller (2007) show that a multi-agent RL approach can very well compete with traditional solution approaches when solving the task of job-shop scheduling. Waschneck et al. (2018) state that even though Deep Q-Learning (DQL) has produced many success stories in the past years, there are hardly any serious applications in the manufacturing industry. In their work, deep Reinforcement Learning is applied to production scheduling to complex job shops such as semiconductor manufacturing. Cooperative DQL agents, which utilize deep neural networks, are trained with a user-defined objective to optimize scheduling. Qu, Wang, Govil, and Leckie (2016) present a multi-agent approximate Q-learning algorithm for dealing with a synchronized-station-based flow shop with a multi-skill workforce and multiple types of machines.

In recent work Han, Guo, and Su (2019) claim to be the first to study a RL based method for solving a HFSP. The conclusion of this paper can be summarized as: RL would be a decent alternative for a solution heuristic for the specific HFSP at hand. However, the proposed solution approach does not allow to generalize beyond this specific case study. Furthermore, the solution procedure is not reproducible based on their report. At the same moment in time van Ham (2019) proposed a single agent RL algorithm which can efficiently solve the picking process in a warehouse. This picking process is modeled as a HFSP. The proposed RL model is applicable for scheduling problems similar to the one in this research, with given sets of users and transports.

Jiménez (2012) presents a general framework for applying MARL for scheduling problems. Small part of his work was dedicated to the HFSP. With regards to the HFSP, his main conclusion was that both the Genetic Algorithm and RL approaches are able to efficiently solve a batch scheduling problem. Unfortunately, the approach towards solving the HFSP was not very extensively discussed in his work. Hence it is not reproducible.

Waschneck et al. (2018) state that: “cooperative multi-agent learning has been applied successfully

to several areas such as network management and routing, electricity distribution management and meeting scheduling in order to exploit the adaptive dynamics of the approach."

Based on this literature review, we identified a gap in literature which we will attempt to fill. Literature on RL, and in particular MARL, applied to the HFSP is scarce. Additionally, it is important to include multiple resources in a hospital scheduling planning. Therefore we wish to include the IC and MC departments in the OT hospital scheduling problem. We propose to model the OT hospital flow as a three-stage HFSP. Then, we wish to research whether MARL techniques can be used to solve the HFSP model, and in particular, a hospital OT flow modeled as a HFSP.

6 Reinforcement Learning

In this chapter the concepts of Reinforcement Learning (RL) that are relevant for this thesis will be introduced, using input from many valuable sources. In particular, Sutton and Barto (2018) and the RL Course by Silver (2015) have provided many useful insights.

Reinforcement learning is based on the natural learning process of animals. Think about how a child or an animal learns something by interacting with its environment. An ape sees fire for the first time and approaches it. As he does so and gets closer, he feels the nice warmth of the fire: positive feedback. As he gets closer to the fire he tries to touch it: "ouch! That hurts!'. This is a negative feedback signal, which will prevent the ape to touch fire in the future. This is what we call natural learning. Interacting with the environment and receiving a reinforcement signal.

When raising a cat named Teddy as a pet, we apply the concept of natural learning to our advantage. If Teddy shows behaviour that is not desired, we use a water sprayer to let it know that it does something wrong. Teddy receives negative feedback from the environment, its boss, when performing a certain action. Repeating this for multiple times, Teddy will in the future no longer exhibit this behaviour. Teddy has learned by interacting with its environment that e.g. she is not allowed to scratch the couch. Reinforcement Learning is the computational version of this natural learning process. In recent years Reinforcement Learning has become famous because of several successful applications in complex gaming environments. The combination of RL with deep learning has led to impressive results. Agents can now learn to play games like Go (Silver et al., 2016), Chess (Lai, 2015), and also the full spectrum of Atari games (Mnih et al., 2013). More recently, RL agents have even learned to play difficult games like Dota (Berner et al., 2019) and Starcraft (Vinyals et al., 2019). In all these games it has been demonstrated that a RL agent can achieve superhuman performance on a difficult task.

What makes RL different from other machine learning paradigms is that there is no supervisor, but only a reward signal. RL is typically considered in between supervised and unsupervised learning methods. Moreover, the feedback is delayed and not instantaneous. RL is applied to Sequential decision making problems. Here, an agents' actions affect the subsequent data that it receives from the environment, hence the data the agent receives is not i.i.d.

The structure of this Chapter is as follows: Section 6.1 will introduce the basic concepts of RL. Section 6.2 presents additional theory required for applying RL in more complex, and uncertain settings. Then, Section 6.3 will elaborate upon the concept of multi-agent reinforcement learning (MARL) together with the challenges and pitfalls. Finally, 6.4 will introduce Deep Reinforcement Learning (DRL). DRL is needed to be able to deal with very large environments, and additionally it allows RL agents to generalize from past experiences.

6.1 Concept of Reinforcement Learning

An agent learns through interaction with its environment. The agent is not told how to behave but instead it will discover what behaviour is desirable. He will do this by interacting with the

environment and discovering what actions maximize his reward. The goal of RL is to learn an agent how to behave, i.e. how to map situations to actions, so as to maximize a numerical reward signal.

While there are different definitions about an agent in the literature, it comes down to the agent being the learner and decision maker. The environment is the thing that the agent interacts with. An agent learns from interaction with the environment in a trial and error fashion. The agent performs some action which is selected according to a policy or at random, and in return receives feedback from the environment in terms of a reward. The goal of a RL task is to find the behavior that maximizes cumulative rewards in the long run. The agent and the environment interact at discrete time steps: $t \in \mathbb{N}^+$.

To illustrate the concepts that were introduced, Figure 1 provides an example of the environment of the cart-pole problem. This example was first introduced by Barto, Sutton, and Anderson (1983). In this environment the agent observe sthe position and the angle of the cart and the pole. The pole starts upright and it's unstable but it can be controlled by moving the cart. The goal for the agent is to balance a pole on a cart and prevent it from falling over by applying a positive force or a negative force to the cart. The action set can be described by $\mathbb{A} = \{a_1 : \text{move left}, a_2 : \text{move right}\}$. A reward of $+1$ is provided for every time step that the pole does not fall over. The episode ends when the pole is more than 15 degrees from vertical or the cart moves to far away from the center. The agent must learn how to balance the pole by trial and error.

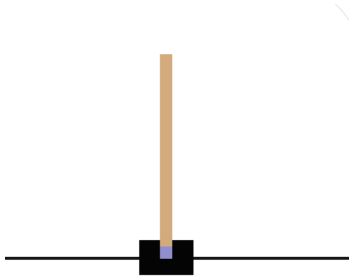


Figure 1: This Figure presents an example of the Cart-Pole environment.

The agent has to make a decision by looking at the current state of the environment. He has to decide whether to move the cart to the left or to the right. He makes his decision based on the current state of the environment. Therefore, it is desirable that the state summarizes past experiences in such a way that all relevant information remains.

In this upcoming section, all necessary RL concepts will be introduced.

6.1.1 Markov Decision Processes

Formally, a RL agent environment interaction setting can be modelled as a Markov decision process (MDP). Markov decision processes model decision making problems in stochastic, sequential envi-

ronments. By choosing a particular action the decision maker does not only influence the immediate state of the environment. This action also affects the probabilities of future transitions and through this the future rewards. A MDP can be represented using the 5-tuple $(S, A, R, \mathbb{P}, \gamma)$, where:

- S represents the state space. States must be defined in such a way that they represent a moment when an action must be taken by the agent
- $A(s)$ represents the action space of an agent in state $s \in S$
- $R(s_t, a_t, s_{t+1})$ represents the reward function, where the agent receives a reward at each step where he transitions from s_t to $s_{t+1} \in S$ by taking action $a_t \in A(s_t)$
- P represents the state-transition probability matrix; $\mathbb{P} : S \times A \times S \to [0, 1]$. This gives the probability of transitioning to state $s_{t+1} \in S$ when the agent chooses action $a \in A(s)$ in state $s \in S$
- γ represents the discount factor which is applied to discount long term rewards, $\gamma \in [0, 1]$. The discount factor will be further elaborated upon in Section 6.2.3

The policy π is what determines the behaviour of the agent. The goal of an agent is to maximize the expected cumulative reward. This can be achieved by learning how to behave and improve his behaviour based on experience. Later in this chapter we will further elaborate upon the concept of the policy.

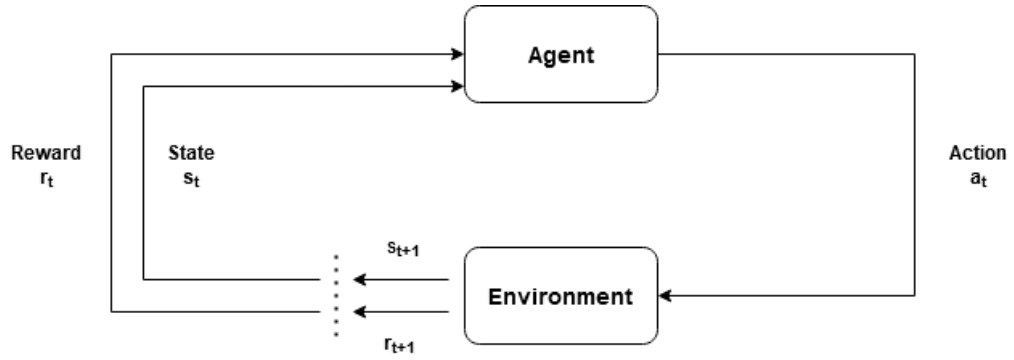


Figure 2: *This image illustrates the concept of agent-environment interaction in a standard RL problem, modeled as a MDP (Sutton & Barto, 2018).*

The main idea of a reinforcement learning setting is depicted in Figure 2. This is the visualization of the following: The agent observes the state of the environment at time step t: $s_t \in S$. Based on this state he takes an action: $a_t \in A(s)$. One time step later the agent receives feedback in terms of a numerical reward: r_{t+1} and he observes a new state of the environment $s_{t+1} \in S$. Now the agent can make a new decision of what action to take in state $s_{t+1} \in S$. This results in a sequence of interactions between the agent and the environment: $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, ...$ This sequence of interactions is regarded as the experience of an agent. The agent observes the current state of the environment and based only on this observation he has to decide what action to take. Therefore it is

desirable and necessary that the environment and the state representation summarizes all relevant information about the past. Meaning, the state representation must have the Markov Property.

A state is said to possess the Markov Property when the probability of each possible value for s_t and r_t depends only on the immediately preceding state s_{t-1} and action a_{t-1} and not on earlier states or actions (Reijnierse, 2013). This implies that a state must contain all information about the past, so about past actions and past states, which are relevant for the future. So, given the present, the history of a MDP does not influence its future. In mathematical formulation:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_0, a_0, ...s_t, a_t)$$
$$P(r_{t+1}|s_t, a_t) = P(r_{t+1}|s_0, a_0, ...s_t, a_t)$$

This implies that when you know $s_t \in S$, the history up until s_{t-1} can be deleted.

6.1.2 Policy

The policy determines how an agent behaves, i.e. what action an agent chooses in state s. The policy π is a probability mapping from states to actions: $\pi : S \to A$. If an agent is following a policy $\pi_t(a|s)$ at time t, $\pi_t(a|s)$ gives the probability of this agent selecting action $a \in A$ in state $s \in S$. A policy can be deterministic or stochastic. In case of a stochastic policy, the policy function looks as follows:

$$\pi(a|s) = \mathbb{P}(a|s) \quad \forall s \in S, \forall a \in A(s)$$

Where $\pi(a|s) \geq 0$ and $\sum_{a \in A(s)} \pi(a|s) = 1$.

In case a policy is deterministic, the policy returns a single action.

$$\pi(a|s) = \begin{cases} 1 & \text{for some action } a' \in A(s) \\ 0 & \text{for all other } a \in A(s) \setminus \{a'\} \end{cases} \qquad \forall s \in S$$

When learning more about the environment, the agent will update and adjust the policy. The goal of the RL agent is to figure out what is the behaviour which will optimize the expected return. Hence, the goal is to find the optimal policy. Section 6.1.8 and 6.2 will further elaborate upon the concept of learning.

6.1.3 Terminal state

An agent will continue until he reaches the goal of an episode or the predefined terminal state, means the end of an episode. This terminal state is the absorbing state in the MDP. Meaning that once this state is reached, the system will never leave this state. Mathematically speaking, in the terminal state the following two conditions are true:

$$P(S_{T+1}|S_T) = 0 \quad \text{for} \quad S_{T+1} \neq S_T \quad \text{and} \quad P(S_T|S_T) = 1$$

Theorem 6.1 (Absorbing Markov chains) *A MDP process that has the Markov Property is called a Markov Chain. In addition, a Markov Chain is an Absorbing Markov Chain if the following two properties are satisfied:*

- *There is at least one absorbing state*
- *It is possible to reach the absorbing state from each non-absorbing state in a finite number of steps*

If one is interested in more theory and literature on markov chains and absorbing markov chains in particular, the reader is referred to Grinstead and Snell (2012).

It is important that a terminal state or stopping criterion is defined for a RL task. Theorem 6.1 then implies that, if a problem can be defined as an Absorbing Markov Chain, then the probability that the absorbing state is reached is equal to one. The probability that our agent thus reaches the terminal and goal state is then also equal to one.

6.1.4 Episodic and Continuing Tasks

There are two kinds of RL tasks: episodic and continuing tasks.

Episodic tasks:
For episodic tasks the agent-environment interaction naturally breaks down into a sequence of separate episodes. Meaning, there is a final time step T or a terminal state that naturally means the end of an episode. For example, think about the example of playing a game of tic tac toe. The game either ends in a win, a lose or a draw for the agent. The episode ends when one of these outcomes occur. Thus for episodic RL tasks, an episode ends when a predefined goal is met. Thereafter, the next episode starts in the simulation.

Continuous tasks
For continuous tasks there is no terminal state. The interaction does not naturally break down into separate episodes. The task could continue forever. As an example: think about an automatic stock trader. There is no terminal state here and the agent will keep on running unless we decide to stop him. This stopping criterion could be a predefined number of steps, a predefined time period or a desired amount of reward. This stopping criterion can be regarded as the terminal state of this continuous task. By doing so, the continuous task can be treated similar as an episodic task.

6.1.5 Rewards

One of the most prominent concepts in RL is the *reward*. After every action, the agent receives a reward from the environment, which represents to which extent the outcome of the action is desirable. Rewards are represented numerically and are not necessarily positive and can be negative as well. The *reward function* determines the reward based on the current state, the action taken

and the new state: $R(s_t, a_t, s_{t+1}) : S \times A \times S \to \mathbb{R}$. The reward obtained in time period t is denoted by r_t.

In general RL settings, it's not just the immediate reward r_t we are interested in. We are also concerned in the rewards we will obtain in future time periods. For now assume that we have a sequence of realisations of the reward, which we denote by $r_{t+1}, r_{t+2}, \ldots$. More specifically, we focus on the *cumulative future reward*.

In a setting with a finite horizon, the cumulative future reward G_t can then be defined as:

$$g_t = r_{t+1} + r_{t+2} + ... = \sum_{k=0}^{T} r_{t+k+1} \tag{9}$$

Where time step T denotes the final time step. Equation (9) can be used in settings where a final step is reached naturally, like for example in a game of tic tac toe or a game of chess. Here, g_t is just one sample of the rewards obtained in one episode of the environment.

Sometimes, there are no intermediate rewards. Meaning the reward at each non-final time step t is zero: $r_t = 0$, where $t \neq T$. An agent only receives a reward at time step T. An example of this reward scheme was used by Silver et al. (2016) for solving the game of Go, where it is very hard to assess intermediate states. The reward scheme which was used: the agent receives no intermediate rewards, and he gets a positive reward of +1 if he wins the game and a negative reward of -1 when he loses the game.

In settings where T can become infinitely large, the definition of the cumulative future reward must be slightly adjusted. A discount factor $\gamma \in [0, 1]$ is introduced to discount future rewards. In settings were T is infinite, $\gamma < 1$ avoids that g_t grows to infinite proportions. The cumulative future reward at time t is then be denoted as:

$$g_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \qquad \gamma \in [0, 1] \tag{10}$$

$\gamma^{k-1} r_{t+k}$ represents the present value of a reward received $k-1$ periods from now.

The closer γ is to zero, the more the agent cares about short term rewards. For $\gamma = 0$, the agent only cares about the immediate reward. The closer γ is to one, the more the agent cares about the long term rewards. So the discount factor tells us how much we care and look into the future. In an episodic environment without uncertainty, γ can be equal to one. Note that Equation (9) is then in fact Equation (10) with $\gamma = 1$.

Of course, in a real life situation, rewards are not known beforehand. Therefore, instead of using the realisation r_t, we will focus on the random variables R_t. Similarly, our study will not focus on g_t, but instead on the random variable G_t.

When designing a RL system the reward function is extremely important. The reward function is your way of communicating to the agent what goal should be pursued. It is important that the

reward signal should not be used to communicate any sort of prior knowledge to the agent. The reward function should only communicate what the agent should achieve but not of how this should be achieved.

6.1.6 The Value Function

Most RL problems focus on estimating value functions and/or values. The value of a state indicates how good it is for an agent to be in a certain state. The value of a state reflects the long term value of being in that state. Some states have a higher value than others. If for example our goal is to find the exit of a maze, then a state in which our agent is closer to the exit will have a higher value than a state that represents our agent somewhere at the beginning of the maze. The notion of how good it is, is determined in terms of expected reward. Sutton and Barto (2018) state that value functions are important for efficient search in the space of policies and that the use of value functions is what distinguishes RL methods from evolutionary methods that search directly in the policy space guided by evaluations of entire policies.

Sutton and Barto (2018) define the value function $V_\pi(s)$ as the expected future reward when starting in a state s and following a fixed policy π in future states. Mathematically, the value function $V_\pi : S \to \mathbb{R}$ can now be defined as follows:

$$V_\pi(s) = \mathbf{E}_\pi[G_t|S_t = s] \qquad \forall s \in S \tag{11}$$

6.1.7 The Bellman Equation

Combining the definition of G_t and $V_\pi(s)$ from Equation (10) and (11) one discovers the following recursive relationship between the value of a state s and its successor state s_{t+1} (Sutton & Barto, 2018):

$$\begin{aligned} V_\pi(s) = \mathbf{E}_\pi[G_t|S_t = s] = \mathbf{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] = \mathbf{E}_\pi[R_{t+1} + \gamma V_\pi(s')|S_t = s] \\ = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S} \mathbb{P}(s'|s,a)[R(s,a,s') + \gamma V_\pi(s')], \quad \forall s \in S \end{aligned} \tag{12}$$

Where $s' = s_{t+1}$; $\pi(a|s)$ is the probability of choosing action a in state s when the agent is following policy π. $\mathbb{P}(s'|s,a)$ is the probability of transitioning to state $s' \in S$ when performing action $a \in A(s)$ in state $s \in S$. The value of a state can thus be decomposed into two parts, the immediate reward, and the discounted value of the successor state.

Equation (12) is referred to as the Bellman equation for V_π in literature and is one of the most fundamental relationships in RL literature.

Recall that the goal of solving a RL method is to find the policy which results in the highest expected reward in the long run. An optimal policy is the policy that in each state selects the

action that leads to s' with the highest $V(s')$: the highest expected reward. The optimal value function can thus simply be derived by filling in this optimal policy in the Bellman function for V_π (Melo, 2001).

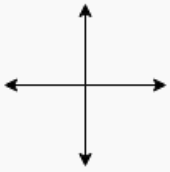
The Optimal Value Equation:

$$V^*(s) = \max_\pi V_\pi(s) = \max_{a \in A(s)} \sum_{s' \in S} \mathbb{P}(s'|s,a)[R(s,a,s') + \gamma V^*(s')] \tag{13}$$

Example 1 presents an example for the optimal value function from Equation (13) works. This Example is based upon an example which was provided by Silver (2015).

Example 1 ***Small maze*** *An agent is located in a maze and has to find the shortest path to the exit. The state space is described by all the cells that the agent can be located in:* $S = \{1, 2, 3, 4, 5, 6, 7, 8, Exit\}$. *The upper left corner is the exit of the maze. The goal of the agent is to find the exit as fast as possible. An agent can move up, right, down and left:* $\mathbb{A} = \{\uparrow, \rightarrow, \downarrow, \leftarrow\}$. *E.g. if the agent moves up in state 8 he moves to state 5. An action that leads the agent out of the grid leaves the state unchanged: e.g. if* $s = 6$, $a :\leftarrow$, *then* $s' = 6$. *The states and the actions are depicted in the figures below.*

Exit	1	2
3	4	5
6	7	8



Actions

The agent receives a reward of -1 for every action he performs until the terminal state is reached. The goal of the agent is to find the exit as fast as possible. i.e. the goal of the agent is to maximize his cumulative reward.

$$r(s,a) = \begin{cases} 0 & if\ s = exit \\ -1 & otherwise \end{cases}, \quad \forall s \in S, a \in A \tag{14}$$

The rewards for each state are depicted in the Figure below.

0	-1	-1
-1	-1	-1
-1	-1	-1

Rewards

0	-1	-2
-1	-2	-3
-2	-3	-4

Value per state

In this example, the optimal value is equal to the minimal number of steps that it takes the agent to reach the exit. The optimal value of a state follows the optimal value Equation (13). E.g. for $s = 2$, *the agent can get to the exit in two steps:* $V^*(s = 2) = 2$. *Step 1:* $s = 2, a =\leftarrow$ *brings him to* $s' = 1$. *Step 2:* $s = 1, a =\leftarrow$, $s' = exit$. *The optimal value of each state is depicted in the right figure above.**

**Example 2 will later exactly illustrate how these optimal values come about.*

The value function tells an agent how good it is to be in a certain state. The next step is to determine what is the best way for an agent to act when in this state. Or in other words: to

determine the optimal policy. Trying to determine the optimal policy using only values and a policy, often gives one a hard time. That is why Q-values and Q-learning will be introduced in the next section.

6.1.8 Q-values

Q-learning was first proposed by Watkins (1989) and is one of the most popular RL techniques. The Q-Value is a function of a state-action pair, and assigns a value to each state-action pair indicating how good it is for an agent to perform a certain action in a given state. The 'Q' stands for quality. The notion of how good a state-action combination is, is again expressed in terms of expected return.

The value V and the Q-value are closely related. The value of state s is the expected return, summarized over all possible actions $a \in A(s)$ the agent can perform. The Q-value of an action $a \in A(s)$ is the expected return, given that that particular action if performed. Therefor, the value can be seen as the expected Q-value, where the expectation is taken over the possible actions $a \in A(s)$.

The Q-value function can be mathematically denoted as:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \quad \forall a \in A(s), \forall s \in S \tag{15}$$

The relationship between the Value- and the Q-value function is as follows :

$$\begin{aligned} V_\pi(s) \overset{(11)}{=} \mathbf{E}_\pi[G_t|s_t = s] &= \sum_{a \in A(s)} \mathbb{P}_\pi(A_t = a|S_t = s)\mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &\overset{(6.1.2,15)}{=} \sum_{a \in A(s)} \pi(a|s)Q_\pi(s,a), \quad \forall s \in S \end{aligned} \tag{16}$$

The Q-value denotes the expected return for a (s,a)-pair and the Value function denotes the expected return for a state s. Hence the value function can be written as the expectation over the Q-values from that state. Now, the answer to the question "how to determine the optimal policy" can be derived. Under the optimal policy, the agent chooses the action with the highest $Q(s,a)$ value in each state s. The optimal value $V^*(s)$ for a state s, is the highest expected return which can be achieved from this state.

$$V^*(s) = \max_{a \in A(s)} Q^*(s,a) \quad \forall s \in S \tag{17}$$

The definition of the Q-value function in Equation (15) can be rewritten as:

$$Q_\pi(s,a) = \sum_{s' \in S} \mathbb{P}(s'|s,a)[R(s,a,s') + \gamma V_\pi(s')] \quad \forall a \in A(s), \forall s \in S \tag{18}$$

Here one can clearly see the dependency between Q and V. This Equation (18) is referred to as the Bellman Equation for Q_π in literature (Sutton & Barto, 2018).

