Order Acceptance under Uncertainty: a Reinforcement Learning Approach

Beta

Research School for Operations Management and Logistics

This thesis is number D-85 of the thesis series of the Beta Research School for Operations Management and Logistics. The Beta Research School is a joint effort of the departments of Technology Management, and Mathematics and Computer Science at the Technische Universiteit Eindhoven and the Centre for Telematics and Information Technology at the University of Twente. Beta is the largest research centre in the Netherlands in the field of operations management in technology-intensive environments. The mission of Beta is to carry out fundamental and applied research on the analysis, design and control of operational processes.

Dissertation committee

Printed by Gildeprint Drukkerijen, Enschede, The Netherlands
© M.Mainegra Hing, Enschede, 2006

ISBN 90-365-2405-9

ORDER ACCEPTANCE UNDER UNCERTAINTY: A REINFORCEMENT LEARNING APPROACH

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. W.H.M. Zijm,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op donderdag 7 september 2006 om 13.15 uur

door

Marisela Mainegra Hing
geboren op 15 april 1972
te Santiago de Cuba, Cuba

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. A. van Harten

en de assistent-promotor:

Dr. P. C. Schuur

Acknowledgements

Many people have contributed to this thesis in different ways. I thank Aart van Harten and Peter Schuur very much for giving me the posibility to work in this project, for the nice discussions we had on several topics that helped me to finish this thesis. Special thanks go to the team from computer science: Martijn van Otterlo, Marko Snoek and Mannes Pool; for introducing me on the field of Reinforcement Learning, for the many discussions we had of all kind and for the yearly team-lunch we had. Thanks to the BETA school for organizing the PhD activitiess. Thanks to Mark Ebben, Erwin Hans and Floris Olde Weghuis for the insights in order acceptance and for facilitating me their integrated/generic simulation environment. Thanks to Jorge Luis for his help with the implementation of the ANNs to work with in eM-Plant.

Thanks to Prof. J.W.M. Bertrand, Prof. C. Hoede, Dr. M. Poel, Prof. J. van Hillegersberg, Prof. J.H.A. de Smit and Prof. J. Telgen for participating in my graduation committee.

This thesis has taken many years to come to an end, and I could not have gone through this period without the support of all the many wonderful people I have met here and there, at both sides of the ocean.

From one side. First, my Dutch non-dutch family: Adriana and Andrei. My colleagues from the OMST/OMPL department: Matthieu, Erwin, Mark, Elke, Gerhard, Luitzen, Leo, Hans, Marco, Noud and Martijn. A big and international group of friends, Annetty, Isora, Dorina, Israel, Zenith, Angel, Bas, Guillermo, Corina, Alina, Artiles, Ariadna, Valia, Albita, Marko, Martijn, Matthieu, Kees, Lukman, Bert, Daniel, Samuil, Zlatko, and Elena.

From the other side. First my parents and brother, who have always been with me in a way, for all their support and their lifes as models of eternal study. My colleagues from the Mathematic Department at the Universidad Central de Las Villas in Santa Clara, Cuba. Thanks to my friends Julito, Kerstin, Maripili, Annieyis, Noelys and Jorge Luis.

Enschede, August 2006
Marisela Mainegra Hing

Contents

Chapter 1

Introduction

Order Acceptance (OA) is one of the main functions in a business control framework. Basically, OA involves for each incoming order a 0/1 (i.e., reject / accept) decision. Traditionally this problem is solved as follows: always accept an order if sufficient capacity is available. However, always accepting an order even when capacity is available could disable the system to accept more profitable orders in the future. This unpleasantness is measured by means of the concept of opportunity cost, namely: the highest-valued alternative that one is losing. Here we shall focus on decision rules that take opportunity costs into account.

An important aspect for OA is the availability of information to the decision maker. Generally in the literature information regarding negotiation with the customer such as an estimate of the work content of an order, a norm for the necessary processing time, the price and the due-date are assumed to be known or estimated, and a model of the production process is also considered to be known beforehand. However it could be difficult to obtain such information. Uncertainty in OA has received little attention in dynamic models. Applying dynamic models with uncertainty could be very expensive in computation time. Also there may be different degrees of uncertainty, e.g., with respect to order arrivals, processing times, machine break downs, etc. Moreover, one may not know the parameters of the stochastic dynamics.