Under an optimal policy, the Bellman Equation for Q_π gives us the optimal action-value function $Q^*(s, a)$:

$$\begin{aligned} Q^*(s,a) &= \sum_{s' \in S} \mathbb{P}(s'|s,a)[R(s,a,s') + \gamma V^*(s')] \\ &\overset{(17)}{=} \sum_{s' \in S} \mathbb{P}(s'|s,a)[R(s,a,s') + \gamma \max_{a' \in A(s')} Q^*(s',a') \end{aligned} \tag{19}$$

6.1.9 Q-Learning using Q-Tables

Equation 19 provides us with recursive relationship that in theory enables us to calculate the value of Q^* directly. However, such recursive calculations are prohibitively costly in practically every real world application, due to the exponential growth in required calculations as you move deeper in the recursion. Therefore, researchers use a method what is called *Q-learning* where Q-values are calculated and updated iteratively. For this, a so-called *Q-table* is used to store estimates of Q^*. These estimates of Q^* are often referred to as the *Q-values*.

A Q-Table is used to store the estimated Q-values for each (s, a) pair. The dimensions of this table are therefore $|S| \times |A|$, where the states $s \in S$ are the rows, and the actions $a \in A(s)$ are the columns. For some cases, the action space is state dependent, as represented by $A(s)$. In these cases we only look at the subset of actions that are possible in a state $s \in S$. Here, the dimension of the Q-Table is not equal to S x A, but instead the dimension of the column depends on the size of the action set in this state: $A(s)$. Recall by Equation (17), that the optimal value is equal to the maximum Q-value. Sometimes this value function is added as an extra column to the Q-Table for convenience.

The Q-values are updated in an iterative fashion. The Q-Table is first initialized, usually with all zeroes. Updating an estimated Q-value is done as follows. For a state $s \in S$ we choose an action $a \in A(s)$. The new state s' and the reward $R(s, a, s')$ are then observed. After which the Q-value for this $((s, a), s', R(s, a, s'))$ combination is calculated using Equation (18). For the value of Q^* in the right-hand side of the equation, we use the current value in the Q-table, avoiding the need for recursive calculations.

In finite problems with a relatively small state space, the updates are done for all Q-values simultaneously, by calculating the values for every (s,a) pair and updating the values in the Q-table. This process is continued until either an convergence criterion is met, or until a predefined number of iterations have been performed. The pseudocode for the Q-learning algorithm can be found in Appendix B. Example 2 continues based on Example 1 and illustrates how the Q-learning algorithm works. It will be shown that the Maze example can be solved using Q-learning.

Melo (2001) shows that for $0 \leq \gamma < 1$ both the Q-value function (Equation (19)) and therefore also the value function (Equation (13)) converge to their optimal value functions: $V^*(s)$ and $Q^*(s, a)$. Readers that are interested in the full proof of the Convergence of Q learning are referred to this

article.

In practical applications, the state space if often extremely large. In these situations, the Q-learning method is too inefficient to be usable in practice. A reason for this is that in Q-learning, a lot of time may be spend in updating Q-values for actions that are inferior in terms of result. Moreover, for larger instances, it might be impossible to store the entire Q-table in the first place. Methods that incorporate solutions to these problems will be reviewed in Section 6.2 and Section 6.4.

Example 2 ***Continued: Small maze Q-Learning*** *In the example of the maze we are dealing with a small and episodic environment, hence the discount factor can be set to* $\gamma = 1$. *Moreover, there is no transition uncertainty. This means that if an agent chooses* $a = \uparrow$ *in a state* s, *then the agent will move upwards with probability 1. The policy* π *that is employed is: always select the best action, i.e. the action corresponding to the highest Q(s,a) value. Therefore the update rule for Q(s,a) can be rewritten as:*

$$Q_\pi^{i+1}(s,a) = R(s,a,s') + V_\pi^i(s') \tag{20}$$

$$\text{where:} \quad V_\pi^i = \max_{a' \in A} Q_\pi^i(s',a') \tag{21}$$

When the exit is reached, $V_\pi(exit) = 0$. *Now, we can illustrate how the Q-learning algorithm works. The Q(s,a) Table is initialized with only zeroes as in the Table below. All yellow (s,a) combinations, are pairs that transfer to itself, so* $(s,a) \rightarrow s' = s$. *The green cells, are (s,a) pairs that end up in the exit:* $(s,a) \rightarrow Exit$.

Q-Values					
States	$\uparrow$	$\rightarrow$	$\downarrow$	$\leftarrow$	*Value*
1	*0*	*0*	*0*	*0*	*0*
2	*0*	*0*	*0*	*0*	*0*
3	*0*	*0*	*0*	*0*	*0*
4	*0*	*0*	*0*	*0*	*0*
5	*0*	*0*	*0*	*0*	*0*
6	*0*	*0*	*0*	*0*	*0*
7	*0*	*0*	*0*	*0*	*0*
8	*0*	*0*	*0*	*0*	*0*

For each (s,a)- pair the Q-values will be calculated. Example: For $s = 4, a = \uparrow$, $s' = 1$. *We can then calculate* $Q(4,\uparrow)$*:*

$$Q_\pi(4,\uparrow) = R(4,\uparrow,1) + V_\pi(1) = -1 + 0 = -1$$

When all $Q_\pi(s,a)$ *values are calculated for each (s,a) pair, these values are updated simultaneously. Then,* $V_\pi(s)$ *can also be updated:*

$$V_\pi^1(s) = \max_{a \in A} Q_\pi^1(4,a) = -1$$

After the first iteration, the Q-table looks as follows:

Q-Values					
States	$\uparrow$	$\rightarrow$	$\downarrow$	$\leftarrow$	*Values*
1	*-1*	*-1*	*-1*	*-1*	*-1*
7	*-1*	*-1*	*-1*	*-1*	*-1*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
8	*-1*	*-1*	*-1*	*-1*	*-1*

Note that for $(1,\leftarrow)$ *and* $(3,\uparrow)$, *new state* $s' = Exit$. *The value for* $V(s') = V(exit) = 0$.

$$Q_\pi(1,\leftarrow) = R(1,\leftarrow,exit) + V_\pi(Exit) = -1 + 0 = -1$$

All the yellow cells are transits to itself: in this case $s = s'$, *and* $r(s,a,s') = -1$.

After iteration $i = 5$, the Q-values do not change anymore: the optimal Q-values are found. The Q-Table then looks as follows:

	Q-Values				
States	$\uparrow$	$\rightarrow$	$\downarrow$	$\leftarrow$	Value
1	-2	-3	-3	-1	-1
2	-3	-3	-4	-2	-2
3	-1	-3	-3	-2	-1
4	-2	-4	-4	-2	-2
5	-3	-4	-5	-3	-3
6	-2	-4	-3	-3	-2
7	-3	-5	-4	-3	-3
8	-4	-5	-5	-4	-4

Note that the Values correspond to the 'Value per state' Image in Example 1.
From this Q-Table, the optimal policy for the agent can be determined, i.e. the optimal way to behave. In each cell, the agent chooses the action with the highest corresponding $Q(s, a)$ value. This optimal policy how to behave are the coloured cells in the Q-Table. As one can see, starting in $s = 8$ there are multiple policies leading to the same overall minimal return. Meaning, there is no unique optimal policy in this example when starting in $s = 8$.

6.2 Learning in More Complex, Real-Life Settings with Uncertainty

In the example and the theory that we have considered so far, the MDP for the environment was completely known and the environment was deterministic. In almost each real life scenario, the dynamics of the environment are not entirely known. This results in uncertainty in state and action transitions. This also gives rise to problems where the state and action space are continuous and hence infinitely large. The goal of this section is to introduce learning methods which can be applied in situations where there is uncertainty and where the exact dynamics of the environment are not known.

Learning methods can be divided into two classes (Jiménez, 2012):

- Model based approaches: where an explicit model of the environment is known and used to derive an optimal policy.
- Model free approaches: these are able to derive the optimal policy for the agent to behave in an environment without knowing the exact dynamics of the environment

In a real life setting, constructing a perfect model of the environment is practically not possible. Hence, in these settings a model free solution approach will be utilized. These approaches do not need an environment model at all. Sutton and Barto (2018) state "Because models have to be reasonably accurate to be useful, model-free methods can have advantages over more complex methods

when the real bottleneck in solving a model is the difficulty of constructing a sufficiently accurate environment model.". Model free methods thus concern the estimation of the value function of an unknown MDP.

In a model free setting, there are two ways of learning (Silver, 2015):

- **Monte Carlo (MC) Learning**: this learning method works for an episodic task. An agent interacts with the environment, and rewards are collected only at the end of a finite episode. The value is estimated using the mean return. The episode is one finite run of the environment. Recall that the end of an episode can either be when the agent reaches a finite state, or when a predefined number of (time)steps has been performed. Instead of remembering old episodes and keeping track of all statistics, an incremental update procedure can be used. Meaning, a running mean is used by introducing a learning rate α. This is another way to compute the expected value, by updating the estimate for a state at each time step, such that old episodes can be forgotten. The value of each state that has been visited during an episode is then updated using the actual future return G_k from state s onwards. After each iteration, we move α in the direction of the difference between the observed reward and the previously expected reward from this state: $(G_t - V(S_t))$.

$$V_{k+1}(s) = V_k(s) + \alpha(G_k - V_k(s)), \quad \forall s \in \mathbb{S}, \alpha \in [0, 1] \tag{22}$$

  Where k is a discrete time instance, $\mathbb{S}$ are all states that have been visited during episode k. $V_k(s)$ is the estimated expected return at the k^{th} time period of state s. G_k is the actual return in episode k following after the occurrence of state s.

- **Temporal Difference (TD) Learning**: The fundamental idea behind TD learning is bootstrapping. This entails that the remainder of the trajectory (of the episode) is replaced by the estimate of what will happen from a certain point onwards. Hence instead of waiting until the end of the episode, values are updated at each step performed by the agent. This simplest form of TD is called one-step TD learning or TD(0) learning. In this setting, $V_{k+1}(s)$ is updated based on the estimated future return from state s' on wards:

$$V_{k+1}(s) = V_k(s) + \alpha(R(s, a, s') + V_k(s') - V_k(s)), \quad \alpha \in [0, 1] \tag{23}$$

  Where $V_k(s)$ is the estimate of the maximal future return in state s in the previous episode. Estimates are thus updated using estimates. a is the action that was chosen in s, and s' and $R(s, a, s')$ are observed. TD learning can learn before knowing the final outcome of the episode. Under certain conditions for α which are discussed below, Equation (23) converges to the true value function. Readers interested in the proof are referred to Sutton and Barto (2018). Moreover, Sutton and Barto (2018) state that the theorems regarding the convergence of the TD(0)-learning algorithm also holds when TD(0)-learning is applied to the Q-function. This implies that TD(0) can thus also be applied in the Q-learning algorithm. Where the Bellman Equation is replaced by Equation (24). In Equation (24), $R(s, a, s') + \gamma \max_{a' \in A(s')} Q_k(s', a')$ is the estimated future return at time k, and is referred to as the TD-target. This finding was one of the early breakthroughs in RL. The learned Q-function directly approximates the optimal Q-function, $q*$, independently of what policy is followed. If the reader is interested in a more theoretical and technical background, he is referred to Sutton and Barto (2018). This

update rule uses per step observations of $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, and is therefore referred to as the SARSA-rule in literature.

$$Q_{k+1}(s,a) = Q_k(s,a) + \alpha[R(s,a,s') + \gamma \max_{a' \in A(s')} Q_k(s',a') - Q_k(s,a)] \tag{24}$$

6.2.1 Size of the Learning Rate α

Both in Monte Carlo and Temporal Difference learning, the step size parameter α determines the rate of learning in the RL learning method. Therefore, this α is also called the learning rate. The learning rate can either be a fixed value or a function of the number of iterations (the number of value updates): $\alpha = \alpha_n(s,a)$.

Sutton and Barto (2018) provide us with the following conditions that are required for the convergence of Equation (23) to the true values with probability 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \tag{25}$$

$$\sum_{n=1}^{\infty} \alpha_n^2(a) < \infty \tag{26}$$

Equation (25) is required such that the steps are large enough to overcome the initial conditions or random fluctuations. Where (26) guarantees that eventually the steps become small enough such that convergence is guaranteed. Parameter settings that satisfy both these criteria are however seldom used in practice (Sutton & Barto, 2018).

Note that when a constant learning rate α is chosen, then Equation (26) does not hold. In the case of a constant α when using TD(0)-learning for the Q-learning algorithm, the estimates for Q will thus never completely converge. Instead, they will continue to vary in response to the most recently received rewards. In this case the Q_{k+1} is a weighted average of past rewards and the initial estimate. Here, more weight is assigned to more recent rewards than to rewards that are further in the past. This is desirable in non-stationary problems. Sequences that have parameters which satisfy both Equation (25) and (26) converge very slowly or need considerable tuning in order to obtain a satisfactory convergence rate Sutton and Barto (2018). Due to this, and because of the non-stationary nature of RL settings in practice, learning rates that satisfy both these criteria are rarely used in practice. Instead, usually a constant learning rate is used in practice. With a value of $\alpha = 0$ then the Q-values are not updated at all. If $\alpha = 1$, then the old value is simply replaced by the new estimated value for Q. With a too large value for α, the Q-values will oscillate too heavily around the true value. Hence, the step size should be chosen small enough to avoid this. Usually, therefore a small value for α is chosen, for example, $\alpha = 0.1$ (Jiménez, 2012). For the remainder of this thesis, a constant value for α will be chosen.

6.2.2 Action Selection Methods and the Exploration-Exploitation Trade-off

One of the challenges in RL is the choice of action selection methods, taking into account the exploration-exploitation trade-off. This was not relevant for the example we encountered so far. Up until now, the examples were so small that all (s, a)-pairs could be evaluated and hence all information about the environment was available, and no action selection method was required. However, in a complex real-life environment, the agent never has full knowledge about the environment, yet his goal is to maximize his expected reward. In order to reach a high reward, the agent has to use knowledge he has obtained in the past to select which action leads to the highest expected reward: **exploitation**. However, to discover such actions he also has to explore the environment by taking sub-optimal actions: **exploration**. This is the exploration-exploitation trade-off. To construct an effective learning method it is important to properly control this trade-off. To do this, there are several action-selection methods that can be used. Here, the two most common approaches will be introduced: the ϵ-greedy approach and the softmax (Boltzmann) approach (Jiménez, 2012).

- **ϵ-greedy approach**: an agent chooses a random action with probability ϵ, alternatively with probability $1 - \epsilon$, he chooses the greedy action which corresponds to the action with the highest Q-value. Note that, $\epsilon \in [0, 1]$.

$$a_t(s) = \begin{cases} \text{Random action} & \text{with probability } \epsilon \\ \arg\max_{a \in A} Q_t(s, a) & \text{with probability } 1 - \epsilon \end{cases}$$

  If a constant ϵ is chosen, the agent will remain exploring the environment with the same probability. Alternatively, a decay function for ϵ can be chosen. Usually, at the beginning of the training phase ϵ is initialized to a large probability close to 1, as the agent gets more information about the environment, the ϵ is decreased until it reaches a small constant (Juliani, 2016). A drawback of this approach is that when the agent explores, he chooses equally amongst all actions. The probability of selecting the second-best option is thus equal as choosing the worst action.

- **The Softmax Exploration approach** varies the action selection probabilities based on the estimated values of the corresponding actions. The greedy action still has the highest probability, and the other actions are ranked based on their estimated value.

$$\pi(a|s) = \frac{e^{Q_t(s,a)/\tau}}{\sum_{b=1}^{m} e^{Q_t(s,b)/\tau}}$$

  where m is the total number of available actions in state s. τ: temperature, is a positive parameter which controls how greedily the agent will behave. As $\tau \to 0$, the more greedy the agent will behave and the more difference there is in the probabilities over the different actions. When $\tau \to \infty$, the actions become equally probable. Just like in the ϵ greedy approach, at the beginning exploration is desired, while at a later moment exploitation is preferred. Hence the parameter τ is decayed over time. Something that should be noted here is that by using Q-values "what the agent is estimating is a measure of how optimal the agent thinks the action is, not how certain it is about that optimality" (Juliani, 2016).

These are just two of many action selection criterion. Also, there have been many experiments to determine what action selection method is better. The answer to this question is problem dependent, so there is no one answer to this question (Jiménez, 2012).

6.2.3 Size of the Discount Rate γ

As was discussed in Section 6.1.5 the discount factor indicates whether an agent cares more about immediate returns or more about future returns. Next to this, the discount factor also serves to avoid practical numerical issues of infinite returns. Moreover, for the convergence of Q-values, a discount factor $0 \leq \gamma < 1$ should be chosen (Section 6.1.8). If a discount factor close to 1 is used, then in Q learning, the values will represent the cumulative expected future reward.

Not much literature is available on what value to choose for the discount factor. Some research has been done towards a dynamic discount factor (François-Lavet, Fonteneau, & Ernst, 2015). Their conclusion is that a dynamic discount factor should be further researched but there are no concrete suggestions or conclusions given for adapting a dynamic discount factor. In many papers and applications, a constant discount factor is used and arbitrarily set to 0.9. Hence in the remainder of this work, a constant discount factor γ of 0.9 will be utilized as well.

6.3 Multi-Agent Reinforcement Learning

In this section the concept of Multi-Agent Reinforcement Learning (MARL) will be introduced. This elaboration is based upon the extensive theory and definitions presented by Nowé, Vrancx, and De Hauwere (2012) and Jiménez (2012).

In a Multi-Agent System (MAS) there are multiple agents who act in the same environment. The interaction between multiple agents and the environment can best be illustrated by Figure 3 (Nowé et al., 2012). Multi-agent systems naturally occur in many real life situations where there are a large number of learning agents. Examples of domains are: robotics, telecommunications, economics, distributed control, auctions, traffic light control, etc. (Nowé et al., 2012). One of the main characteristics that distinguishes a multi-agent system from a single-agent system is the notion of decentralized control. MAS are used because systems are either naturally decentralized or, because of the complexity of the domain.

The main advantage of a multi-agent system is that agents can be located at different locations and each agent can be responsible for a different part of the system. There are two different types of multi-agent systems. There are systems with cooperative agents and systems with non-cooperative agents. In a cooperative system agents work together and pursue a common goal. A multi agent decision process in which all agents share the same reward function is a Cooperative Markov Game. Due to the interaction between the agents, the complexity of a multi-agent problem can rise rapidly with the number of agents (Panait & Luke, 2005). Alternatively, agents have individual interests and goals and hence the agents do not cooperate. This is called a Non-Cooperative Markov Game.

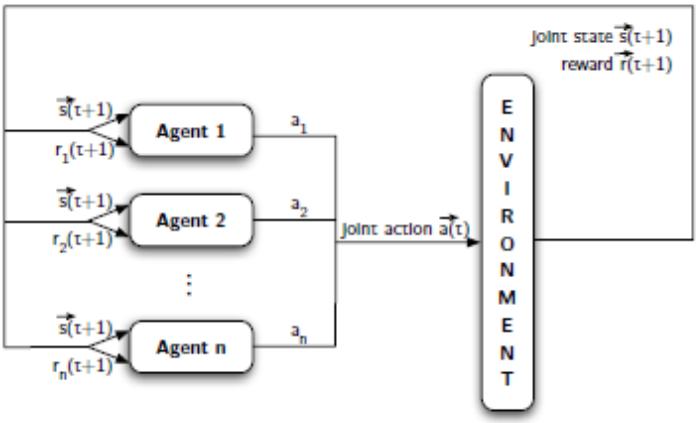


Figure 3: *Figure obtained from Jiménez (2012) which illustrates how Multi-Agent - Environment interaction occurs in literature.*

6.3.1 Joint vs. independent learners

If the agents are joint learners, the assumption that agents should be able to observe the actions of the other agents must be satisfied. Independent learners do not have to observe the actions of other agents.

Existing work on MAS can be divided in two parts, namely in team learning and concurrent learning. In team learning there is a single learner which tries to learn the jointly optimal behaviour and solutions to the multi-agent problem. In concurrent learning on the other hand there are multiple simultaneous learners, where there is often one agent per task (Panait & Luke, 2005). In this thesis, concurrent learning is the topic of interest.

6.3.2 Markov Game

In section 6.1.1 the single Agent framework was modeled as a MDP. This notion can be extended to the multi-agent setting, by defining the MAS as a Markov Game. Markov Games were originally introduced in Game Theory, but more recently Littman (1994) proposed Markov Games as a standard framework for multi-agent RL.

The mathematical definition of a Markov Game (Jiménez, 2012):

- M is the number of agents
- $S = s^1, ..., s^N$ is a finite set of system states
- $A_m(s^i)$ is the action set of agent m in state s^i

- $P : S \times A_1 \times \ldots \times A_M \times S \rightarrow [0, 1]$ is the transition function
- $R_m : S \times A_1 \times \ldots \times A_M \times S \rightarrow \mathbb{R}$: is the reward function of agent m

$\overrightarrow{a}^i = (a_1^i, a_2^i, ..., a_M^i)$ is used to denote the joint action of all agents in state i, with $a_m^i \in A_m(s^i)$. Here, $A_m(s^i)$ are all available actions for agent $m \in M$ in state s^i. Furthermore, s^i is used to denote the starting state, and s^j is used to denote the ending state. Transition probabilities $P(s^i, \overrightarrow{a}^i, s^j)$ and rewards $R(s^i, \overrightarrow{a}^i, s^j)$ now depend on the starting state s^i, the joint action of all agents $\overrightarrow{a}^i$ and the new state. Jiménez (2012): "States and actions are thus the result of multiple agents choosing an action independently". Transitions in this setting are also assumed to have the Markov Property. The Markov Property was introduced in Section 6.1.1.

In a setting where agents have what is called full knowledge about the environment, each individual agent observes the whole system state. The observed state definition is then the same for each agent. In this case $s_1^i = \ldots s_m^i$. In this thesis we will consider a case setting in which all agents have full knowledge about the environment.

The reward function is different for each agent, and each agent m receives an individual reward $R_m = R(s^i, \overrightarrow{a}^i, s^j)$. Different agents can receive a different reward for the same state transition. One important property of a Markov Game is that agents make decisions simultaneously and independent of each other.

In everything that has been mentioned about the multi-agent setting, it is assumed that each agent has full knowledge about the environment. With this assumption however, one of the main advantages of a multi-agent setting is disregarded. Namely, the notion of decentralized control. If the reader is interested in more literature on decentralized decision making in an environment where agents do not observe the full environment, the reader is referred to Jiménez (2012).

6.3.3 Challenges and Pitfalls of MARL

Learning in a multi-agent setting is fundamentally more challenging than the single agent case as the agent needs to interact with the environment and potentially with each other at the same time. Also, the feedback that an agent receives is influenced by these other agents. The key challenges of MARL settings are listed below (Kouridi, 2020).

- **Non-Stationarity problem**: The key challenge in MARL is to learn the best response to the behaviour of other agents (Hernandez-Leal, Kaisers, Baarslag, & de Cote, 2017). However, all agents learn at the same time. If other agents change their behaviour, then the learning target moves. As a result, a policy that is a best response now, might not be a good policy in the future. If the agent ignores this issue and only focuses on optimizing his own policy, the algorithms will likely fail to converge. With as a result for each agent an endless cycle of adapting to other agents. Hernandez-Leal et al. (2017) provide a coherent overview of work that addresses the non-stationarity problem in the MARL setting.
- **Credit assignment**: It is difficult for the agent to assess his own individual contribution to the received reward. For instance, an agent might have chosen his best possible action in a

given state. All other agents choose an exploratory action, which is not their best possible action. The joint action then leads to a reward which is relatively bad. The agent who chose the best action then also reduces his expected reward for choosing his best action, and thereby reducing the probability of choosing this action (Kouridi, 2020). A possible approach to avoid this credit assignment issue, is a method where each agent estimates the impact of his own action on the return with a counterfactual baseline. The baseline is an estimate of what would have happened on average if the agent would have chosen a different action. This approach is called Counterfactual Multi-Agent Policy Gradients (Foerster, Farquhar, Afouras, Nardelli, & Whiteson, 2018).

- **Computational burden of training multiple Q-networks**: Training a Q-network usually requires a lot of computational resources. In a multi agent setting, multiple networks have to be trained. This can be a huge computational burden. Therefore, research has been done on reusing knowledge that can come from other agents, this is what is called Transfer Learning. Da Silva and Costa (2019) encourage researchers to work towards reusing all knowledge resources in a MARL setting. Their paper provides an in-depth discussion of current research on transfer learning and open research questions. The computational burden was for the case considered in this research not an issue. The models can be trained within several hours which was done overnight.

- **Co-evolutionary behaviour**: Multiple agents are learning at the same time. With his exploratory actions, an agent can influence the outcomes and the learning process of the other agents. A possible pitfall of this is relative overgeneralisation. Wiegand (2003): "Relative overgeneralisation: the coevolutionary behavior that occurs when populations in the system are attracted towards areas of the space in which there are many strategies that perform well relative to the interacting partner." Approaches to prevent relative overgeneralisation are: Leniency, distributed Q-learning, and hysteretic Q-learning (Kouridi, 2020).

- **Partial observability**: This occurs when agents have only access to part of the information of the environment. This type of problem can be modelled using a Partially Observable Markov decision Process. The pitfall of partially observable agents, is that they may learn suboptimal behaviours.

- **Global exploration**: Just as in the single agent setting, in a MA case, the agents are facing the crucial exploration-exploitation dilemma. In the multi-agent setting this dilemma is even more challenging. The parts of the environment that an agent explores depend on the behaviour of the other agents. Hence if the exploration is not coordinated then exploration might not lead to a better understanding of the environment but instead lead to policy oscillations. Verbeeck, Nowé, and Tuyls (2005) report about a MA coordinated exploration method, so called Exploring Selfish Reinforcement Learning, to overcome this issue.

- **Coordination of behaviour**: While working on a cooperative multi-agent example with a joint objective, we experienced one main problem. How does one coordinate the behaviour of the multiple agents? In other words, how does one make sure that the individual actions lead to joint optimal decisions? The optimal policy for an agent not only depends on its own actions, but it highly depends on the policies and the actions of other agents. It turns out that this is a challenging topic which currently occupies a lot of researchers. There are several papers that propose solution approaches for dealing with coordination of the behaviour of the

multiple agents, e.g. Littman (2001). These solutions incorporate concepts of Game Theory to describe the problem setting of a MAS, and for modelling the behaviour of the other agents: teammate modelling (Zhou & Shen, 2011). This team-mate modelling is an aspect of the system that requires observing and learning an additional model for the behaviour of other agents (Zemzem & Tagina, 2018). This teammate modeling is then combined with traditional Q-learning. This is an exciting yet challenging and highly complex field of research, which goes beyond the scope of this thesis.

There are many empirical success stories about application of MARL, like the game of Go (Silver et al., 2017), or the playing of real-time strategic games. However, the theoretical background in literature is lacking. To this end, Zhang, Yang, and Başar (2019) have provided a review of MARL algorithms which are backed by theoretical analysis. Readers that are interested in MARL algorithms and the current state of research in this area are referred to Zhang et al. (2019) or to Hernandez-Leal et al. (2017). Their review provides the reader with a good idea about the current state of the MARL research together with its challenges and directions for further research.

6.4 Deep Reinforcement Learning

Silver (2015): "In many real life examples, the state space becomes very large. Take for example the game of backgammon, which has 10^{20} states, or the game of Go which even has 10^{170} states." RL has been able to successfully solve these games. Imagine having to keep track of all these states in a physical Q-table. This would be highly inefficient, and is simply computationally infeasible. Even if it would be possible to store all state-action pairs, then it would still be too slow to learn the value of each state individually. This is what is called the curse of dimensionality. To avoid this curse of dimensionality, in situations with very large states and actions sets, Deep Reinforcement Methods are used.

Deep Reinforcement Learning (DRL) is used to indicate the combination of deep learning and RL, and is used to partially overcome the curse of dimensionality (Nguyen, Nguyen, & Nahavandi, 2020). For the deep Q-learning method, this means that instead of learning the value of each state-action pair individually, the Q-value function will be estimated. This will not only allow us to reduce the memory stored, but it will also enable the agent to generalize from experience. The idea is that a function is fitted to the state-action pairs that the agent has seen. This function can also be applied to state-action pairs he hasn't encountered in the past. Simply by plugging in a new (s, a)-pair in the estimated function. In this way, the agent is able to generalize to new states from experience. Mnih et al. (2015) were the first to announce the success of deep RL. They demonstrated this by creating an agent that was able to achieve a professional human level across a set of 49 Atari games.

The field of Deep RL is a complicated and active field of research, about which full theses could be written. In this work, we simply wish to utilize the DRL technique such that we are able to deal with larger instances, and such that the agent will be able to generalize from previously gained experience. The upcoming section will present the fundamental concepts and the idea behind DRL needed to work with this solution method. If the reader is interested in a more theoretical background, or in the challenges and ideas for future research on Deep RL, he is referred to literature. The website openAI (2018) provides a good overview of key papers on the different topics within Deep RL.

6.4.1 Deep Q-learning

Deep Reinforcement Learning is introduced since it will enable the model to generalize from past experiences. Meaning that acquired knowledge about somehow similar states can be used when the agent encounters new states. The advantages of applying Deep RL are thus: 1) to be able to deal with larger environments 2) to be able to generalize from experience. The Deep RL method which will be utilized in this research is deep Q-learning.

We wish to construct a function $\hat{Q}(s,a)$ which best approximates the true $Q(s,a)$ function. In the Deep Q-learning algorithm, neural networks are used as an approximation function. In a classical neural network approach, the required data points for constructing such an approximation function $\hat{Q}(s,a)$, would be $((s,a),Q(s,a))$. The true values of $Q(s,a)$ are however not known. These values have to be approximated too. In this work, we will build a $Q(s,a)$-function approximator that takes a (s,a)-pair as input and gives a Q-value as an output. This variant of a function approximator is chosen because we will be dealing with a setting where the action space is state-dependent. Since it is not possible to train a network which has a variable output size, we will vectorize the (s,a)-pair and provide this as an input to the neural network. This NN has one output value, namely the value $Q(s,a) \in \mathbb{R}$. The estimated value for $Q(s,a)$ which is used to train the neural network, is called the Q-target. This Q-target is calculated via the TD(0)-update rule. In this setting the TD(0)-update rule becomes as in Equation (27). Q-targets are thus the estimate for the observed $Q(s,a)$-value for an (s,a)-pair and are used to train the NN. Note that the structure of the network is a feedforward neural network (FNN). Explaining the theory of FNN is out of scope for this thesis. If the reader wants to know more about FNNs, he is referred to Vera (2019-2020).

$$\tilde{Q}^{\pi}_{k+1}(s,a) = (1-\alpha)Q^{\pi}_{k}(s,a) + \alpha[R(s,a,s') + \gamma \max_{a' \in A(s')} Q^{\pi}_{k}(s',a')] \tag{27}$$

Where Q_k is thus the output from the neural network after iteration k, by plugging in $(s',a'), a' \in A(s')$. Recall that s' is the state that is reached after performing action a in state s. The outcome of Equation (27) $\tilde{Q}$ is what we call the Q-target in case of TD(0)-learning. The Q-values contain the subscript π because, states are visited and selected by an agent who is following a certain policy π.

When neural networks are used to approximate the Q-function, then TD(0) is not stable, or even tend to diverge (Mnih et al., 2015). This is partly due to the correlation between the Q values and the $\tilde{Q}$ target values. Mnih et al. (2015) propose a solution for this: don't update the Q-values each iteration but instead freeze the old $\tilde{Q}$ targets for a while and then update the neural network weights using a batch of targets. This will reduce the correlations between Q and the $\tilde{Q}$-targets used.

Therefore we will first gather a batch of observations and then calculate the $\tilde{Q}(s,a)$-targets for all these observations. The batch size is a hyperparameter which we will denote by BS. For this batch of training samples, now consisting of $((s,a), \tilde{Q}^{\pi}_{k+1}(s,a))$, the neural network is trained so as to minimize the loss function. This loss function is the mean squared error between the Q_k value and $\tilde{Q}_{k+1}(s,a))$. The weights of the neural network are thus updated and fitted using a training sample of size BS. The parameter BS, which stands for batch size, is a hyperparameter which determines

the number of required training samples on which the Q-network will be trained.

This constitutes the fundamentals for the Deep Q-learning method. Figure 4 provides an overview and summary of this algorithm.

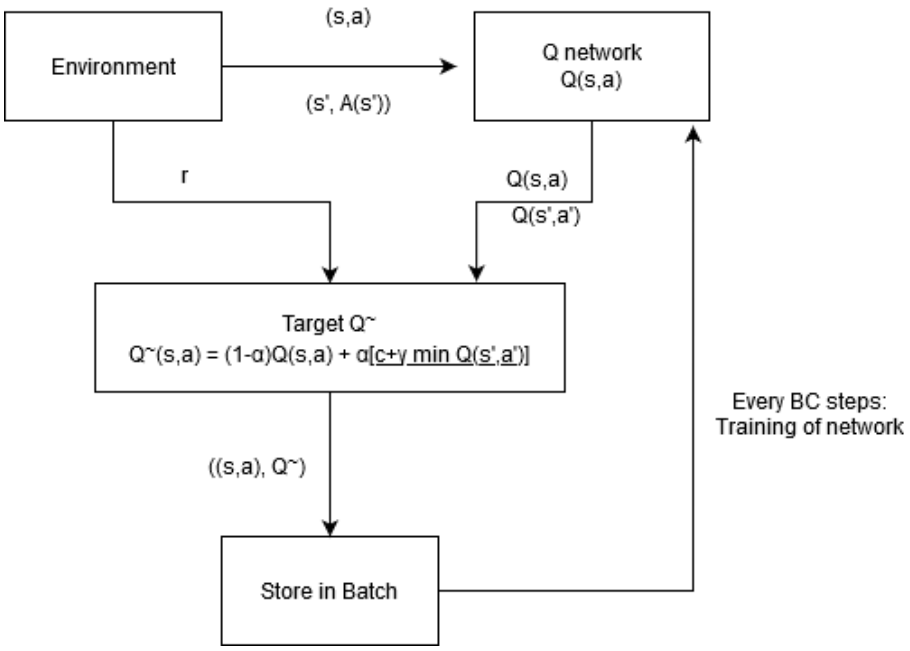


Figure 4: *This Figure summarizes the Deep Q-learning Algorithm. An agent interacts with the environment. Through this interaction, an experience tuple is gathered consisting of* $e = (s, a, r, s', A(s'))$*. These (s,a) and* $(s', A(s'))$ *are passed to the Q-network, which provides the current estimates Q(s,a) and Q(s',a') for all these (s,a) pairs. For each action* $a' \in A(s')$*, a* $Q(s', a')$ *value is estimated. Using these estimates and the received reward* r*, the* $\tilde{Q}$ *target can then be calculated using Equation (27). This* $((s, a), \tilde{Q}(s, a))$ *pair is then stored in a batch. This process is continued until the required number (BS) of training pairs are gathered. Then, the network is trained on this batch of training samples.*

6.4.2 Multi-Agent DQL

DRL techniques enable multiple autonomous agents to make successful decisions in a shared environment. In the Multi Agent Setting, each agent must learn his own Q-value function. Hence, for each agent an individual network will be trained. In Chapter 9 DQL is applied in a MA case. A pseudocode for the deep Q-learning solution method for that specific setting is provided in Appendix E.

7 Multi-Agent Reinforcement Learning Framework for a HFSP

In the previous chapters the concepts of reinforcement learning and a hybrid flow shop problem were introduced. In this chapter these theories will be used to build a multi-agent reinforcement learning framework for a HFSP.

The main idea behind the multi-agent framework for a HFSP, is to represent every machine in the flow shop as an agent. The agents make decisions on what job from the queue they should start to process. They act as independent learners and optimize their own policy towards minimizing their cost. The ultimate goal is that the agents act in such a way that the objective value of the resulting solution for a HFSP (i.e. the obtained schedule) is as good as possible.

To build up the multi-agent framework step by step, we start small and investigate the flow shop in a situation where just one machine is a learning agent in Section 7.2. The other machines will follow a predefined behaviour. We will start by investigating the situation where this predefined behaviour is deterministic. Subsequently we will study the situation where this behaviour is stochastic, to mimic the uncertainty in the environment which will be present in the multi agent situation as well.

Although this single agent setup does not represent a reasonable approach toward solving a real-life HFSP, it is useful towards building the multi-agent framework. It enables us to investigate the dynamics of reinforcement learning in a HFSP problem setting in detail, and to isolate the effects that different aspects of the problem have on the learning process.

Subsequently, in Section 7.3 we will show that this framework can be extended to a multi-agent setting, which is the proposed solution method for the HFSP.

Throughout this chapter, we will use a small case for illustration which will be introduced in Section 7.1. Results of experiments, obtained by applying the proposed framework to this small case, are used to support certain presumptions, and to provide deeper insights in the inner workings of the reinforcement learning algorithm. Generalization of the introduced framework to other cases, is considered straightforward. Therefore, in depth details concerning this are not always supplied for brevity's sake.

7.1 Problem Introduction

The problem environment consists of K departments, and has M^k machines in department $k \in K$. In this chapter a small HFSP setting will be considered. This small case can be easily extended to a larger HFSP. In fact, later in this thesis in Chapter 9 and 10 a three-stage HFSP is solved using this framework.

In this example, there are 2 departments $K = \{1, 2\}$ which each consists of 2 parallel machines. The cardinality of K and M^k is thus two: $|K| = 2$ and $|M^k| = 2, \forall k \in K$. The set of machines at department 1 is $M^1 = \{1.1, 1.2\}$ and the set of machines at department is $M^2 = \{2.1, 2.2\}$. Jobs must be processed in each department and in the same order: first department $k = 1$, then

department $k = 2$. In this example a case with two departments is considered, each consisting of two machines.

The set of jobs P contains all jobs that have to be processed by the system. Each job is of a certain type. There are PT different job types. In this example $PT = 2$. The type indicates the required processing time of a job per department. In this example, the job set contains four jobs: two jobs of type 1 and two jobs of type 2. The processing time per department differs for the different job types and can be found in Table 3. The problem setting is schematically shown in Figure 5. The machines within a department are identical. Each department has a queue in which the jobs are waiting that need to be processed in this department. The set of all queues is denoted as Q. The machines within a department share a common queue, where $q^k \in Q$ is the queue that belongs to department $k \in K$. This means that jobs that are waiting to be processed in an OT, are waiting in a common queue for department 1: q^1. This implies that they can either be processed by machine 1.1 or by machine 1.2. The queue q^k indicates how many jobs of each type are waiting in line to be processed by department $k \in K$. $q^1 = [0, 2]$ means that there are 0 jobs of type 1 and 2 jobs of type 2 waiting in the queue of department 1 ($k = 1$).

The set S contains all possible states of the environment. In this example, all possible states can be generated upfront. The set A contains all possible actions. Later in this chapter, these notions will be further elaborated upon. The sets and parameters that are used in this section are summarized in Table 1 and 2.

Set	**Index**	**Description**
K	k	Departments
M	m	Machines
P	p	jobs
Q	q	Queue
S	s	State set
A	a	Actions

Table 1: *The various sets that are used in the PTTR schedule and RL framework are defined in this table.*

Notation	**Meaning**
PT	Number of different job types

Table 2: *In this table the parameter that is used in this chapter is presented.*

Processing time	$k = 1$	$k = 2$
Job Type 1	4	4
Job Type 2	2	3

Table 3: *The required processing time at a department per job type can be found in this table.*

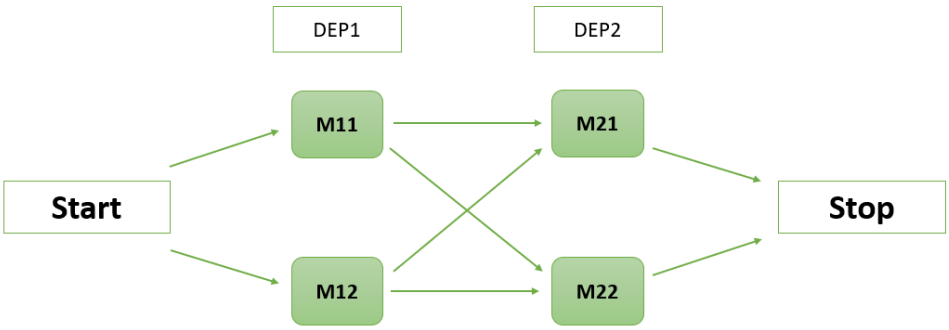


Figure 5: *Illustration of the problem setting which is used in this section. The scheduling environment consists out of 2 departments. Where each department has 2 parallel machines.*

7.2 Single Agent Setting

When building towards the Multi-Agent Reinforcement Learning framework for the HFSP, it's useful to first consider a setup with only a single agent. In this subsection, we will consider such a setup, where one of the machines in the Flow-Shop is a learning agent. The rest of the machines follow a predefined behaviour, which will be deterministic at first, and stochastic later. We will discuss the challenges and details of applying Reinforcement learning in this situations, and set the stage for the Multi-Agent approach.

7.2.1 Agent

A very important first step in the construction of the RL framework is the definition of agents. The goal of this chapter will be to research whether it is possible to solve the earlier introduced setting with a Multi-Agent RL framework. In this first solution approach, we will start of with a single agent RL setting. There is one single agent n_1 who represents Machine 1 in Department 1. All other machines always act in the same predefined way. Only the agent n_1 will learn by interacting with the environment. The behaviour of the other machines is fixed throughout this example.

The purpose of this example is to get familiar to the solution approach and to show that the approach under consideration works.

7.2.2 States

The system state $s \in S$ contains all relevant information about the environment for making a schedule. All relevant information at a specific moment in time consists of:

- A vector for each queue indicating how many jobs of each type are currently waiting in the queue: $q^k, \forall k \in K$
- The remaining processing time of for each machine in every department. If the remaining processing time of a machine is equal to zero, it means that this machine is available to start processing a new job.

The length of the queue vector is equal to the number of job types, in this example: $PT = 2$. $q^k = [2, 1]$ means that there are 2 jobs of type 1 and 1 job of type 2 waiting in q^k. Each department $k \in K$ has one queue which is shared between all machines in the department. Hence the cardinality of the set Q is equal to $|K|$.

For the remaining processing time per machine, the vector RPT is introduced. RPT is a vector of size $|M|$ that contains the remaining processing time for all machines. In this example RPT is thus a vector of size 4 as in total there are 4 machines. RPT_m denotes the remaining processing time for machine $m \in M$. RPT_1 then denotes the remaining processing time for machine 1 and thus for the agent n_1.

In this example, the job set contains four jobs. Two of type 1 and two jobs of type 2. When the jobs enter the system, all jobs enter the system in the queue of department 1: q^1. At the start the queue of department 2, q^2, is naturally empty as all jobs must first be processed by department 1. Hence, at the starting state the queues look as follows: $q^1 = [2, 2]$ and $q^2 = [0, 0]$.

The state size is dependent on a) the number of departments: this determines how many queue vectors there are; b) the number of different job types: this determines the length of the queue vector and c) the number of machines: this determines the length of the vector for the remaining processing times.

Example 3 provides an example of the state representation of the starting state of the environment.

Example 3 ***State Representation*** *There are 4 jobs present in the system. 2 jobs per job type. All jobs enter the system in the queue of department 1: q^1. At the start, all machines are idle and hence the remaining processing time for each machine is zero. The different state entities look as follows:*
$q^1 = [2, 2]$, $q^2 = [0, 0]$, $RPT = [0, 0, 0, 0]$
Combining the separate state entities, the state representation at time step zero is equal to:
$S_{start} = [2, 2, 0, 0, 0, 0, 0, 0]$.

7.2.3 Actions

Decision moments can be described in a discrete way. The decision of the agent consists of deciding what job from the queue he will start processing. The action of an agent thus consists of removing a job from the queue, and start processing this job. A decision moment arises when the agent is not processing a job, and at the same time there are jobs waiting in the queue. An agent can

only process one job at a time. The action set of the agent thus naturally depends on the jobs present in the queue. Each job type that is present in the queue, is a possible action for the agent. Additionally, to remain idle is also a possible action for an agent.

The action set $A^k(s)$ is defined by the jobs present in the queue q^k in state $s \in S$. $\forall i$ for which $q_i^k > 0$, $a_i \in A^k(s)$. Where a_i is a vector of size PT with a 1 at location i and zeros at all other places. a_i is the action of starting to process a job of type i. E.g. $a_1 = [1, 0]$ is the action of admitting a job of type 1. This will become more clear when looking at Example 4 below.

The agent corresponding to machine m can perform an action when the following criteria are both satisfied:

- The agent is not currently processing a job, i.e. $RPT_m = 0$.
- There are jobs waiting to be processed in the queue for department k. Hence, $\exists i, q_i^k > 0$, s.t. $0 < i \leq PT$, where k such that $m \in M^k$.

Recall, that a state $s \in S$ represents an observation of the environment in which the agent must make a decision. Hence, for each state $s \in S$ the above criteria must be satisfied. In this specific example where there is only one learning agent in department one, this implies that: Each $s \in S$ has $RPT_1 = 0$ and $q^1 \neq [0, 0]$. Note that the state representation in Example 3 is a valid state representation as it represents a situation in which the agent n_1 has to make a decision. Namely, 1) $RPT_1 = 0$ and 2) $q^1 \neq [0, 0]$.

The size of the action vector a_i for the agent is equal in size to the length of the queue and thus equal to PT. To perform an action is then equivalent to subtracting the action vector from the current queue.

When an agent performs an action, he will start processing a job. Naturally, the remaining processing time of this agent is then adjusted according to the processing time needed to process a job of the corresponding job type.

Example 4 provides an example of the action space.

Example 4 ***Action Space*** *State $s = [2, 2, 0, 0, 0, 0, 0, 0]$. Possible actions are then: 1) process a job with type 1: $a_1 = [1, 0]$; or 2) process a job of type 2: $a_2 = [0, 1]$. Additionally, 3) the agent can choose to remain Idle, and not process a job: $a_0 = [0, 0]$. The full action space is then $A(s) : \{[1, 0], [0, 1], [0, 0]\}$.*

Note that in Example 4, machine 2 in department 1 is also available to start processing a job. Recall that in this specific example, only machine 1 is a learning agent. The behaviour of all other machines is predefined. Therefore, this behaviour of the other machines can be considered as part of the environment. These other machines do affect the environment and through this they also have some effect on the interaction of agent 1 with the environment. However, the agent does not have

any influence on this. All he needs to know of the other machines should be and will be included in the state representation the agent observes.

7.2.4 Reward system

An objective function which is commonly used in scheduling problems, is the minimization of the total completion time of all jobs. This will also be the objective which we wish to pursue in this research. The goal is to design a reward function which strives to minimize the total completion time. Another way of putting this objective function, is that we wish that all jobs leave the system as fast as possible. We will base our cost function on the feedback signal proposed by Gabel and Riedmiller (2007). In their work it is stated the time span of a schedule is minimized if there are as few as possible machines with queued jobs in the system.

Hence, as a starting point we will define the cost function as the number of jobs that are present in the queues of the system after an action was performed. Note that in this case a cost function is used, which naturally implies that the agent's goal will be to minimize the expected cumulative cost. As opposed to using a reward function, where the goal is to maximize the expected cumulative reward. Intuitively: the agent gets a negative reward for each job which is present in the queue in the system. Therefore it is in the interest of the agent to process all jobs as fast as possible such that they leave the system. Hence this resembles the objective of minimizing the total completion time.

The cost depends on current state s, action a and new state s'. This new state s' is the next state in which the agent has to make a decision. The feedback signal that is communicated to the agent is the total number of jobs that are waiting in the queues in the system in state s'. The cost function is defined as follows:

$$C(s, \tilde{a}, s') = \sum_{k=1}^{|K|} \sum_{i}^{PT} q_i^k(s') \tag{28}$$

7.2.5 State Transitions

When the agent performs an action, the environment is affected in the following fashion. First, the queue decreases as the result of the action, a. Second, the remaining processing time for the agent is adjusted depending on the required processing time for the chosen job. In this example, all other machines choose an action based on their default actions.

The agent will receive a new state of the environment $s' \in S$ in which he has to make a new decision. Therefore, this s' represents the environment after the agent has finished processing the job. Once all available jobs have been processed by department 1. The agent will not have to process jobs anymore, and the episode ends.

Assumption 1 *If the agent chooses to remain idle, he remains idle until the next time jump has been made. This time jump happens first machine that is currently processing a job, finishes with*

his job. When he becomes available again, he does not release a job, but he will still receive a reward.

This means, that when $a = [0, 0]$, the next s' will be after x time steps. Where the number of time steps is determined according to Equation (29).

$$Timesteps = \begin{cases} \min RPT_{>0} & \text{if } \exists m, RPT_m > 0 \\ 0 & \text{Otherwise} \end{cases} \tag{29}$$

The implications of 1 will be illustrated by means of Example 5.

Example 5 ***To remain Idle*** *The state of the environment is* $s = (2, 2, 0, 0, 0, 0, 0, 0)$ *and the agent chooses to remain idle:* $a = (0, 0)$*. The state of the environment right after he chooses to remain idle is:* $s^{intermediate} = (1, 2, 0, 0, 0, 2, 0, 0)$*, i.e. machine* 2 *has started to process a job of type 1 (the processing times per job type and department can be found in Table 3). By definition,* $s^{intermediate}$ *would be a state in which the agent can make a decision. However, no time has passed since he has chosen to remain idle. Assumption 1 implies that the new decision moment* $s' = (1, 2, 1, 0, 0, 0, 0, 0)$*. Hence, the agent will remain idle until the first occupied machine becomes available again. Following Equation (29): he becomes available after* $\min_{>0}[0, 4, 0, 0] = 4$ *time-steps.*

7.2.6 Greedy Solution

Before the problem is solved using RL with learning agents, the specific example from Section 7.1 will be solved using a greedy method. In this greedy method, there will be no learning agents. Meaning all machines are kept dumb and will act in a predefined way. This solution will then be used as a benchmark, such that the solutions found when using learning agents can be compared to this benchmark.

Each machine starts processing the first available job he encounters, where first jobs of type 1 are processed. When there are no more jobs of type 1, the machines starts processing jobs of type 2, when available.

The resulting planning can be seen in Figure 19. The individual rewards in this planning can be found in Table 4. As one can see, each machine receives two rewards. Each machine thus 'makes a decision' twice. For this decision each machine follows the predefined rules. The total completion time of this planning is equal to $8 + 8 + 11 + 11 = 38$ time units.

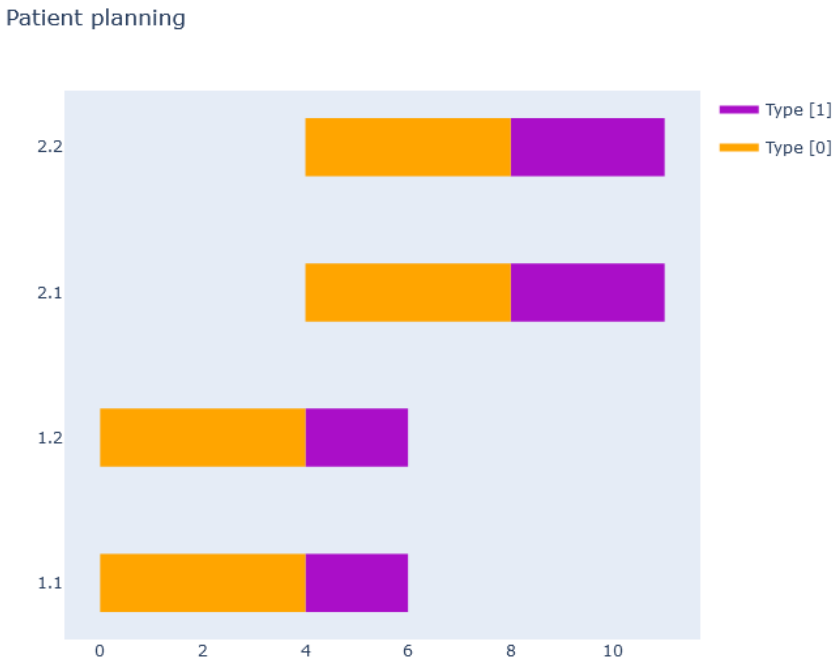


Figure 6: *Illustration of the planning which is the result of a greedy solution method. In this greedy solution approach, each machine starts processing the first he encounters. Here priority goes to s of type 1. When there are no more of type 1 that need to be processed, the machine will process a of type 2, when available.*

Machine	1.1	1.2	2.1	2.2
Reward	[4.0,2.0]	[4.0,2.0]	[2.0,0.0]	[2.0,0.0]
Total Reward	6.0	6.0	2.0	2.0
Total Completion Time	**38**			

Table 4: *The rewards that each machine receives in the solution found by the greedy method. Each machine receives a reward twice. The first reward for Machine 1.1 is 4: there are 4 s in the queue in s'. Note that s' is the next state of the environment in which new machines can start processing a job. The Total Reward is the total cumulative reward each machine receives in this setting.*

7.2.7 Implementation of Q-learning Algorithm in Case of 1 Learning Agent

Now all necessary RL concepts have been defined, the example can be solved by implementing the regular Q-learning Algorithm. In this solution approach agent n_1 corresponding to machine 1.1 will no longer act according to predefined rules, instead he will learn how to behave based on

experience he gains by interacting with the environment. The other machines will behave according to the same rules as in the greedy approach in section 7.2.6. Hence, the results of the Q-learning algorithm for 1 learning agent in this section can be compared to the results of the greedy approach. A Q-learning approach similar to the one in Section 6.1.8 will be used. First, the Q-Table will be initialized for agent n_1. Then, when all possible (s, a)- pairs are evaluated, the Q-table will be updated for each (s, a)-pair simultaneously.

The dimension of the Q-table is $|S|$ x $|A|$, 118 x 3. It is still possible to solve a problem of this size using regular Q-learning. However, for the sake of clarity, the full Q-Table will not be provided.

The problem at hand is an episodic task, therefore the discount factor is set to $\gamma = 1$.

In this setting the goal is to minimize the expected cost, therefore the equation for calculating the Q-values looks as in Equation (30).

$$Q^{i+1}(s, a) = C(s, a, s') + \gamma \min_{a' \in A(s')} Q^i(s', a') \tag{30}$$

where i denotes the i^{th} iteration.

The Q-learning solution approach for this setting will be illustrated by means of Example 6. This approach follows the pseudo code of the Q-learning Algorithm which can be found in Appendix B.

Example 6 ***Implementing the Q-learning algorithm***

First, initialize a Q-table of size 118×3 *with all zeroes.*

Iteration 0	*Q-Values*			
States	*(1,0)*	*(0,1)*	*(0,0)*	*Value*
(2,2,0,0,0,0,0,0)	*0*	*0*	*0*	*0*
(1,1,0,1,0,2,0,0)	*0*	*0*	*0*	*0*
⋮	⋮	⋮	⋮	⋮
(1,2,1,0,0,0,0,0)	*0*	*0*	*0*	*0*

Then, proceed as follows:

1. *For each* (s, a)*, observe* s' *and* $C(s, a, s')$
2. *Calculate* $Q^i(s, a)$ *for each* (s, a)*-pair, where* i *is the iteration*
3. *Update the Q-Table* $Q^i(s, a)$ *and update the V-vector*
4. *Repeat Step 1 - 3 until some convergence criterion is met*

Step 1*:*
For $(s, a) = ((2, 2, 0, 0, 0, 0, 0, 0), (1, 0))$*,* $s' = (0, 2, 2, 0, 0, 0, 0, 0)$ *and* $C(s, a, s') = 4$
Step 2*:*

$$Q^1((2, 2, 0, 0, 0, 0, 0, 0), (1, 0)) = C(s, a, s') + Q^0((2, 2, 0, 0, 0, 0, 0, 0), (1, 0)) = 4 + 0 = 0$$

Step 3*: The Q-Table after the first iteration looks as follows:*

Iteration 1	*Q-Values*			
States	*(1,0)*	*(0,1)*	*(0,0)*	*Value*
(2,2,0,0,0,0,0,0)	*4*	*3*	*4*	*3*
(1,1,0,1,0,2,0,0)	*2*	*3*	*3*	*2*
⋮	⋮	⋮	⋮	⋮
(1,2,1,0,0,0,0,0)	*2*	*2*	*3*	*1*

Step 4*: Then after the third iteration, the Q-values do not change anymore. Meaning we have found the optimal* $Q^*(s, a)$ *and* $V^*(s)$*. The Q-Table after the third iteration thus contains the optimal* $Q^*(s, a)$ *and* $V^*(s)$ *values, and looks as follows:*

Iteration 3	*Q-Values*			
States	*(1,0)*	*(0,1)*	*(0,0)*	*Value*
(2,2,0,0,0,0,0,0)	*6*	*5*	*6*	*5*
(1,1,0,1,0,2,0,0)	*2*	*4*	*4*	*2*
⋮	⋮	⋮	⋮	⋮
(1,2,1,0,0,0,0,0)	*2*	*4*	*5*	*2*

Step 5*: Determine* π^* *following Equation (32). The optimal actions for the agent are the ones corresponding to the grey cells in the optimal* $Q^* -$ *table above.*

In the first iteration, the Q-values are equal to the immediate return. This is due to the fact that all entries of $Q^0(s,a)$ are initialized as zero. Hence Equation (30) boils down to Equation (31).

$$Q^1(s,a) = C(s,a,s') \tag{31}$$

After 3 iterations, the Q-values and values do not change anymore. This means that the optimal $Q^*(s,a)$ and $V^*(s)$ values have been obtained. The corresponding optimal policy π^* can now also be derived. Recall, that in this example a cost function is considered instead of a reward function. Hence, the goal is to minimize the expected return. Therefore, Equation (32) defines the optimal policy.

$$\pi^*(s) = \min_{a \in A(s)} Q^*(s,a) \tag{32}$$

Now that the optimal policy for agent n_1 has been found, a new planning can be made. In this schedule machine 1.1 behaves according to the optimal policy found by the Q-learning algorithm in Example 6. All other machines behave according to the predefined rules from the Greedy method in section 7.2.6. The schedule corresponding to this solution is depicted in Appendix C.

The individual reward for each machine is tabulated in Table 5. Recall that the goal of agent n_1 a.k.a. machine 1.1 was to minimize his total reward. Compared to the greedy solution, the individual reward of machine 1.1 has indeed decreased by 1 unit. The total reward of machine 1.2 did also decrease by 1 unit. However, the individual reward of 1.1 and 1.2 have decreased at the expense of the reward of machine 2.1 and 2.2, whose rewards have increased. The combined reward for all machines has increased by 1 unit. The total completion time is now 8+11+5+10=34, and has decreased by 4 time units.

Machine	*1.1*	1.2	2.1	2.2
Reward	[3.0,2.0]	[3.0,2.0]	[2.0,1.0]	[2.0,2.0]
Total Reward	5.0	5.0	3.0	4.0
Total Completion Time	**34**			

Table 5: *This table displays the rewards that each machine receives in the planning found in the setting where machine 1.1 has learnt his optimal behaviour using the Q-learning Algorithm. Machines 1.2 - 2.2 all act in a deterministic way according to the predefined rules in Section 7.2.6. The goal of agent n_1 = machine 1.1 was to minimize his total reward. Compared to the greedy solution, the individual reward of machine 1.1 has indeed decreased. His reward has decreased at the expense of the individual reward of machine 2.1 and 2.2 whose rewards have increased. The total reward of machine 1.2 did also decrease by 1 unit. The overall reward has increased by 1 unit, where the total completion time of the total schedule has decreased by 4 time units.*

Let's zoom in on how agent n_1 has accomplished to decrease his individual reward. There are 2 s waiting in line of each type. Machine 1.2 always acts in the same predefined way: he first starts with processing a of type 1. If machine 1.1 also processes a of type 1, machine 1.1 and 1.2 are finished at the same moment in time with this operation. These s then both move the queue of department 2, q^2. Meaning both these s are in the queue at department 2 at the next decision moment. Hence the reward machine 1.1 receives for this action is equal to 4. Now what would happen if machine 1.1 chooses to process a of type 2 instead. Since the processing time of a of type

1 and 2 are different, machine 1.1 will finish processing this before machine 1.2 is done. Hence, at his next decision moment machine 1.2 is still processing the of type 1. There will thus be one less in the queue in the system, and the reward for machine 1.1 will be 3 instead of 4. The fact that in the optimal policy machine 1.1 thus chooses for this latter option, shows that optimizing based on the reward function and corresponding Q-values works.

7.2.8 Learning Under Uncertainty

The problem setting that has been studied in this section so far, was a deterministic setting. In this deterministic setting the Bellman Equation (Equation (18)) was used in the Q-learning algorithm. In Section 6.2 it was discussed that to deal with uncertainty in the environment, TD(0)-learning can be applied in the Q-learning algorithm. Equation (24) is then used to update the Q-values.

If Equation (24) is utilized in this deterministic example however, the Q-values still converge to the target values Q^* as in the case where no learning rate is used. This simply takes longer because the Q-values are updated in smaller steps. The largest value in the Q-table of the agent is 6. Using a learning rate of $\alpha = 0.1$, one would expect that the Q values converge after 60 iterations. In our experiments we see that the Q-values indeed converge. After 60 iterations, the optimal Q-values were approached with $\epsilon < 0.05$.

Introducing uncertainty:
Now let's consider an example in which the other machines no longer act in the deterministic predefined way, but instead choose randomly what type they will start processing. Due to this, the state transitions our agent observes are no longer deterministic. Instead, transitions from state s to s', given action a now occur according to the probabilities $\mathbb{P}(s'|s,a)$. This is what we refer to as uncertainty in the environment. In this setting the TD(0)-learning equations are used to retrieve and update the Q-values.

let's define in more detail how the other machines perform actions. The machines choose a random action, where the "remain idle" action is disregarded. Every of these actions is equally likely to be chosen. If only one non-idle action is possible, this action is chosen with probability 1. Example 7 illustrates this behaviour.

Example 7 $s = (2,2,0,0,0,0,0,0)$, *then for machine 1.2:* $\pi(a|s) = \begin{cases} 0.5 & \textit{for a=(1,0)} \\ 0.5 & \textit{for a=(0,1)} \end{cases}$

$s = (0,1,1,1,0,0,2,0)$, *then for machine 1.2:* $\pi(a|s) = \begin{cases} 1 & \textit{for a=(0,1)} \\ 0 & \textit{else} \end{cases}$

Recall that usually, in TD(0)-learning, Q-values are updated while interacting with the environment. Therefore, one has to make a decision about the action selection criteria up front, since the exploration-exploitation trade-off must be made. In this example however, the state space and the number of steps performed in one walk through the environment, are relatively small. Therefore,

we choose to evaluate all (s, a)-pairs in every iteration, avoiding the need for this action selection criteria.

The behaviour of the non-learning machines is not provided to the learning agent. Therefore, from the perspective of the agent, this causes uncertainty in its environment. Section 6.2.1 discussed the choice of the learning rate in settings with uncertainty. In this example the step size is set to $\alpha = 0.1$. Given that this learning rate does not satisfy the stated criteria for convergence, we expect the Q-values will never completely converge to Q^*. More likely, they will continue to vary in response to the most recently received rewards. From our experiments, we can indeed confirm this behaviour. For a few states, the state transition is still deterministic. This is true for states for which there is only 1 type in the queue. The actions of the other machines are in this case fixed. The Q-values for these states do perfectly converge. This is however not true for states in which there are two different type of jobs waiting in the queue and where the other machines thus choose randomly. To mitigate the absence of convergence, define the *convergence factor* as $CF_{k+1} = \sum_{(s,a)} |Q_{k+1}(s, a) - Q_k(s, a)|$. The learning is stopped when $|CF_{k+1} - CF_k| < \epsilon$, where ϵ is some small value.

Recall the findings for the deterministic setting of this example (Section 7.2.7). The agent was able to reduce his individual reward by choosing the opposite action as machine 1.2. Now that machine 1.2 chooses his action randomly, there is no strictly dominant action for the agent. In Appendix D, the table with $Q(s, a)$ and $V(s)$ are depicted for some illustrative states, found in the performed experiments. The Q-values show that the problem is the highly symmetric, as we find that for quite some states, the value of $Q(s, a_1) \approx Q(s, a_2)$. If one would continue the learning, the best policy continues to change. These changes happen in states where the Q-values are practically equal to each other. A small increase in one of the Q-values therefore might change the best policy. In the scope of this thesis, this is not a big issue, as long as the objective value of the final schedule is not affected. Given that in such a situation the two actions have a very comparable Q-value, and are therefore expected to yield a similar performing schedule. Even though the Q-values and the corresponding best policy for the agent do not converge perfectly it is safe to state that the agent did learn what is the best way to behave in each state.

Agent n_1 has learnt how to behave in each state. This knowledge is stored in the Q-table for agent n_1. To show that the reward really depends on what actions the other machines take, 5 different schedules are generated, and the corresponding outcomes are shown in Table 6.

Rewards: per machine	1.1	1.2	2.1	2.2
Schedule 1	[4.0,2.0]	[4.0,2.0]	[2.0,0.0]	[2.0,0.0]
Schedule 2	[3.0,1.0,1.0]	[3.0,2.0]	[2.0,0.0]	[2.0,0.0]
Schedule 3	[4.0,2.0]	[4.0,2.0]	[2.0,0.0]	[2.0,0.0]
Schedule 4	[3.0,1.0]	[3.0,3.0,1.0]	[3.0,1.0]	[3.0,1.0]
Schedule 5	[4.0,2.0]	[4.0,2.0]	[2.0,0.0]	[2.0,0.0]

Table 6: *5 schedules are constructed where the agent (machine 1.1) chooses an action according to the optimal policy, using the Q(s,a) table which was constructed in this section. The other machines choose a random action. The fact that the schedules are different is due to the randomness in the choices of the other machines. The table depicts the individual reward per machine per schedule.*

7.3 Framework for the Multi-Agent Setting

In this section the single agent setting will step by step be extended to the multi-agent setting, arriving at the solution method for the HFSP proposed in this thesis. In Section 7.2.8 we discussed how agent n_1 learns how to behave in an environment where all other agents act randomly. Now, there are multiple learning agents that will also learn how to behave in the same environment. This is a cooperating system where the agents have the same goal, minimizing the number of s in the queue. Each agent will pursue this goal by minimizing his individual cost.

The structure of this Section is as follows: We will start by introducing the notion of active agents. In the multi agent setting, namely not every agent has to take a decision in each state. In the situation where there are multiple agents, the actions of other machines are no longer part of the environment. Naturally, these are influenced by the learning agents. We will therefore show how a state transition works and how a cost is allocated. Then, in subsection 7.3.1 the learning process in a multi-agent setting is described, together with all necessary modeling choices. To ensure that we understand the dynamics of the interaction between agents, we make the transition from a single agent to four agents, step by step. We will introduce a second learning agent. First we will consider two learning agents in consecutive departments in subsection 7.3.2. Second, in subsection 7.3.3 we will consider two learning agents within the same department. After, all of this has been discussed we will be ready to move to a setting where all machines have a learning agent attached to it which will be studied in subsection 7.3.4.

In addition to the sets introduced in Table 1, we introduce the set N. The set N consists of all agents. Not all agents have to make a decision in a state s, as discussed below. Therefor, we introduce the set $AN^s \subset N$ being all active agents in state $s \in S$. Active agents are all the agents that have to make a decision in state $s \in S$.

An agent $n \in N$ is in AN^s if the following two statements are true:

- The agent n is not currently processing a job, i.e. $RPT_n = 0$
- There are jobs waiting to be processed in the queue for the department of agent n. $\exists i$ s.t. $q_i^k > 0$, where k is such that $n \in M^k$

Each agent determines its action independently of the other agents. Meaning they choose an action simultaneously, and without and before knowing the action of the other agents. Agent n chooses an action $a_n \in A^k(s)$, for which $n \in M^k$. For the state transition, the actions of all active agents are combined into a joint action $\tilde{a} = \{a_n\}, \forall n \in AN^s$. The joint action is then passed on to the environment.

The actions are performed in an iterative fashion. Meaning, first the action of the agent with the lowest index n is performed. The individual actions are applied to the environment as follows. First, the queues of state s decrease as a result of the $\tilde{a}$. Second, the remaining processing time for the agent that performs the action is adjusted. The resulting state representation might be such that there are no active agents in this setting. Since a new state s' should be a representation of the environment where a decision must be made by at least one agent, there must at least be one active

agent. If this is not yet the case, we a time step such that one or multiple agents become available, resulting in the new state s'. Example 8 and Figure 7 illustrate how these state transitions work.

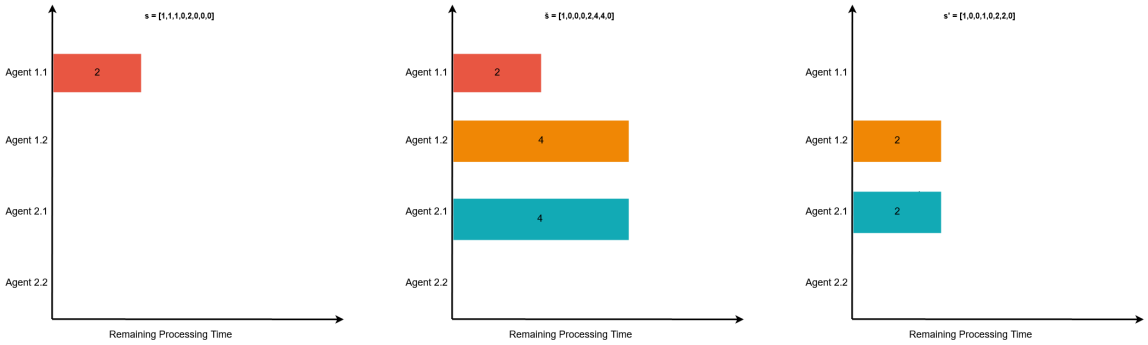


Figure 7: *Example of a state transition.*

Example 8 ***State transition*** *State* $= [1, 1, 1, 0, 2, 0, 0, 0]$. $q^1 = [1, 1]$, $q^2 = [1, 0]$. *Active agents are* n_2, n_3 *and* n_4. *Suppose that the joint action for the active agents is:* $\tilde{a} = \{n_2 : [0, 1], n_3 : [1, 0], n_4 : [0, 0]\}$. *When this joint action is performed,* n_2 *and* n_3 *start processing a . These s are removed from the queues and* RPT_2 *and* RPT_3 *are adjusted. This intermediate setting of the environment is depicted in the middle image of Figure 7. Note that there are no active agents in this setting. Hence this is not a valid* s'. *Therefore we must make a jump in time, to the moment where the first agent becomes available, which is after a time step of 2 time units. This time step is obtained by taking the minimum of the nonzero remaining processing times. The remaining times and the queues are adjusted and the new state* $s' = [1, 0, 0, 1, 0, 2, 2, 0]$ *is obtained which is depicted in the right image in Figure 7.*

In this multi-agent setting we wish to utilize the same cost function as was used in the single agent case, depicted by Equation (28). Each agent continues to pursue an individual goal, namely maximizing/minimizing his individual reward/cost. In this multi-agent setting we must discuss the moment on which the cost is allocated. In a single agent setting, costs are allocated based on s', the state that arises as effect of the action that the agent took. The next observed state and the next decision moment for the agent are always the same. In a multi agent situation, this is not always the case. Before one agent's next decision moment arises, we might observe other states as well, in which other agents need to take actions. To make the analogy between the single and multi agent situation, we choose to allocate the cost for an action taken by an agent, based on the next observed state in which that agent again needs to take an action.

Let's consider our example. As we saw, not every agent has to make a decision in every new state which arises while iterating through the environment. In Example 8 the newly obtained s' is a new decision moment for both agent n_1 and n_4. Hence both these agent will receive a cost based on this new state $s' = [1, 0, 0, 1, 0, 2, 2, 0]$, as this is the next action moment since they last took an action. Note thus the states in which agent n_1 and n_4 took their action are different, yet they both receive the reward based on this s'. They will use this received cost to update their knowledge on

state-actions pairs corresponding to these different state. In the considered example, the cost is for both n_1 and n_4 equal to $C(s^{n_1}, \tilde{a}, s') = C(s^{n_4}, \tilde{a}, s') = 1 + 1 = 2$.

7.3.1 Learning in a Multi-Agent Setting

Recall that we wish to build a multi-agent system where the agents learn how to jointly behave. Therefore we will let the agents learn simultaneously while interacting with the environment. Where in the previous section we evaluated each (s,a)-pair for the agent, here this is no longer possible and desirable. Therefore we move to temporal difference learning, where Q-values are updated along the trajectory. This implies that we let the agents interact with the environment, and as the states occur the Q-values for these states are updated in the physical Q-Table. States that are visited more frequently are also updated more often. An additional advantage of moving to TD-learning is that it can also be applied in a bigger environment.

Now that we no longer evaluate each (s,a)-pair, we must choose how the agents select their action when in a state. The action-selection criterion that will be used in this section is the ϵ-greedy action approach. With a linear ϵ decay. We start off with very exploratory agents: $\epsilon^{max} = 1$ and end with very greedily acting agents: $\epsilon^{min} = 0.1$.

A large fixed number of training episodes is set. Epsilon ϵ is then decayed after each completed episode following Equation (33). Note that this epsilon decay happens at the end of each episode. This epsilon is a general epsilon used for each agent. It could be that one agent has performed more actions than an other agent. In such a small example as considered here, this is not yet relevant. Future work could investigate whether an agent-dependent epsilon improves the performance in bigger instances.

$$\epsilon_i = \epsilon_{i-1} - \frac{\epsilon^{max} - \epsilon^{min}}{\#TrainingEpisodes} \tag{33}$$

where i represents the current episode.

Table 7 presents a summary of all modeling decisions which are made in this section.

MARL 2 agents	
Action Selection Method	Epsilon greedy
Q-values calculation method	TD(0)-learning
Training Stopping Criterion	Fixed # of Training Episodes
Storage of Q-values	Q-Table

Table 7: *Summary of the characteristics of the solution approach which is used in this subsection.*

In the upcoming sections we will implement the next step towards building a full MARL system. A second learning agent will be introduced. Still, the case from section 7.1 is used. In this small example, there is a different relationship between machines in consecutive departments and between machines within a department, who share a common queue. Therefore we make a distinction here between two agents in consecutive and two agents within the same department.

7.3.2 Agents in Consecutive Departments

As explained above, a second agent is introduced such that there are 2 agents in different departments. Next to agent n_1, machine 1 in department 2 becomes a learning agent too, which we will call n_3.

The goal of the upcoming section is to evaluate the dynamics of the system, when there are two learning agents, in consecutive departments. The remaining two machines still have the same predefined behaviour as in the previous section. We will evaluate the influence of these learning agents on the outcomes.

Due to small character of the environment, all outcomes are pretty close to each other. Hence we do not expect to see much difference with respect to the setting where there was one learning agent, and the others choose a random non-idle action. In this small environment, the best action for each agent is namely always to start processing a job and not remain idle. Hence, there is in fact not much that the agent n_3 can improve. Therefore one can also not expect any improvements in this section.

The dependency of agent n_3 on the learning of agent n_1 is not very different from the case in which agent n_1 would act randomly. Together with machine 1.2, agent n_1 determines what s enter the queue for department 2. Hence agent n_1 is (partly) responsible for the state in which n_3 gets to make a decision. But this is not much different then in the case that the actions of the agents/machines in the first department would randomly process s. The learning of agent n_1 does have an advantage. Namely: As the training phase continues, both agents will start to exploit the experience they have gained. This will make the state transitions less uncertain.

When running experiments, we see that as expected the results in terms of total completion time and obtained individual rewards are very similar to the setting in which only n_1 was a learning agent.

7.3.3 Agents Within the Same Department

In this subsection, the multi agent approach is applied for 2 learning agents within department 1: agent n_1 and agent n_2. These agents share a common queue. Hence, it is desirable that these agents learn to adjust their behaviour to each other.In this subsection we will evaluate whether the agents can learn to adjust their behaviour to each other. With this we validate the use of the proposed MARL approach.

Recall that for the reward function used in this chapter, it was beneficial for an agent from department 1 to not finish the job at the same time as the other agent of department 1. Namely, in this case there would be 1 less waiting in line as this would still be in process. This is also reflected by the Q-values in Table 8. Namely, in the start state $s = [2, 2, 0, 0, 0, 0, 0, 0]$, the best action for agent $n_1 = [0, 1]$ and for action n_2, the best action is: $[1, 0]$. Note that these results depend on the randomness in the training run. After a new training phase it could be that the agents both prefer the other action.

One notices that now there are 2 learning agents within one department, the agents really adjust their behaviour to each other. Recall that these agents share a joint queue. Looking at the Q-Table for agent n_1 and n_2, what one can see is that the actions of these agents align. For states where the queue for department 1 looks as follows: $q^1 = [1, 1]$, one agent prefers action $[1, 0]$ and the other prefers $[0, 1]$. Note that if both agents choose to start processing a different , this is beneficial for the reward for both agents. This is again verified by the Q-values presented in Table 8. What is also noteworthy, is that the value in each state are more or less similar for both agents. This again shows the symmetric nature of the environment.

	Q(s,a) values for all possible actions	
State	n_1: Q(s;[1,0],[0,1],[0,0])	n_2: Q(s;[1,0],[0,1],[0,0])
[2. 2. 0. 0. 0. 0. 0. 0.]	[6.10157484 5.1703164 8.04824576]	[4.99262253 5.96305823 6.35051259]
[1. 1. 0. 0. 0. 0. 0. 1.]	[1.32718695 1.07533302 2.4193213]	[1.18098422 1.74287277 2.34575616]
[1. 1. 0. 0. 0. 0. 0. 0.]	[1.80677529 2.38924275 3.16558703]	[2.43229516 1.82794521 3.12347153]
[1. 1. 0. 0. 0. 0. 0. 3.]	[1.00314282 2.10946117 2.25131907]	[1.82803069 1.12664744 2.14719863]

Table 8: This Table displays the Q-values for some illustrative states for both agent n_1 and n_2.

Challenges MARL setting
Recall the challenges of the MARL setting which were discussed in Section 6.3.3. Non-stationarity is always a potential issue in the MARL. What we notice in this example however, is that the agents manage to adjust their behaviour to each other. Hence in this very small example the non-stationarity issue is not present. This might however be due to the small nature of the example setting. As non-stationarity is no issue at this point, we do not have a reason to assume that the proposed solution method is not suitable to apply in a MARL setting. We will thus apply this same method to a larger use case later in this work, where non-stationarity might be a bigger issue.

Symmetry in the Environment
An important note that should be made is that the problem is of a very symmetrical nature. Note that the machines within a department are identical. In the solution found in this setting we see that: In the state (1,1,0,0,0,0,0,0) Agent n_1 chooses to start processing a of type 2, and agent n_2 chooses a of type 1. If this would be the other way around (n_1 type 1 and n_2 type 2), then the resulting solution would be exactly the same. There is thus by no means a unique optimal policy in a problem of this nature. This is something that we should keep in mind, as this also has an influence on the learning behaviour of the Q-values. It is inevitable that the agents enter a cycle of endless adjustment to each other. Hence at some point we should truncate the learning process.

Hence once a good solution has been found and the Q-values start to readjust and move towards a different solution, we should stop the learning process. From experiments we have seen that the Q-values will tend to move towards a different local optimum. Note that although there might not be a unique optimal policy, there is a unique global solution. Hence, to validate a found solution, the training process should be repeated multiple times. If there is randomness in the environment, then in order to evaluate a found best policy, multiple test runs should be performed.

7.3.4 4 Learning Agents

With all the building blocks from this chapter, we are now ready to implement the full multi-agent setting. We will again consider the example from Section 7.1. Now each machine is represented by a learning agent.

The nature of the results for agent n_1 and n_2 is the same as in 7.3.3 above. The same results are found for agent n_3 and agent n_4. In case that the queue of department 2 contains one job of each type: $q^2 = [1, 1]$, the actions of n_3 and n_4 are aligned with each other. This is illustrated for some states in Table 9. For the state (0,0,1,2,2,0,0,0) where $q^2 = [1, 2]$, one notices that n_3 prefers action $[0, 1]$, and the other agent is indifferent. The agents have adjusted their behaviour to each other well.

	Q(s,a) values for all possible actions	
State	n_3: Q(s;[1,0],[0,1],[0,0])	n_4: Q(s;[1,0],[0,1],[0,0])
[1. 0. 1. 1. 0. 0. 0. 0.]	[2.23179873 1.54560238 2.50743769]	[1.53906926 1.94526851 2.26994905]
[0. 1. 1. 1. 0. 0. 0. 0.]	[1.13648784 1.95872847 3.33698824]	[2.55515967 1.22838167 3.27464263]
[0. 0. 1. 2. 2. 0. 0. 0.]	[2.8539604 2.33619814 4.52696991]	[2.85841995 2.74458128 4.38348567]
[0. 2. 2. 0. 0. 0. 0. 0.]	[2.13710017 x* 5.12174609]	[2.11979195 x* 5.10261633]

**x: indicates that this action is invalid for this agent in this state*

Table 9: This Table displays the Q-values for some illustrative states for both agent n_3 and n_4.

7.4 Summary

In this chapter we have built a multi-agent framework for solving a HFSP problem. Each machine is represented by an agent. Each agent learns how to behave, in terms of, which job it should start processing when in a certain state, to arrive at a wel performing final schedule. It was shown that the agents are able to adjust their behaviour to each other. Moreover, the system of agents was able to construct a good schedule.

We again would like to note that the illustrative case that was used throughout this chapter is too small to show the real power of MARL approach. It served for the purpose of illustrating the methodology. In the remainder of this thesis, we will look into the performance of the MARL framework, when applied to cases that more resemble real life situations.

8 Hospital planning problem formulated as HFSP

It is generally accepted in literature that "the Operation Theater (OT) is the most costly resource in the hospital", that accounts for 40% of hospital costs (Macario et al., 1995). Therefore a good schedule is of utmost importance to use this valuable resource efficiently. However, as was extensively discussed in Section 5.1, in order to ensure an overall good operating hospital, it cannot be avoided to also take into account the capacity and occupation of other resources as well. With this in mind, a new formulation for the hospital OT scheduling flow is proposed in this chapter. We will show that the hospital flow can be defined as a three-stage no-wait hybrid flow shop model. The three stage here represents the three hospital departments which a patient passes through: the Operation Theater (OT), the Intensive Care (IC) and the Medium Care (MC) department. Later in this thesis, in Chapter 10, this formulation will be used to find solutions for instances of a hospital planning problem.

This chapter is structured as follows. In Section 8.1 we will first describe the hospital scheduling setting in detail. This setting will be introduced using the framework of Samudra et al. (2016) such that all relevant aspects are discussed. Then, Section 8.2 will elaborate upon how the hospital flow scheduling problem can be modelled as a three-stage no-wait HFSP. Here, also a mathematical formulation will be provided. This will be based on the hybrid flow shop theory from Section 5.2. Finally, Section 8.3 will describe how a specific data set can be simulated based on real hospital data, which is based on the detailed data description presented by Dellaert and Jeunet (2010).

8.1 Hospital Scheduling setting

In this section, the hospital scheduling setting will be described using the framework of Samudra et al. (2016). We follow this framework such that all relevant aspects are included. The hospital description is based on the detailed description of the Thorax Centre Rotterdam provided by Dellaert and Jeunet (2010).

8.1.1 Problem Setting

Type of Patients
As discussed in Section 5.1.1, in existing literature there is no unambiguous answer as to what is the best way to incorporate emergency patients in your model. Therefore, we see at this moment no point in deviating from the approach followed by Dellaert and Jeunet (2010). Following their example, a fixed amount of capacity will be reserved for emergency patients. This implies that we do not incorporate emergency patients in our model. The focus of this thesis will thus be on the scheduling of elective patients.

Type of scheduling
The type of scheduling determines for who the schedule is made. The scope of this thesis will be on building a patient schedule. Here we will focus on constructing both the patient-to-day and the patient-to-day-and-time schedule simultaneously.

Hospital flow: Up- and Downstream facilities
The hospital scheduling problem which we will consider consists out of three departments: the OT, IC and MC departments. These departments are based upon the description of the hospital flow in the Thorax Center Rotterdam provided by Dellaert and Jeunet (2010). Patients are admitted straight into the OT. After the operation has finished, patients stay for some days at the IC. After their IC stay patients stay for some additional days in the MC for recovery. Since apart from the OT department also the IC and MC department are taken into account, patients are only admitted to the hospital if the availability of all three departments allow for this.

Objective
We wish to construct an efficient schedule in which all patients are processed as fast as possible. The objective used when constructing a schedule is therefore to minimize the total completion time of all patients in the schedule.

Deterministic vs Stochastic
For now, a deterministic model is assumed. A predetermined and hence deterministic OT duration and stay at IC and MC of elective patients is used. The duration of these stays are dependent on the patient type. The uncertain arrival of non-elective patients is also not covered at this point.

Mathematical Model
The hospital OT scheduling problem will be modeled as a three-stage no-wait hybrid flow shop. This formulation will be introduced in the next section.

Solution method
The solution method which will be used to solve the hospital scheduling problem is the in Chapter 7 proposed Multi-Agent Reinforcement Learning Framework for solving a HFSP. The no-wait constraint in the hospital setting will require some adjustments to the framework. These adjustments and the implementation in a hospital setting will be extensively discussed in Chapter 10.

A summary of the problem setting is presented in Table 10. This summary presents a clear overview of the exact hospital case study which will be used in the remainder of this thesis.

Type of patients	Elective patients
Up-and Downstream Facilities	Yes. Includes OT, IC and MC
Objective	Minimize the Total Completion Time
Type of Scheduling	Patient scheduling: both patient-to-day and patient-to-room-and-time scheduling
Mathematical Model	3-stage no-wait HFSP Deterministic model, Integer Linear Programming
Solution Method	Multi-Agent Reinforcement Learning

Table 10: *Summary of the problem setting considered in this thesis.*

8.2 Formulation of Hospital Flow using a HFSP

In this section, the hospital setting described by Table 10 is defined as a three-stage no-wait hybrid flow shop. Additionally, a mathematical formulation will be defined.

Department 1 are the operating theaters, department 2 is the IC department and department 3 the MC department. The jobs are patients, and the machines are either operation theatres or IC/MC beds. The set of N jobs to be processed, are now a set of N patients that have to be operated. The department S are the different hospital departments of which there are three. Namely, the OT, the IC and the MC.
The standard HFSP criteria which were defined in Section 5.2 are all satisfied:

- The hospital environment consists out of 3 stages/departments.
- Each departments consists out of multiple machines which can operate in parallel.
- Each patient is treated in the same processing order. Each patient starts in the OT, transfers to the IC and afterwards to the MC

The following additional assumptions are made:

1. Each OT has the same capacity and has the same surgery and equipment availability. Meaning all "machines" within stage 1 are identical.
2. The beds within the IC stage and within the MC stage are identical. Hence, for department 2 and 3 it also holds that all machines are identical.
3. An OT and a bed can be occupied by only 1 patient at the same time.
4. All scheduled patients are available on the given day. Meaning they are already present in the hospital and all necessary preparations have been done.
5. An operation or a treatment at the IC/MC cannot be interrupted.
6. The preparation and cleanup time are included in the operation duration.
7. There is no transportation time in between departments.
8. All patients have to be processed by each department.
9. Processing times are deterministic and known upfront.
10. No-wait assumption: There is no queue capacity in between departments. This implies that patients must wait in the previous stage until he can be processed in the next department.
11. The number of patients per type in need of an operation is assumed to be known, when constructing the schedule.

MILP Formulation for the three-stage HFSP
The mathematical model is very similar to the general model for a HFSP formulation. We must make one adjustment to satisfy Assumption 10. This assumption assures that there can be no time between the processing of a patient in between two stages. Hence constraint (36) contains an equality sign, such that the no-wait property is satisfied. The provided mathematical formulation can be generalized beyond this setting, and can be applied to any hospital which wishes to model the flow through multiple departments.

Sets	
P	The set of patients to be processed, $P = \{1, ..., N\}$, N total number of patients
S	The set of hospital departments through which a patient must to be processed $S = \{1 : OT, 2 : IC, 3 : MC\}$
M_s	The set of operation theaters or beds in department $s \in S$

Hence $|M_s|$ thus represents the capacity of department $s \in S$, i.e. the number of OTs/beds in a department s.

Parameters
d_{is}: processing time of patient i in department s*
**Note that here it is thus assumed that the processing times are known beforehand.*

Decision Variables

$$Y_{isl} = \begin{cases} 1 & \text{if patient } i \in P \text{ is scheduled on OT/bed } l \in M_s \text{ in department } s \in S \\ 0 & \text{otherwise} \end{cases}$$

$$X_{ijs} = \begin{cases} 1 & \text{if patient } i \in P \text{ precedes patient } j \in P \text{ in department } s \in S \\ 0 & \text{otherwise} \end{cases}$$

$C_{i,s}$: completion time of patient $i \in P$ at department $s \in S$, and let $C_{i,0} = 0$

Let $Q \geq \sum_{i,s} |M_s| d_{is}$ be a large number, but make sure it is chosen not too large. Choosing a Q which is too big, might lead to violation of constraints (37) and/or (38).

Then the model can be formulated as follows:

$$\min \sum_{i \in P} C_{i,MC} \tag{34}$$

$$st. \sum_{l \in M_s} Y_{isl} = 1 \qquad \forall s \in S, i \in P \tag{35}$$

$$C_{i,s} - C_{i,s-1} = d_{is} \qquad \forall i \in P, s \in S \tag{36}$$

$$Q(2 - Y_{isl} - Y_{jsl} + X_{ijs}) + C_{is} - C_{js} \geq d_{is} \qquad \forall i, j \in P, \forall l \in M_s, \forall s \in S \tag{37}$$

$$Q(3 - Y_{isl} - Y_{jsl} - X_{ijs}) + C_{js} - C_{is} \geq d_{js} \qquad \forall i, j \in P, \forall l \in M_s, \forall s \in S \tag{38}$$

$$Y_{isl} \in \{0, 1\} \qquad \forall i \in P, s \in S, l \in M_s \tag{39}$$

$$X_{ijs} \in \{0, 1\} \qquad \forall i, j \in P, s \in S \tag{40}$$

$$C_{i,s} \geq 0 \qquad \forall i \in P, s \in S \tag{41}$$

The objective function of the model is to minimize the total completion time of all patients. This is equivalent to minimizing the cumulative finishing time at the MC department which is done by (34). Constraint (35) ensures that a job will only be assigned to one machine (OT/bed) at each department. Constraint (36) is the no-wait constraint, and restricts the starting time of the treatment at department s to be equal than the finishing time of the previous department. It does this by setting the finishing time of department $s+1$ equal to the finishing time at the department s plus the required processing time in department $s+1$. The following paragraph will illustrate how the Constraints (37) and (38) make sure that no two jobs are being assigned to the same capacity space at the same time.

If two patients i and j are both assigned to the same machine l in stage s, meaning $Y_{isl} = Y_{isl} =1$, and they would be erroneously scheduled at the same time, then X_{ijs} would be zero since i does not preceed j in stage s. However, in this situation it would thus hold that $(2-Y_{isl}-Y_{jsl}+X_{ijs}) = 0$, and thus to satisfy Constraint (4) it should hold that $C_{is} \geq C_{js} + d_{is}$. Meaning, the completion of job i in stage s must be larger than the completion time of job j plus the processing time of job i at stage s. Hence, this implies that job j must be processed before job i in stage s, contradicting the illustrated situation of i and j being scheduled at the same time. In this situation, where j thus precedes i, the value of $Q(3 - Y_{isl} - Y_{jsl} - X_{ijs}) =1$, hence in this setting Equation (38) also holds because Q was chosen to be a large number. Likewise, in the setting where patient i does precede j, it holds that $X_{ijs} =1$. Hence we have that $(2-Y_{isl}-Y_{jsl}+X_{ijs}) \neq 0$ and thus constraint (37) holds. In this setting, the equation $(3 - Y_{isl} - Y_{jsl} - X_{ijs})$ equals 0, therefore requiring that $C_{js} - C_{is} \geq d_{js}$ holds. Through this it makes sure that the processing of patient j does not start before the completion time of patient i. In the setting where the patients are not both assigned to the same machine in a department s, both $(2 - Y_{isl} - Y_{jsl} + X_{ijs})$ and $(3 - Y_{isl} - Y_{jsl} - X_{ijs})$ are not equal to zero, hence Equation (37) and (38) both hold due to the value of Q is chosen larger than any difference between completion times.

Constraints (37) and (38) thus together make sure that no jobs are being operated at the same OT simultaneously, and there are no jobs lying at the same bed at the same time. Finally, constraint (39), (40) and (41) define the domain of the decision variables Y_{isl}, X_{ijs} and $C_{i,s}$.

Figure 8 provides an example of a three-stage HFSP for a hospital setting satisfying all the above

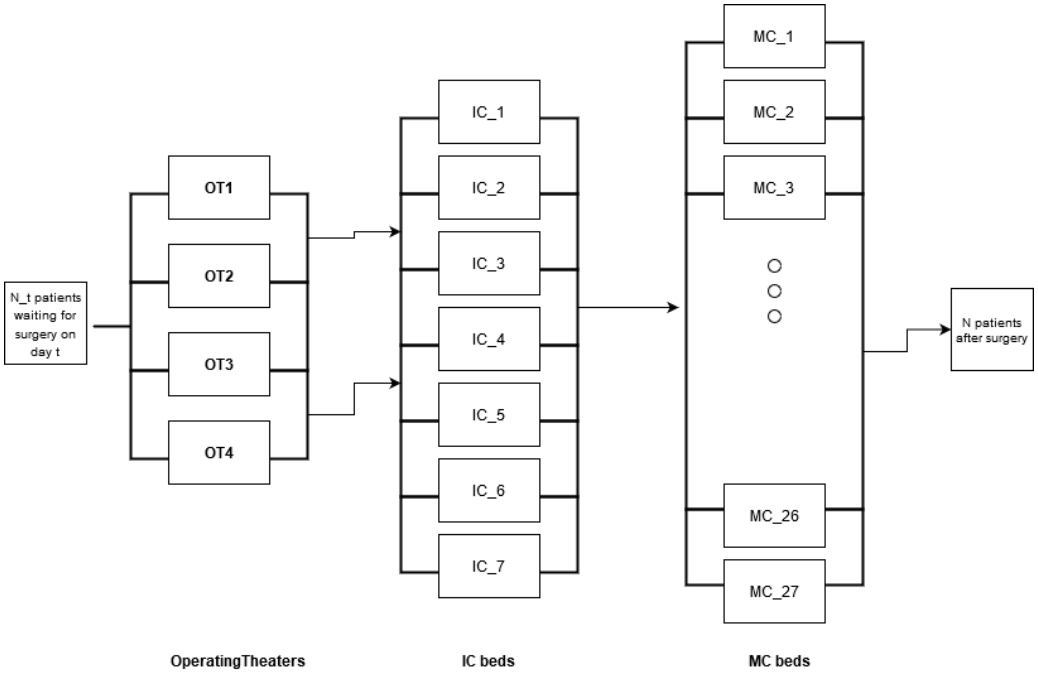


Figure 8: *Illustration of a three-stage HFSP hospital setting which satisfies the above assumptions. In the depicted HFSP, there are 4 OTs:* $|M_{OT}| = 4$*, 7 IC beds:* $|M_{IC}| = 7$*, and 27 MC beds:* $|M_{MC}| = 27$

mentioned assumptions. Note that this figure illustrates a specific case study, and serves for illustrative purposes only. The provided problem setting and mathematical formulation are applicable to each no-wait HFSP and can be generalized beyond this example.

8.3 Data Generation

Based upon the detailed data description presented by Dellaert and Jeunet (2010), a data set can be simulated. For this setting, all the above mentioned assumptions are taken into account. Data is retrieved from the Thorax Centre Rotterdam and is based upon empirical data of 576 patients in 2006 (Dellaert & Jeunet, 2010). All reported data concerns a 4-week period.

In this section we will describe how a data set can be generated using the information of Dellaert and Jeunet (2010).

Patients
Patients are divided in 8 groups. Appendix A Table 18 provides information about the different groups and also provides an example procedure. All patients within a group are relatively similar in terms of OT duration and IC/MC requirements. The probabilities for the length of stay of a

patient at the IC can be found in Appendix A Table 16. The probabilities for the length of stay of a patient at the MC department can be found in Appendix A Table 17. The available information about the different patient groups considers a 28 days period. For every group we have information regarding the expected number of arriving patients and the expected duration of the operation at the OT. This information can be found in Appendix A Table 15.

Based on the expected number of arriving patients of a certain type (Appendix A Table 15) we have calculated the probabilities that an arriving patient is of a certain type. Using these probabilities a patient set of any size can be constructed. For each patient, the patient type is known.

Recall from the hospital setting presented in Section 8.1 that a deterministic length of stay per patient is used. The required processing time per patient type is determined by calculating the expected duration of stay based the probability distribution in Appendix A Table 16 and 17.

Additional assumptions:

- Each patient, of every patient type, that is being processed is assumed to take up 1 day at the OT.
- It is assumed that the capacity during the weekend is similar to the capacity during weekdays.

8.3.1 Standard Scheduling Notation

Using the standard scheduling notation, which was introduced in Section 5.2.4, each scheduling problem can be classified. For clarity, we also wish to classify the setting which was introduced in this chapter. The notation differs per case size, as it includes the number of machines in each stage. For illustrative purposes, we will present the notation for the case depicted in Figure 8. In a setting that satisfies all assumptions made in this chapter, the standard scheduling setting is denoted as: $FH3, (P4^1, P7^2, P27^3)|r_j, block|\bar{C}$.

$P4^1$ indicates that stage 1 consists of 4 parallel machines
r_j indicates that job j cannot start processing before its arrival date
block implies that buffer capacity in between stages is limited. Jobs must wait in the previous stage until sufficient space in the next stage is released.
$\bar{C}$: the total completion time is the objective that's being considered

9 Case Study: Solving the HFSP

In Chapter 7 a MARL solution approach for solving a general HFSP was designed. In Chapter 8 we show that the hospital planning problem that we consider in this thesis can be modelled as a special case of the HFSP: a variant with three stages and in which a No-Wait constraint is imposed. In Chapter 10, we will study the capability of the MARL framework to solve this special case of the HFSP. However, before this special case is studied, it is valuable to investigate the performance of the MARL framework on the general HFSP, where the No-Wait constraint is disregarded. This is the main topic of this chapter. We will implement the MARL approach on a general HFSP of a bigger size and present results to evaluate the proposed solution method. It will be shown that we can achieve reasonably good results. Also, the main advantage of using a RL approach will be illustrated: the trained agents can instantly provide a good solution when faced with a new problem instance, based on past experience, without having to go through a learning or optimization phase.

The structure of this chapter is as follows. Below, we will first introduce the case which will be used throughout this chapter. There are 10 problem instances which we will use to train and evaluate the MARL approach. Section 9.1 introduces a benchmark method, which provides a benchmark against which we can compare the solutions found by the MARL approach. Then in Section 9.2 the implementation of MARL approach will be elaborately discussed. Deep Q-learning is used, for which the details are presented in section 9.2.1. As discussed in Section 6.4.1, the main benefit of this method is that it will allow the agents to generalize from experience, to situations it has never encountered before. Then in section 9.3 we present and evaluate two different training approaches. The first set of agents is trained only on 1 training sample. Secondly, we train another set of agents on 5 samples. Statistics are provided on the training for both these sets of agents. In section 9.4 we will dive into the performance of the solutions produced by the two agent sets, and compare them to the benchmark. More specifically, we will assess the generalization capability of the agent sets, by producing solutions for problem instances that were not used in the learning process. we will see that the agents which were trained on multiple training samples have better generalization capabilities. In fact, they perform just as well on test samples as on the training samples. Finally, Section 9.5 presents a summary of the results combined with a discussion of the MARL approach.

Throughout this Chapter a general three stage hybrid flow shop will be used. An overview of the case is presented in Figure 9. The characteristics are summarized in Table 11. It is coincidence that the number of jobs is equal to the total number of machines. Unlike the hospital setting, the no-wait constraint does not have to be satisfied in this case. The size of the considered case is determined such that a MILP can still obtain a feasible solution. This MILP solution is then used as a benchmark against which we can compare the solutions of the MARL approach. Data for the problem instances is generated based on the numbers from Section 8.3. Note that this is hospital data, and therefore in a real life situation the no-wait constraint should hold. However, in this chapter we ignore this for the purpose of illustrating the performance and the working of the MARL method on the general HFSP.

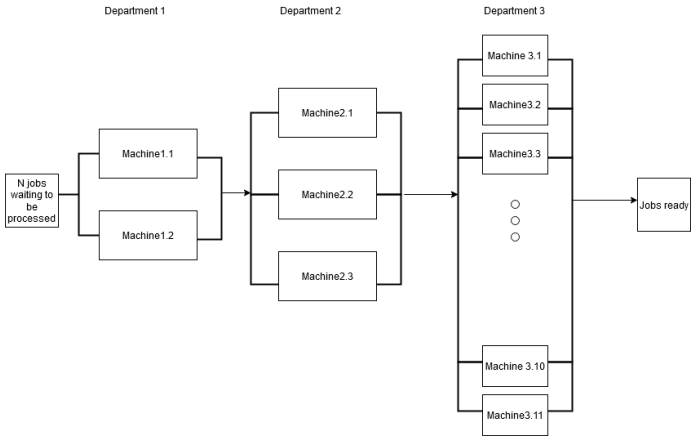


Figure 9: *This Figure provides an overview of the HFSP which is considered in this chapter.*

Department1	Department2	Department3	Types of jobs	# jobs / instance	Problem instances
2 machines	3 machines	11 machines	8	16	10

Table 11: *Summary of the characteristic of the case study considered in this chapter.*

9.1 Benchmark Method: Solving the HFSP MILP

We wish to obtain a benchmark against which we can evaluate the MARL learning approach. In Section 5.2.2 the MILP formulation for the general HFSP was provided. Within the data science firm Pipple, a generalized MILP solving module was developed. By making some minor adjustments, we can utilize this module to solve the MILP. In this module, the Python-MIP package version 1.7.3. is utilized. Additionally, the optimization solver Gurobi was used. The objective which is used here is, as mentioned multiple times now, the total completion time. Even though the problem setting is not particularly large, it was not possible to solve this problem within a reasonable amount of time till optimality. Hence, as a heuristic we choose to stop the optimization process after one hour. We then take the found feasible objective value for the total completion time as a benchmark. Applying this heuristic to each of the ten problem instances provides us with the total completion time values as in Table 12.

Total Completion Time	Sample1	Sample2	Sample3	Sample4	Sample5	Sample6	Sample7	Sample8	Sample9	Sample10
Benchmark value	200	183	185	181	203	180	171	172	172	187

Table 12: *This table contains the total completion time for each of the problem samples. These values are used as a benchmark. The total completion time is expressed in time units. This is the sum of all the individual completion times.*

What is interesting is that, even though this constraint is not enforced, there is no waiting time in between the stages for the patients. Intuitively, this makes sense, because, you wish to let patients

enter the system when you can finish them.

9.2 Implementation of the MARL Solution Approach for the HFSP

In this section we will discuss the implementation of the MARL framework for the three-stage HFSP case which was depicted by Figure 9 and summarized in Table 11 . The MARL approach that was introduced in the Methodology Chapter 7 will be applied. There is one learning agent associated with each machine. In this setting there are 16 machines thus 16 learning agents. Each agent will try to minimize his own individual expected total reward. In the case which we consider there are 16 jobs.

The objective which is used is again the minimization of the total completion time. The cost function which will be used to pursue this objective was introduced in Section 7.2.4. Now, in this setting the cost will be assigned based on a different moment. Instead of assigning the cost based upon s', we wish to assign the cost based on the release moment of a job. We will call this release moment $\tilde{s}$. In many cases, s' and $\tilde{s}$ coincide. However in cases where there is not yet a job in the queue when a job is released, $\tilde{s} \neq s'$. It only seems fair, that the cost the agent receives is based on the release state which is the direct result of him processing that job. Therefore the cost function is from now on denoted as $C(s, a, \tilde{s})$.

For a case of this size it is no longer efficient to use regular Q-learning. Furthermore, the goal is to learn the agents to generalize from experience. Hence after training the agent on training samples, they will be faced with new states. For this purpose, regular Q-learning is no longer applicable. Therefore, in this chapter the Deep Q-learning (DQL) algorithm will be applied. This DQL algorithm was discussed in Section 6.4.1. This implies that 16 different neural networks will be trained, one for each agent. Each FNN is trained such that it approximates the individual Q(s,a)-function.

9.2.1 Deep Q-learning Algorithm

To implement the MARL framework for this scheduling setting, the deep Q-learning (DQL) algorithm is used. This solution method was discussed and introduced in Section 6.4.1. In this section, we will describe how the DQL method will be implemented.

We utilize the by Mnih et al. (2015) proposed solution to reduce the correlation between the estimated Q-values and the Q-targets. Hence we don't update the Q-values at every step, but instead first a batch of observations is gathered. A batch size BS of 100 is utilized. Now, we let the agents iterate through the environment and gather the experience (s, a, s', r, a')-tuples for each agent. When we have collected the desired number of observations for each agent, the Q-targets are calculated. This is done for each experience tuple using Equation (27). Together with the (s,a)-pairs, the Q-targets constitute the desired training sample for the Neural Network which is used to estimate the $Q(s, a)$- function.

Hyperparameters

Table 13 presents all parameters needed for the set up of the neural network, which is used as a function approximator for the $Q^{agent}(s, a)$ function. Tuning of the hyperparameters for a NN structure is a difficult and tedious task. The optimization of the hyperparameters is outside the scope of this thesis. The values which have been utilized in this work are summarized in Table 13. These have been obtained in a trial and error fashion and have been selected based on their ability to validate the proposed solution approach in this work.

Additionally, Table 13 contains the parameters for the TD(0)-learning approach. Based on the literature review (Section 6.2.1 and 6.2.3) a constant learning rate of $\alpha = 0.1$ and a constant discount rate $\gamma = 0.9$ are chosen.

The pseudocode for the Multi-Agent Deep Q-learning algorithm used in this section can be found in Appendix E.

Neural Network Setup		Training Parameters		TD(0)-learning	
# Hidden Layers	3	Batch size	100	Discount rate γ	0.9
# Neurons	4	Epochs	250	Learning rate α	0.1
Activation Function for:					
Hidden Layer	tanh				
Output Layer	linear				

Table 13: *This table presents the hyperparameters for the neural network set up; the parameters for the training phase of the NN; and the parameters for the TD(0)-learning.*

Exploration/Exploitation Trade-Off
As an action selection method, the ϵ-greedy approach was used. With a linear ϵ decay. We start off with very exploratory agents: $\epsilon^{max} = 1$ and end with very greedily acting agents: $\epsilon^{min} = 0.1$.

Modeling choice: we choose to work with a joint epsilon. Meaning that all agents constantly explore and exploit with the same probability. Note that, in this algorithm we work with TD-learning. Hence we gather experiences for agents as we iterate through the environment. Some agents observe more states in one episode and thus gather experiences faster than other agents. We train the function approximator (the NN) after BS number of experiences have been gathered for an agent, this is what we call a batch fit. When using a linear epsilon decay, we decay the value of ϵ, and hence decrease/increase the probability of exploration/exploitation. This decay is performed when also the last agent has performed a new batch fit.

We will stop the learning process when a predefined number of fits **for each agent** has been performed. Using trial and error we have set this number of total required batch fits to 100. Note that this 100 batch fits is not the same as the batch size of 100. The batch size of 100 means that there are 100 observations within 1 batch fit. After every fit, ϵ is decayed by: $\frac{\epsilon^{max} - \epsilon^{min}}{100}$

Below we will shed some light on the training process. One set of agents is trained only on sample 1 whereas, another set of agents is trained on sample 1- sample 5. We will show the progress of the average total completion time and the average individual total reward per agent with respect to the number of batch fits. But first a benchmark method will be introduced against which we can compare the training results.

9.3 MARL Learning Statistics

Now that the benchmark has been established, we will look at the results of implementing the MARL approach. As mentioned, one set of agents is trained only on Sample 1. We will call this method 1 in the remainder of the text. Another set of agents is trained on Sample 1-5, which we will call method 5. Both method 1 and method 5, are trained using 100 training batches. This number has been set after a lot of trial and error. The training phase for both these methods takes around 25 minutes on a medium range laptop computer. After the training phase has been completed, both methods can provide a solution for a new problem instance < 1 second.

Let's compare the training progress of agents which are only being trained on sample 1 to agents which are trained on 5 samples. So on the one hand we have the agents which have only seen Sample 1, and on the other hand we have agents that are trained on Sample 1 - Sample 5. The total completion times for these 5 samples differ, which can be seen in Table 12. Therefore it does not make sense to show the full training evolution of these agents. For the sake of comparison, we thus only include statistics about the performance of the agents on sample 1.

Figure 12 displays the training progress of the average total completion time over the different batch fits.

What we see is that there are multiple local minima, however there is a clear downward trend visible. What can be seen is that the MARL system which is trained on this sample naturally performs better when its applied to this same sample than the MARL system which was trained on multiple samples. The average total completion time is averaged over all the sample runs in the batch.

Figure 10: *This Figure displays the average total reward for an agent in department for both Method 1 and Method 5 when trained using 100 batch fits.*

Note, that the total completion time is not directly optimized, the rewards are. Figure 10 shows the average individual rewards for an agent in department 1, 2 and 3 consecutively. The blue line represents the agents trained on Sample 1, and the orange line represents the agents trained on Sample 1-5. Again, for the latter, only statistics on Sample 1 are included. Note that method 5 is not inferior to method 1, even when applied to Sample 1.

The same downward sloping trend for the average rewards as for the average total completion time is visible. To show that 100 learning batches suffice, and the reward would not continue to decrease forever, Figure 11 shows the average reward for training the systems of agents using 500 batch fits. One can see that at the end of the training phase, the average rewards per machine fluctuate around the same level as in Figure 10.

What one does notice is that the training process of method 5 is more volatile than that of method 1. This can be explained by the fact that the agents encounter a more varying environment because they observe more samples. What one also sees is that there is more volatility in the lower batches. This is due to the exploratory behaviour of the agents in the batches in the beginning. The higher the batch number gets, the more the agents will choose their actions based on exploitation.

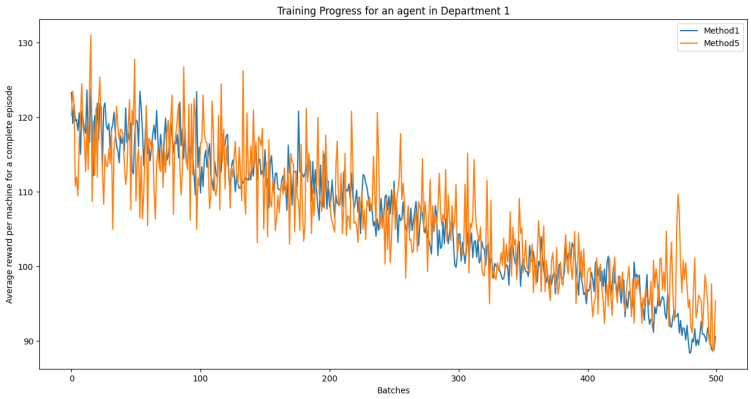


Figure 11: *This Figure displays the average total reward for an agent in department for both Method 1 and Method 5 when trained using 500 batch fits.*

To illustrate the influence of the learning rate on the training process, different values for the learning rate α and discount factor γ are compared in Figure 13. From these figures we can see that parameter choice for $\alpha = 0.1$ and $\gamma = 0.9$ are reasonable.

9.4 Solution Quality and Generalisation Capability

Now both of the above trained systems of agents are applied to all 10 problem instances. For each of these scheduling solutions the results are displayed in Figure 14.

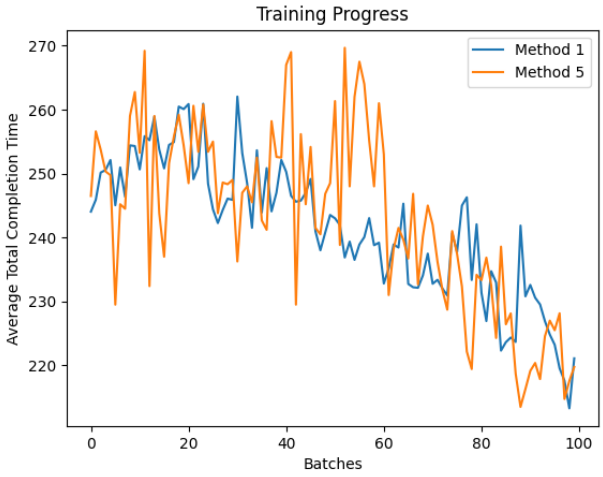


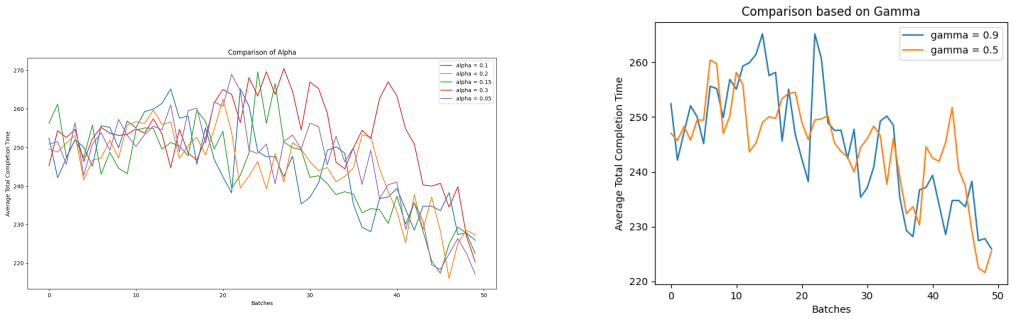
Figure 12: *This Figure displays the average total completion time for both Method 1 and Method 5 trained on 100 batches.*

The orange bars represent the MARL solution for method 1. Where the gray bars represent the MARL solution for method 5. As expected, method 1 outperforms method 5 when applied to sample 1. However, the solution of method 5 performs better for all other problem samples.

For both methods 1 and 5, the samples 6-10 are instances they have never seen before. By means of these test samples we will show and assess the generalisation capabilities of a MARL system. For the benchmark, as explained, it requires a run of one hour to obtain the reported solution. For the trained MARL solution however, a solution to these problem instances can be obtained instantly ($<$ 1 second). The error-percentage with respect to the benchmark in terms of total completion time is displayed in Figure 15. What we see is that method 5 which has been trained on multiple problem instances has better results when it is faced with the test samples. Note that for method 1, in fact all sample2-sample10 are test instances. The average test error for method 1 is: 22.6% over these 9 instances. Where the average test error for method 5 is 9.3%.

Hence from these findings and from Figure 14 and 15 we can conclude that a MARL system which has been trained on multiple training instances, has a better general performance and can generalize better from experience than one that has only been trained on a single sample.

What this also shows is that, the agents can generalize from experience to new solutions and perform just as well in unseen cases as in cases they have seen before when the training setting is diverse enough.



(a) Comparison based on α

(b) Comparison based on γ

Figure 13: *In this figure the average total completion time on sample 1 are displayed for different values of alpha. All other parameters are kept the same.*

Figure 14: *This Figure shows the total completion time for the benchmark, MARL solution trained on only 1 sample, and the MARL solution trained on the first five samples, applied to each problem instance.*

9.5 Summary and Discussion

The main finding this chapter lies in that fact that we showed that the MARL system possesses generalization capabilities. This entails the most important benefit of the MARL approach compared to the benchmark used here, and other solutions methods that are generally used. This means that, once it is trained, a MARL system can obtain a solution for a new instances, within a fraction of a second. In this case "new" means that the instances were not used during the training process. This feature is very promising, especially in situations where new solutions may be required fast, for example in the case of last-minute unexpected disruptions. To place this into perspective, consider that the benchmark method used in this section was programmed to run for an hour to produce a solution. If we would map this to, for example an hour of downtime in a factory full of expensive

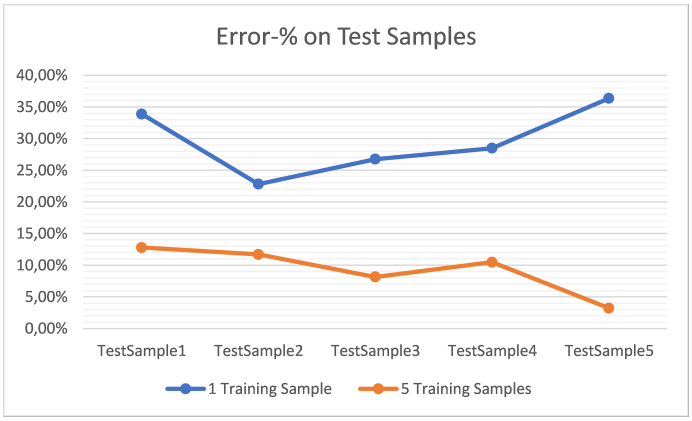


Figure 15: *This Figure summarizes the error percentage between the total completion time in the scheduling solution found by the learned agents and the benchmark solution. How this benchmark solution was obtained is discussed in Section 9.1.*

production machines, or an hour-long interruption of an OT, the benefits of this feature become obvious. From the findings in this chapter we can conclude that it is best to train a system of agent on multiple problem instances, which substantially improves the generalization capability. Therefor, this will be done in the remainder of this thesis.

In general, this generalization capability, combined with the fact that the obtained solutions were within reasonable range of the benchmark solutions, we conclude that the presented approach is a promising alternative to existing solution methods for the HFSP. However, further research is required. In the next subsection, we will discuss the limitations and directions for future research following from this Chapter.

9.6 Limitations and Suggestions for Future Research

These findings provide a good starting point for future research to further improve the MARL algorithm for HFSP. Below, the most prominent assumptions and that have been made, and their impact on the solution method will be discussed. Moreover, we will provide suggestions for future research, based on the findings from this chapter.

Firstly, this MARL approach can also be used and applied on large instances easily. The underlying Deep Reinforcement Learning technique has already shown to be very successful when applied to very large and complex problems, as we discussed in Chapter 6. It would be very interesting to see how this approach scales to larger problem instances. Moreover, it will be worthwhile to investigate how the capability to generalize scales, when to problem instances become larger.

Secondly, in the setting of this chapter, the assumption was made that the required processing time in department 1 for each job, regardless the patient type, is one time unit. The different type of patients thus only differ in terms of duration of stay at department 2 and 3. This makes that the

agents in department 1 do not have to take into account their own processing time when deciding on what patient type they start processing. In addition, the agents in department 2 and 3 do not make the choice which jobs enter the hospital flow. This decision is made by the agents in department 1. The joint goal is to have as few patients present in the queues of the system. Therefore, to reduce the total completion time of the schedule, agents in department 1 must make decisions, about what type of jobs can enter the system, that benefit the overall system. Still, because queues are allowed, agents in department 2 and 3 learn to make decisions on what type of jobs they can best process when in a specific state. Nevertheless, through this assumption cooperation between agents from department 1 and the consecutive departments is enforced. Note that even though this assumption might enforce cooperation between the agents, it does simplify the decision process. Hence, future work should reveal whether the proposed MARL solution approach still works when an extra layer of complexity is added in the form of different processing times in the first department of the system.

Another assumption we made, is that when there is only 1 job of a certain type in the queue, and 2 agents choose to process this job, then the second agent who chooses this job will remain idle. When an agent remains idle, by assumption, it remains idle until after the next time step. The reward that this agent receives is as-if he chose the non-idle action, when in fact he remained idle. We made this assumption to enforce cooperation between the agents. Future research should reveal whether it would be better to assign this cost to the remain idle action instead, to avoid underestimating the value of the associated non-idle action. Alternatively, the assumption that an agent remains idle until after the next time step could be relaxed. A possible approach would be to allow the agent to choose a new action after a predefined number of time-steps.

Another noteworthy aspect of the presented analysis, is that all processing times were deterministic. The proposed MARL framework can be easily extended to also deal with stochastic times. This can be done by providing a probability distribution per job type on the processing time to the agent. Then, when an agent starts processing a job of a certain type, the actual processing time is determined based on the probability distribution for a patient of this type. Additionally, this processing time may also be machine-dependent. In other words, the MARL framework can also be applied to a flow shop in which the machine within a department are not-identical. These are however different learning settings, in which agents might prefer to process patients of a certain type first. This will thus influence the learning process, and will therefore require separate hyperparameter tuning.

In the used solution approach, the action selection was done using the ϵ-greedy method. Future work should reveal whether a different action selection method would enable the agents to learn smarter, and whether better solutions can be found. Action selection methods which could be researched are the Soft-Max approach or the Exploring Selfish RL action selection method, which were introduced in 6.2.2 and in 6.3.3, consecutively.

Lastely, we note that the optimization and tuning of the hyperparameters of the TD(0)-learning and the neural network structure were outside the scope of this thesis. Optimizing these parameters using e.g. a grid search could improve the performance of the learning process. Moreover, we choose to terminate the learning process after a predefined number of learning batches. This number was set using a lot of trial and error. Figure 10 confirms that the we stopped the process at a good moment for the agents of Method 5. Instead of terminating after a predefined number of batch fits, one could also set a convergence factor.

10 Case Study: Solving the Hospital planning problem

In Chapter 7, a Multi Agent Reinforcement Learning framework for solving general Hybrid Flow Shop Problems was introduced. In Chapter 8 it was shown that the hospital OT scheduling flow can be described as a particular form of the HFSP. Chapter 9 demonstrated the potential of using the proposed MARL approach for solving a the general HFSP. Now what remains is to find out whether this MARL approach is also applicable to solve a hospital scheduling problem, which is the aim of this chapter. The main hurdle we aim to overcome is the addition of the No-Wait constraint in the Hospital planning formulation of the HFSP, compared to the general HFSP formulation. We will explore multiple approaches to enforce this constaint in our MARL solution approach.

This chapter will answer the research question: Is the proposed multi agent RL framework suited to solve the hospital scheduling problem? This question can be divided into to sub-questions: Firstly, we investigate whether the MARL framework from Chapter 7 be adjusted such that it can also be applied in the hospital case setting. Secondly, we assert whether we can use it to obtain desirable results.

The structure of the upcoming chapter is as follows: Section 10.1 will answer the question: Can the no-wait constraint be enforced by adjusting the cost function? Here, three different cost functions will be discussed and implemented, finally arriving at the point in which we are able to enforce the No-Wait constraint. Section 10.2 will then discuss the implementation details and the performance on the Hospital planning problem, using the setting and data introduced in Chapter 8. Finally, Section 10.4 will present limitations and directions for future research.

10.1 Implementation of the MARL Framework in a Hospital Setting

When implementing the MARL framework in Chapter 9, the No-Wait constraint was not taken into account. In fact, the state representation which was designed contains the queue for all departments. Of course, in a hospital flow it is not possible for patients to wait in between the departments (OT, IC and MC). Hence, if we wish to apply the MARL approach in a hospital setting, it must somehow be adjusted such that this constraint is enforced. In this section we will research in what way we can enforce this no-wait constraint by adjusting the cost function. For this purpose three different cost functions are studied, building up to a case in which the no-wait constraint can be enforced. As before, we work with a cost function. Hence the goal remains to minimize the overall cost.

In this section, for the training and test sets, the samples from Chapter 9 will be utilized.

10.1.1 Penalizing Based on Patients in the Queue

In this subsection we will explore a different feedback signal which could intuitively lead to compliance with the no-wait constraint. This feedback signal is composed of a high punishment for patients that have to wait for the IC or MC department.

$$C(s, \tilde{a}, \tilde{s}) = \sum_{i=1}^{PT} q_i^{OT} + 10 \sum_{i=1}^{PT} q_i^{IC} + 10 \sum_{i=1}^{PT} q_i^{MC} \tag{42}$$

Punishing the agents for patients that are waiting in the intermediate queues a lot, should lead to avoiding such situations. When implementing this cost function we do not observe the desired effect. The agents do learn to avoid situations in which patients are in the queue of the IC and the MC, however not in the way how we would like to see it. Instead of better adjusting the release moment to an available machine in the next department, the Q-values for agent 1 in department look as in Table 14. The Q-values for agent 2 in department 1 look similar. Hence, if the agents act greedy with respect to their gained knowledge, no patients are ever processed. Hence the agents will not start with treating a patient, with the result that no patients are treated at all.

Q(s,type1)	Q(s,type2)	Q(s,type3)	Q(s,type4)	Q(s,type5)	Q(s,type6)	Q(s,type7)	Q(s,type8)	Q(s,idle)
[198.51825]	[191.69643]	[192.20944]	[193.68312]	[197.39474]	[195.29367]	[201.31781]	[194.1238]	[181.92255]

Table 14: *This table presents the Q(s,a)-values for all possible actions for agent 1 in department 1. The Q-values for agent 2 look similar. State s was the starting state here. The corresponding actions, denoted by typeX, represent the action of starting to process a patient of type X. As one can see, if the agents are faced with this setting and if they act according to the experience they have gained, they will minimize over Q(s,a) and thus remain idle. Hence no patients are ever processed.*

A logical explanation for this result is that, as was described in Section 9.2, the cost is occurred at the moment when an agent releases the patient, $\tilde{s}$. If a patient is released by an agent, the patient logically appears in the queue for the next department. Since the feedback signal for the agent is based upon this $\tilde{s}$, the patient which he just released is in the queue and thus penalized as well.With this reward function. there is no distinction made between a desirable event, namely state $\tilde{s}$ in which this patient can be admitted to the next department right away, and a state $\tilde{s}$ in which the patient indeed has to wait in the queue. Hence. for each patient that is processed he incurs a big cost of 10 units, which is 10 times as big as when this patient would have remained in queue 1 and would not be processed. The agent does not just optimize his immediate reward but instead focuses on the expected return. However, using this reward structure, for each patient which is processed a big cost is incurred. So even if the agents would observe a episode which satisfied the constraint, the costs will add up, making that it is not beneficial to start processing.

Based on this analysis with this cost function, we have gained additional insight. In fact, what the no-wait constraint means is that no patient is released by a department if there is no bed available in the next department. Hence, we wish that the agents learn to adjust their behaviour to the availability of the agents in the subsequent department. Then, the cost function should also reflect this desired property. Therefore in the next subsection, a cost function will be designed which aims to adjust the release moment of an agent to the availability of an agent in the next department.

10.1.2 Adjusting the Release Moment to the Availability in the Subsequent Department

The cost function which we will discuss next is designed such that we better adjust the release moment of jobs to the availability of machines in the next department.

The cost function still depends on $\tilde{s}$, where $\tilde{s}$ is the state of the environment when the agent finishes and releases the patient. It is required that an agent in department k releases a patient only when another agent in department *k+1* is available. In this case, the agent in department *k+1* could immediately start processing this agent. In hospital terms: a patient should be released by the OT department, only when there is an IC bed available at the time the operation for this patient is finished. Hence, the reward function should resemble this.

To enforce that patients are released at a moment when there is a bed available at the next department, the cost function will be adjusted as explained below.

The new cost function at release moment $\tilde{s}$ consists out of three parts:

1. The number of patients waiting to be processed by the OT department *
2. The number of patients waiting to be processed in the queue for the IC department, reduced by the number of IC beds which are available at moment $\tilde{s}$
3. The number of patients waiting to be processed in the queue for the MC department, reduced by the number of MC beds which are available at moment $\tilde{s}$

$$C(s, \tilde{a}, \tilde{s}) = \sum_{i=1}^{PT} q_i^{OT} + \max\{0, \sum_{i=1}^{PT} q_i^{IC} - \sum_{l \in M^{IC}} \mathbb{1}_{RPT_l=0}\} + \max\{0, \sum_{i=1}^{PT} q_i^{MC} - \sum_{l \in M^{MC}} \mathbb{1}_{RPT_l=0}\} \quad (43)$$

** the number of patients in queue for the OT's is not reduced by the number of available OT rooms, because we want to stimulate that all patients are processed as fast as possible*

Note that the notation $\tilde{s}$ is used instead of s'. Recall s' is the next state in which the agent must make a decision. The moment that the agent releases a patient $\tilde{s}$ is not always a new decision moment for the agent. An example of a case where $\tilde{s} \neq s'$: an IC agent that releases a patient does this at a moment where there are no patients waiting in line for the IC department.

If there are no more patients that have to be processed by the system, the reward for an agent equals zero.

Now, implementing the cost function as in Equation (43) raises a new issue when there are no more patients in the queue for the OT and there are only a few patients left in the system. There is no incentive for the machines to process patients waiting in the queue for the IC and MC department as long as their number does not exceed the number of available machines. A situation equivalent to

Example 9 is in terms of reward equal to reaching the final state. Therefore, also this cost function does not lead to a situation which we really wish to see accomplished. What happened while using this cost function, was that the system reaches a state like in Example 9 after which it never leaves it.

Example 9 ***System gets stuck*** *$|M^{OT}| = 2$, $|M^{IC}| = 3$, $|M^{MC}| = 11$; hence in the extreme situation where the queues in the system looks as follows: $q^{OT} = (0,0,0,0,0,0,0,0)$, $q^{IC} = (1,1,1,0,0,0,0,0)$, $q^{MC} = (1,1,1,1,1,2,2,2)$, and all beds in the system are not occupied, i.e. the remaining processing times are $RPT^{IC} = (0,0,0)$ and $RPT^{MC} = (0,0,0,0,0,0,0,0,0,0,0)$. Then the $C(s,\tilde{a},\tilde{s})$ associated to this state are 0. Which is equivalent to reaching the final state, in which there are no more patients left in the system and in the queue.*

Moreover, this method is also not able to find solutions which satisfy the no-wait constraint. Figure 16 provides statistics on how often a training episode satisfies the constraint. Which is practically never. The cost function under consideration in Equation (43) is thus not yet sufficient to enforce the no-wait constraint.

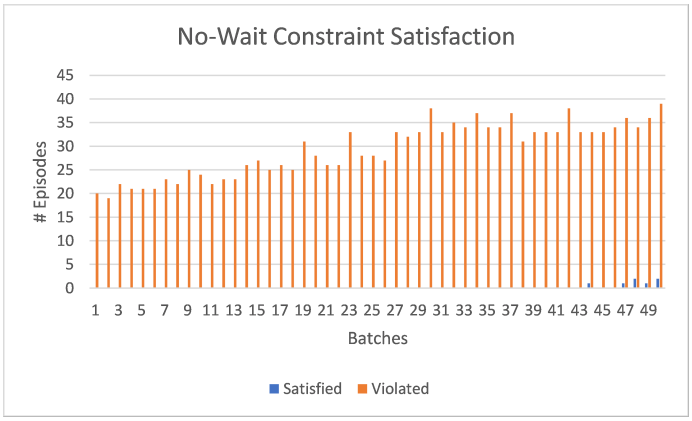


Figure 16: *This Figure presents statistics on the number of episodes that satisfy and violate the no wait constraint in a training episode per batch fit.*

Again, we gained additional insight. From Figure 16 we can conclude that the agents spend too much of their training time exploring wrong parts of the environment, since a solution which violates this constraint is no feasible option anyway. If we wish to see the constraint satisfied, instead of using an implicit way to learn this to the agents, perhaps we should move to a more explicit way. A possible method would be to, in case the constraint is violated, assign a very large negative reward signal and terminate the learning episode. In this way we wish to stimulate that the agents explore the parts of the environment which are feasible. We will explore this idea in the following subsection.

10.1.3 Focus on Exploring Feasible Solutions

Like we saw, using the cost structure presented in the previous subsection, the agents explore too much of the infeasible region and too little of the feasible one. Hence in this subsection we will unroll a more rigorous approach. If the no-wait constraint is violated, we assign a very large penalty to the agents who have contributed to reaching this state. Equation (44) displays this. The large penalty is set to 150 here. In addition to this large punishment, the episode is terminated as soon as the constraint is violated. The penalty is assigned to all agents that have contributed to this violation. This implies that also agents that are still processing a job at the moment of termination, receive the fine. In this fashion we wish to let the agents explore more of the feasible region of the environment.

$$C(s,\tilde{a},\tilde{s}) = \begin{cases} \sum_{i=1}^{PT} q_i^{OT} & \text{if} \max\{0, \sum_{i=1}^{PT} q_i^{IC} - \sum_{l \in M^{IC}} \mathbb{1}_{RPT_l=0}\} + \max\{0, \sum_{i=1}^{PT} q_i^{MC} - \sum_{l \in M^{MC}} \mathbb{1}_{RPT_l=0} = 0 \\ 150 & \text{Otherwise} \end{cases} \tag{44}$$

By means of these adjustments, we wish to enforce the no-wait constraint. This no-wait constraint naturally implies that the beds in the IC and MC department cannot remain idle if there is a patient waiting in the queue. Hence the action to remain idle is no longer a valid action for agents in department 2 and 3. Therefore it is removed from the action set.

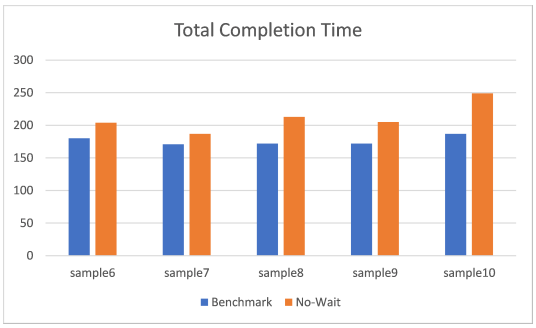
Note, that the agents in department 1 are responsible for admitting the right patients to the system. Hence the OT agents are also to blame for the violations of the no-wait requirement in department 3. However, here it is difficult to attribute this to 1 specific agent in department 1. It is also difficult to punish them in retrospect. Hence we choose to punish the agents in department 2 for a violation in department 3. Because the agents in department 1 take into account the expected future return, this punishment for the agents in department 2 will propagate back to the agents in department 1.

Note that a negative feedback signal is in fact a positive numerical signal because the goal of the agents is to minimize the expected return.

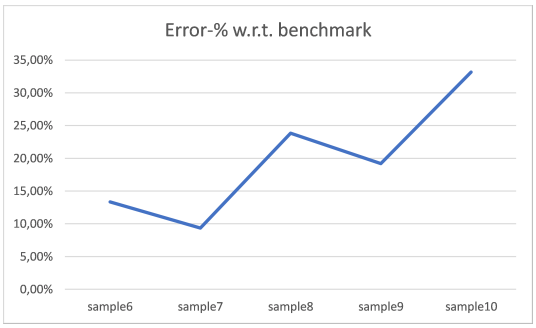
10.2 Implementation and Results

Now we will apply the new cost structure from the previous section using Equation (44). From Section 9.4 we learnt that a system of agents which has been trained on multiple training samples can generalize better from experience. Hence, the agents here are also trained on Sample 1- Sample 5.

Figure 17 summarizes the results when this solution method is applied to test Sample 6-10. When implemented and applied to the test cases, we are able to successfully constitute a schedule which satisfies the no-wait criterion. The total completion time has an average error of 19.8%with respect to the benchmark solution.



(a) Total Completion Time comparison



(b) Error percentage w.r.t. benchmark

Figure 17: *In this figure the results from agents who have been trained to satisfy the no-wait constraint using the above mentioned method are presented. The results are compared against the benchmark method from Section 9.2.*

Note that in the solutions of the benchmark method in Table 12, the no-wait property was satisfied. Hence these same numbers are used to evaluate the results in this section.

These results are within a 20% margin of the benchmark, which is significant worse than the setting in which the no-wait constraint is not present. An explanation for this can be derived from the statistics in Figure 18. This figure presents statistics on the number of fully observed episodes that satisfy the no wait constraint during the training phase. From this we can see that although the number of episodes in which the constraint is satisfied is somewhat higher than before (Figure 16). However, this number is still not very large. This small number of no-wait episodes per batch can be explained. Whenever the no-wait constraint is violated, the episode is terminated. The observations up until the moment the constraint is violated are still stored in the batch and taken into account for the learning process.

Also there is no real increasing trend over the number of batch fits. This is probably partly due to the action selection mechanism of the agents. In this setting greedy-ϵ action selection method is used. Meaning that whenever an agent explores, he does so with equal probability over all actions. Therefore there is still a lot of time spent learning on constraint violating episodes. So instead of learning how to behave in feasible episodes, the agent spent most time on learning how to behave based on infeasible episodes. This should be resolved such that the agents can learn better how to behave in case of feasible parts of the environment. Making use of a smarter exploration method could go a long way. Another possible explanation for the bad exploration capabilities of the agent might be the global exploration issue which can be a challenge due to the MARL nature. Coordinated exploration could resolve this issue 6.3.3.

10.3 Summary and Discussion

In this chapter we have investigated the performance of the introduced MARL framework on the special case of the HFSP corresponding to the hospital planning problem. The main difference with

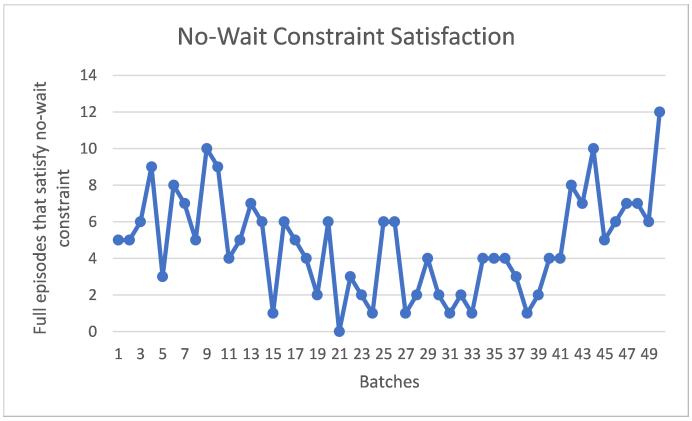


Figure 18: *This Figure presents statistics on the number of episodes that satisfy the no wait constraint in an episode per batch fit. This is for the case where whenever the constraint is violated, the episode is terminated.*

this problem, compared to the problem considered in Chapter 9 is the addition of the No-Wait constraint. In order to enforce this additional constraint in the solution produced by the MARL, we investigated several alterations to the learning process of the agents. All of these methods focus on adjusting the cost function to stimulate the agents to make decision that lead to solutions that satisfy the No-Wait constraint. The three notions that were explored are:

- Heavily penalizing the existence of queues in the IC and MC department
- Rewarding proper adjustment of the release moment to the availability in subsequent department
- Directly penalizing decisions leading to schedules that violate the No-Wait constraint

In addition, the latter cost-function adjustment was combined with a policy in which the learning episode was immediately terminated when a non-feasible decision was made by an agent. In the latter case, experiments showed the effectiveness in enforcing the No-Wait constraint. Therefore, this method was used in the remainder of the chapter.

After applying the MARL framework with the adjusted cost function and episode-termination policy to the case described in Chapter 8, we arrived at similar conclusions as in Section 9.5. Using the MARL, we were able to provide solutions that performed reasonably well, when compared to the benchmark solution method. Moreover, we again showed the generalization capabilities of the MARL framework. This leads to the conclusion that also for the hospital planning specific HFSP, the MARL is a promising solution method, and future research would be appropriate. The remainder of this Chapter is concerned with providing insights in the limitation of the methodology used in this chapter, and providing the most promising directions for future research.

10.4 Limitations and Suggestions for Future Research

In this subsection we present the limitations of the analysis performed in this chapter. Moreover, we will present suggestions for future research that focus on solving the hospital planning problem using the presented MARL framework.

Similar as in Chapter 9, the assumption of similar processing times in department 1 simplifies the decision process. The framework is built such that altering this assumption can be done easily. Namely simply by adjusting the required processing time. However, this will affect the learning process of the agents. Hence for this new setting, tuning of the used action selection mechanism, as well as the optimization of hyperparameters and the effect on the learning process should be studied. Future work should reveal whether the proposed MARL solution approach still works when an extra layer of complexity is added in the form of different processing times in the first department (OT) of the system.

Moreover, future research could be done on Constrained Policy Optimization, which makes use of a Constrained Markov Decision Process to enforce constraints (Achiam, Held, Tamar, & Abbeel, 2017). This is especially convenient and designed to be used for systems with learning robots that physically interact with or around humans. In these settings, a particular behaviour could cause damage or be harmful to a plant or to humans working around it. This approach might prove useful in enforcing the No-Wait constraint.

As was mentioned in 10.1.3, the ϵ- greedy action selection strategy prevents the agent from exploring the feasible region of the environment. By feasible region we mean the part of the environment where the No-Wait constraint is satisfied. With this in mind, the potential increase in performance from an alternative, smarter action selection method, like e.g. Soft-Max, is big. A smart action selection strategy combined with terminating an episode whenever the No-Wait constraint is violated, will allow agents to avoid the infeasible region of the environment. By doing so, the region where the no-wait constraint is satisfied can be further explored. Thereby, probably leading to more knowledge about how to produce good feasible solutions. An an alternative action selection method, joint exploration strategies, like the so called Exploring Selfish Reinforcement Learning, should be considered.

Lastly, we note that only deterministic processing times have been taken into account. In a real life situation, the processing times of a patient are of course not deterministic and not known beforehand. Therefore in future work, we would wish to incorporate stochastic processing times.

11 Conclusion

In this chapter we will answer the research questions and present the most prominent findings of this thesis. There were two main research questions which were presented in the Introduction.

In section 11.1 we reflect on the research question: "Can we design a MARL algorithm to solve the general HFSP?". In Section 11.2 we reflect on the question: "Can the proposed MARL algorithm be adjusted such that it can be utilized to solve a hospital scheduling problem?" Based on the answers to these research questions a final conclusion is drawn. Section 11.3 concludes this work with topics for future research.

11.1 MARL for the HFSP

In this thesis a Multi-Agent Reinforcement Learning framework was designed for a general Hybrid Flow Show Problem. The main idea behind the designed MARL algorithm is that every machine in the flow shop is represented by a self-learning agent, where each agent determines autonomously what job from the corresponding queue he will start processing. The goal of the algorithm is that the agents learn how to make decisions in the short term which will eventually lead to a good final schedule. The framework was introduced in detail in Chapter 7.

We have applied the MARL algorithm in a case study on illustrative cases. In this case study, we show that a system of various agents can learn how to make individual decisions which lead to a joint scheduling solution. One of the benefits of a MARL framework over regularly used techniques is that once the agents have learned how to make decisions based on a given set of training problem instances, it can instantly provide a solution to a new problem instance. In this case, "new" refers to instances of the problem that were not provided to the agents to learn from. This capability is especially useful in situations where new solutions may be required fast, for example in the case of last-minute unexpected disruptions. For comparable solution methods, it is not uncommon that it takes in the range of several tens of minutes to several hours to find a reasonable solution. If we imagine that in such situations, valuable resources operate inefficiently or don't operate at all, the added value of this feature of the MARL framework becomes obvious.

In the conducted analysis, the advantages mentioned above are confirmed. What we do see is that the solution is improved when the MARL framework is trained on multiple training instances instead of on one instance. The performance then deviates about 10% from a benchmark solution, which was obtained using a state of the art MILP solver. For more detailed results we refer the reader to Chapter 9. A detailed summary is provided in Section 9.5.

Combining these results, we conclude that that provided MARL framework is indeed able to provide reasonable good solutions to the studied cases of the HFSP, answering the first research question in a positive way. Moreover, taking into account the extensively discussed generalization capabilities, we judge that future research is worthwhile.

11.2 MARL for a Hospital Scheduling Problem

In Chapter 8 the flow of patients in a hospital was described as Three-Stage No-Wait hybrid flow shop. In Chapter 10 we have researched whether the proposed general MARL algorithm was an appropriate solution approach for solving this specific case of the HFSP. The main difficulty that we had to overcome to apply the general MARL to the hospital setting was the No-Wait constraint, which is imposed in the latter problem. We have shown that this constraint can be enforced through the cost function and by adjusting the learning process of the agents. The performance of the MARL algorithm compared to the benchmark was relatively worse than in the general HFSP setting. However, given the novelty of this approach, we argue that the results are still reasonable, compared to the benchmark. As a benchmark, again a state-of-the-art commercial solver was used. For more detailed results we refer the reader to Chapter 10. A detailed summary is provided in Section 10.3.

Based on these results we can conclude that the MARL algorithm has potential to be a good solution method for the No-Wait HFSP, when applied to the hospital planning problem. This also answers our second research question positively. Again, the generalization property has potential benefits in situations where a new schedule is needed rapidly, e.g. in the case where emergency patients disrupt an existing schedule.

There are still a lot of points for improvement and future research is required before this algorithm could actually be applied in practice. We will elaborate upon these points of improvement in the following section, thereby concluding this thesis.

11.3 Recommendations for Future Research

Both Chapter 9 and 10 concluded with a discussion and suggested topics for future research, focused on the specific problem studies in the chapters respectively. In this new and promising field, there are infinitely many options for improving and continuing this work. In this section we limit ourselves to the main drawbacks of the approach studied in this work.

- Implement a different action selection method, such that the agents explore the right parts of the environment. Potentially, using a joint exploration policy, which can ensure cooperated exploration, can lead to big improvements in results. Alternatively, e.g. Leniency might result in a better coordinated learning process.
- Include different processing times in department 1 such that an extra layer of complexity is added to the decision process of the agents in department 1.
- Implement stochastic processing times. This addition does not require big adjustments in the framework. However, the learning process should be evaluated and the action selection method, the stopping criterion and the hyperparameters used should be studied.
- Optimize the hyperparameters which are used in the MARL algorithm, including the parameters for the neural network set-up.

- Implement the MARL algorithm for a bigger problem instance to show that, compared to other solutions, the MARL methods perform relatively good in big environments.
- In hindsight, the formulation in this MARL might not be the best formulation for a hospital problem. Because agents/beds in the IC and MC department are identical and must in fact directly start with processing a patient when one arises. There is thus not a real decision to make for these agents, nor have they a real influence on the learning setting. Therefore with regards to the hospital case study which was performed in this study, the formulation as a hybrid flow shop of the hospital flow can still remain. However, learning agents in the first department would probably have sufficed. This requires some adjustments in the proposed MARL framework which alters and simplifies the learning process.

A Data Tables

Patient Group	Planned Arrivals (# of patients)	OT Duration (hours)
1	8	4
2	10	8
3	67	4
4	13	8
5	3	4
6	2	8
7	1	8
8	7	2

Table 15: *This table depicts data about the 4 week number of patients to be operated and the OT duration per patient group.*

	Probability of length of stay (days) in IC										
Patient Group	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.07	0.87	0.02	0.02	0.02	0	0	0	0	0	0
2	0	.9	0.08	0.02	0	0	0	0	0	0	0
3	0.01	0.83	0.11	0.03	0.01	0.01	0	0	0	0	0
4	0	0.83	0.1	0.04	0	0.01	0.01	0	0	0	0.01
5	0	0.79	0.07	0.07	0	0	0	0	0.14	0	0
6	0	0	0.14	0.44	0.14	0.14	0	0	0.14	0	0
7	0	0	0	0	0	0	0	1	0	0	0
8	0.79	0.21	0	0	0	0	0	0	0	0	0

Table 16: *This table depicts the probabilities for the length of stay at the IC per patient group.*

	Probability of length of post-OT stay (days) in MC											
Patient Group	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**>10**
1	0.74	0	0	0	0.02	0.1	0.07	0.05	0.02	0	0	0
2	0.83	0	0	0	0	0	0.04	0.04	0.02	0.02	0	0.05
3	0	0.01	0.01	0.04	0.32	0.24	0.12	0.09	0.05	0.03	0.04	0.05
4	0.03	0	0	0.01	0.12	0.16	0.18	0.15	0.10	0.04	0.04	0.17
5	0	0	0	0	0.07	0.07	0.07	0.2	0	0.2	0.2	0.19
6	0	0	0	0	0	0	0	0.14	0	0	0.14	0.72
7	0	0	0	0	0	0	0	0	0	0	1	0
8	0.21	0.3	0.08	0.15	0.13	0.05	0	0.05	0	0.03	0	0

Table 17: *This table depicts the probabilities for the post-OT length of stay at the MC per patient group.*

Patient Group	Example Procedure
(1) Child simple	Closure ventricular septal defect
(2) Child complex	Arterial switch
(3) Adult, short OT, short IC	Coronary bypass
(4) Adult, long OT, short IC	Mitral valve plasty
(5) Adult, short OT, middle IC	Coronary bypass with expected medium IC stay
(6) Adult, long OT, middle IC	Heart transplant
(7) Adult, long OT, long IC	Thoraco-abdominal aneurysm
(8) Adult, very short OT, no IC	Cervical mediastinoscopy

Table 18: *This table provides information about the different groups of patients and provides an example procedure (Dellaert & Jeunet, 2010).*

B Regular Q-Learning Algorithm

In this section the pseudocode for the regular Q-learning Algorithm is presented.

Algorithm 1 Regular Q-learning Algorithm

Initiate Q-Table Q(s,a) with only zeroes, $\forall s \in S, a \in A$
Set discount factor γ
for each episode i **do**
    for each $s \in S$ **do**
        for each $a \in A$ **do**
            Take action a in s, and observe s' and $R(s, a, s')$
            Calculate $Q^{new}(s, a)$
        end for
    end for
    Update $Q^i(s, a) = Q^{new}(s, a)$
    Update $V^i(s) = \max_{a \in A} Q(s, a)$
end for

C Patient Planning 1 learning agent

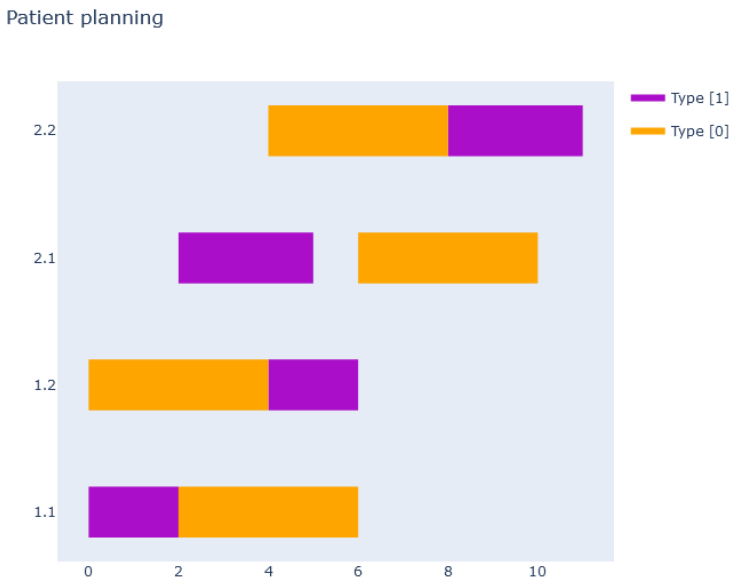


Figure 19: *Illustration of the planning which is the result of the setting in Section 7.2.7. In this solution approach, machine 1.1 has learnt his optimal behaviour using the Q-learning Algorithm. Machines 1.2 - 2.2 all act in a deterministic way according to the predefined rules in Section 7.2.6.*

D Q-Table for 1 learning agent under uncertain state transitions

Iteration 1	Q-Values			
States	(1,0)	(0,1)	(0,0)	Value
(2,2,0,0,0,0,0,0)	5.19	5.41	7.14	5.19
(1,1,0,1,0,2,0,0)	1.99	3.96	3.96	1.99
⋮	⋮	⋮	⋮	⋮
(1,1,1,1,0,0,0,0)	2.16	2.22	3.45	2.16
(1,2,1,0,0,0,0,0)	3.30	3.94	4.48	3.30

Table 19: *This Table depicts the Q(s,a) values for agent n_1 after 52 episodes, in an environment where the actions of machine 1.2, 2.1 and 2.2 are determined randomly. $s = (1, 1, 1, 1, 0, 0, 0, 0)$, is an example of a state for which the $Q(s, a_1)$ and $Q(s, a_2)$ are very close to each other. Depending on the observation in the next iteration, the optimal policy for the agent can change from a_1 to a_2. However, the expected return for the agent will more or less stay the same and be of the same order as it is know. So even though the policy does not completely converge, the behaviour of the agent does not improve significantly. This is due to the constant learning rate, which will leave the agent to continue to move in the direction of the latest observation forever, if not stopped.*

E Pseudocode for Multi-Agent Deep Q-Learning algorithm

Algorithm 2 Multi-Agent Deep Q-Learning algorithm

Set hyper parameters for NN: # hidden layers, # of neurons, activation function of hidden and output layer
Set discount factor γ, and learning rate α
Set ϵ, ϵ-decay rate and # of required batch fits
Set lowest number of fits over all agents: θ=min$_{n \in N}$ # batch fits for n=0
for each agent $n \in N$ **do**
    Initialize Memory $\mathbb{M}^n$ with capacity BS
    Initialize NN with random weights for predicting Q_0
    Initialize Temporary Memory $\mathbb{TM}^n$
end for
while $\theta <$ required batch fits **do**
    Initialize $s = s^{start}$
    while $s \neq s^{final}$ **do**
        for All $n \in AN(s)$ in s **do**
            Choose action $a_n \in A(s)$ based on policy π
        end for
        Combine all actions to get joint action $\tilde{a}$
        Perform $\tilde{a}$ and get next state s' and cost $c(s')$
        for Every agent $n \in AN(s), s' = \tilde{s}$ **do**
            Add experience to memory: Store $(s, a_n, c, s', A(s'))$ in $\mathbb{M}^n$
        end for
        for Every agent $n \in AN(s), s' \neq \tilde{s}$ **do**
            Store s, a_n in $\mathbb{TM}^n$
        end for
        for Every agent $n \notin AN(s), s' = \tilde{s}$ **do**
            for $n \in AN(s')$ **do**
                then we have a complete experience tuple
                remove (s, a_n) from temporary memory $\mathbb{TM}^n$
                Add $(s, a_n, c(s'), s', A(s'))$ in $\mathbb{M}^n$
            end for
            for $n \notin AN(s')$ **do**
                $s' = \tilde{s}$ is no decision moment for this patient yet
                Add $c(\tilde{s})$ to (s, a_n) stored in $\mathbb{TM}^n$
            end for
        end for
        for $n \notin AN(s), n \in AN(s'), s' \neq \tilde{s}$, $\mathbb{TM}^n \neq \emptyset$ **do**
            Remove $(s, a_n, c(\tilde{s})$ from $\mathbb{TM}^n$
            Store $(s, a_n, c(\tilde{s}), s', A(s'))$ in $\mathbb{M}^n$
        end for
        Update $s = s'$
        for Every $|\mathbb{M}^n| = BS$ **do**
            Calculate Q-Targets $Q_{k+1}(s, a)$ using Equation (27)
            for each $(s, a_n, c(\tilde{s}), s', A(s')) \in \mathbb{M}^n$
            Train network for agent n
            using training batch of size BS: $((s, a), Q_{k+1})$
            Clear Memory $\mathbb{M}^n$
            (# batch fits for agent n) +=1
        end for
        for min$_n$ (# batch fits for agent n) $> \theta$ **do**
            θ +=1
            ϵ -= ϵ decay
        end for
    end while
end while

References

Achiam, J., Held, D., Tamar, A., & Abbeel, P. (2017). Constrained policy optimization. *arXiv preprint arXiv:1705.10528*.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*(5), 834–846.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., ... others (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Cardoen, B., Demeulemeester, E., & Beliën, J. (2010). Operating room planning and scheduling: A literature review. *European journal of operational research*, *201*(3), 921–932.

Da Silva, F. L., & Costa, A. H. R. (2019). A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, *64*, 645–703.

Dellaert, N., & Jeunet, J. (2010). Hospital admission planning to optimize major resources utilization under uncertainty.

Emmons, H., & Vairaktarakis, G. (2013). The hybrid flow shop. In *Flow shop scheduling* (pp. 161–187). Springer.

Fei, H., Meskens, N., & Chu, C. (2010). A planning and scheduling problem for an operating theatre using an open scheduling strategy. *Computers & Industrial Engineering*, *58*(2), 221–230.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 32).

François-Lavet, V., Fonteneau, R., & Ernst, D. (2015). How to discount deep reinforcement learning: Towards new dynamic strategies. *arXiv preprint arXiv:1512.02011*.

Gabel, T., & Riedmiller, M. (2007). On a successful application of multi-agent reinforcement learning to operations research benchmarks. In *2007 ieee international symposium on approximate dynamic programming and reinforcement learning* (pp. 68–75).

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics* (Vol. 5, pp. 287–326). Elsevier.

Grinstead, C. M., & Snell, J. L. (2012). *Introduction to probability*. American Mathematical Soc.

Han, W., Guo, F., & Su, X. (2019). A reinforcement learning method for a hybrid flow-shop scheduling problem. *Algorithms*, *12*(11), 222.

Hernandez-Leal, P., Kaisers, M., Baarslag, T., & de Cote, E. M. (2017). A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*.

Jiménez, Y. M. (2012). A generic multi-agent reinforcement learning approach for scheduling problems. *PhD, Vrije Universiteit Brussel*, 128.

Juliani, A. (2016). *Simple reinforcement learning with tensorflow part 7: Action-selection strategies for exploration.* Retrieved from `https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-7-action-`

Kouridi, C. (2020). *A brief summary of challenges in multi-agent rl.* Retrieved from `https://christinakouridi.blog/2020/01/02/challenges-multiagent-rl/`

Lai, M. (2015). Giraffe: Using deep reinforcement learning to play chess. *arXiv preprint arXiv:1509.01549*.

Lamiri, M., Grimaud, F., & Xie, X. (2009). Optimization methods for a stochastic surgery planning problem. *International Journal of Production Economics*, *120*(2), 400–410.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994* (pp. 157–163). Elsevier.

Littman, M. L. (2001). Value-function reinforcement learning in markov games. *Cognitive systems research*, *2*(1), 55–66.

Macario, A., Vitez, T., Dunn, B., & McDonald, T. (1995). Where are the costs in perioperative care?: Analysis of hospital costs and charges for inpatient surgical care. *Anesthesiology: The Journal of the American Society of Anesthesiologists*, *83*(6), 1138–1144.

Melo, F. S. (2001). Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, 1–4.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Nguyen, T. T., Nguyen, N. D., & Nahavandi, S. (2020). Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*.

Nowé, A., Vrancx, P., & De Hauwere, Y.-M. (2012). Game theory and multi-agent reinforcement learning. In *Reinforcement learning* (pp. 441–470). Springer.

openAI. (2018). *Key papers in deep rl.* Retrieved from `https://spinningup.openai.com/en/latest/spinningup/keypapers.html`

Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, *11*(3), 387–434.

Qu, S., Wang, J., Govil, S., & Leckie, J. O. (2016). Optimized adaptive scheduling of a manufacturing process system with multi-skill workforce and multiple machine types: An ontology-based,

multi-agent reinforcement learning approach. *Procedia CIRP*, *57*, 55–60.

Reijnierse, H. (2013). Stochastic operations research models. *Course Notes. Tilburg University, The Netherlands*.

Ruiz, R., & Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European journal of operational research*, *205*(1), 1–18.

Samudra, M., Van Riet, C., Demeulemeester, E., Cardoen, B., Vansteenkiste, N., & Rademakers, F. E. (2016). Scheduling operating rooms: achievements, challenges and pitfalls. *Journal of Scheduling*, *19*(5), 493–525.

Silver, D. (2015). *Ucl course on rl.* Retrieved from `https://www.davidsilver.uk/teaching/`

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... others (2016). Mastering the game of go with deep neural networks and tree search. *nature*, *529*(7587), 484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... others (2017). Mastering the game of go without human knowledge. *nature*, *550*(7676), 354–359.

Souki, M. (2011). Operating theatre scheduling with fuzzy durations. *Journal of Applied Operational Research*, *3*(3), 177–191.

Stanciu, A., Vargas, L., & May, J. (2010). A revenue management approach for managing operating room capacity. In *Proceedings of the 2010 winter simulation conference* (pp. 2444–2454).

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

van Essen, J. T., Hans, E. W., Hurink, J. L., & Oversberg, A. (2012). Minimizing the waiting time for emergency surgery. *Operations Research for Health Care*, *1*(2-3), 34–44.

van Ham, L. (2019). *The application of deep reinforcement learning to the hybrid flow-shop scheduling problem.*

Van Riet, C., & Demeulemeester, E. (2015). Trade-offs in operating room planning for electives and emergencies: A review. *Operations Research for Health Care*, *7*, 52–69.

Vera, J. (2019-2020). *Course notes, decision making with business analytics.* Tilburg University.

Verbeeck, K., Nowé, A., & Tuyls, K. (2005). Coordinated exploration in multi-agent reinforcement learning: an application to load-balancing. In *Proceedings of the fourth international joint conference on autonomous agents and multiagent systems* (pp. 1105–1106).

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., ... others (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, *575*(7782), 350–354.

Visser, M. (2020). *Grootste knelpunt voor ziekenhuis in coronatijd blijft het tekort aan personeel.* Retrieved from `https://www.trouw.nl/binnenland/grootste-knelpunt-voor-ziekenhuis-in-coronatijd-blijft-het-teko`

Wang, Y., Tang, J., Pan, Z., & Yan, C. (2015). Particle swarm optimization-based planning and scheduling for a laminar-flow operating room with downstream resources. *Soft Computing*, *19*(10), 2913–2926.

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., & Kyek, A. (2018). Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, *72*(1), 1264–1269.

Watkins, C. J. C. H. (1989). Learning from delayed rewards.

Wiegand, R. P. (2003). *An analysis of cooperative coevolutionary algorithms* (Unpublished doctoral dissertation). Citeseer.

Zemzem, W., & Tagina, M. (2018). Cooperative multi-agent systems using distributed reinforcement learning techniques. *Procedia Computer Science*, *126*, 517–526.

Zhang, K., Yang, Z., & Başar, T. (2019). Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*.

Zhou, P., & Shen, H. (2011). Multi-agent cooperation by reinforcement learning with teammate modeling and reward allotment. In *2011 eighth international conference on fuzzy systems and knowledge discovery (fskd)* (Vol. 2, pp. 1316–1319).

Zhu, S., Fan, W., Yang, S., Pei, J., & Pardalos, P. M. (2019). Operating room planning and surgical case scheduling: a review of literature. *Journal of Combinatorial Optimization*, *37*(3), 757–805.