In this study we consider order acceptance under uncertainty taking into account opportunity costs. For this purpose we use a stochastic modeling approach using Markov decision theory and learning methods from Artificial Intelligence. Markov decision theory is known to be useful modeling decision making under uncertainty. Learning methods from Artificial Intelligence have been used to deal with incomplete information and to cope with complex problems with implicit information. Reinforcement Learning is a promising approach that combines useful modeling and solution ideas. There are some aspects that make Reinforcement Learning (RL) appealing for our problem. The idea of learning without the necessity of complete model information and the possibility of learning even from delayed rewards allows us to consider different degrees of uncertainty and to take into account the opportunity cost problem in a natural way. Although RL has successful applications in various areas, these applications are not always fully understood at a theoretical level. It means that convergence properties of the corresponding algorithms and procedures for tuning the parameters in the algorithms have to be explored. Hence, much work still has to be done in order to understand how RL can best be applied to a problem and to get insight into why some domains are considerably more tractable for RL than others.

Summarizing, in this thesis we shall deal with the following research questions:

1. How can we model OA under uncertainty taking into account opportunity costs?
2. How to tune the parameters for RL?
3. How does RL perform compared to other heuristics for OA?
4. How can we interpret the knowledge learned by using the RL approach in OA under uncertainty?

This chapter is organized as follows. In Section 1.1 we present a characterization of the Order Acceptance problem that is used consequently in the models of this thesis. In Section 1.2 we present the main ideas of this thesis about the models and the approach to solve them. In Section 1.3 we review the literature on Order Acceptance and on Reinforcement Learning. In Section 1.4 we present an example of the OA problem as studied in this thesis. We conclude this chapter with an outline of the thesis in Section 1.5.

1.1 Order Acceptance under uncertainty

Order Acceptance (OA) is one of the main functions in a business control framework at tactical planning level. Order acceptance is a typical decision problem at the interface of customer relations management (CRM) and production management (PM). The rejection of an order may have repercussions for future customer relations. For an arriving order the implications of acceptance for production must be investigated, especially in terms of availability of production capacity. This leads to a certain estimate for the delivery date for the order, which is communicated to the customer. Comparison with the customer due-date demands may lead to agreement, perhaps after some iterative negotiations. The result of this due-date setting process may have its effect on the customer relations as well. In case of acceptance, the next stage is the actual realization of the contractual order at an operational level. The resulting earliness/tardiness of the actual delivery date is again a main element in terms of customer relation management. Altogether, it is clear that CRM / PM coordination is the essence of order acceptance.

Order acceptance situations can be characterized with respect to

(i) flexibility in capacity resources

(ii) customer flexibility in order definition

(iii) predictability of order arrivals and order attribute parameters

(iv) effects of order loss

(v) uncertainty in order characteristics for production

(vi) policy constraints.

In practice the situations encountered for order acceptance may vary with respect to each of the characteristics mentioned above. Flexibility in resources refers to capacity extension using overwork, hiring temporary flex labour staff or it addresses the issue of subcontracting parts of the order. The market may be such that order definitions of quantity, price and delivery date are essentially nonnegotiable, especially when competition is fierce; or they may be negotiable up to some degree. Predictability of orders may be stochastic due to customer consumption and/or replenishment patterns (say periodic), procurement contracts or logistical advantages (full truck or container loads). Order losses can lead to changes in arrival rates of orders and/or ordered volumes. These losses can be modelled with penalizations including estimates of future losses. Uncertainty in order characteristics is strongly dependent on the type of production environment. Compare a make-to-stock environment with an engineering-to-order environment. In the first case when an order comes in, the realization phase to attend that order is already finished. In the latter case, while discussing an order for acceptance, part of the details for the realization phase may still be unknown, such as, e.g., product routings and capacity requirements. Order acceptance policy constraints refer to a priori assumptions on the control structure. A policy constraint could be that under the circumstances that capacity is available and the due-date can be met, an order is always accepted. Such policy corresponds to myopic management behavior (Wester et al., 1992). Always accepting an order when capacity is available may bring the system in a bad situation for accepting more profitable orders in the future. It is an interesting facet of the order acceptance policies to take opportunity costs into account. How to find a good trade-off between long-term opportunity costs and immediate yield in case of order acceptance under uncertainty is a central problem in this thesis. It complicates the order acceptance decision considerably.

Another important aspect of order acceptance decision making is the availability of information to the decision maker. It is evident that immediate information regarding negotiation with the customer such as an estimate of the work content of an order, a norm for the necessary processing time, the price and the due-date is available. However, as for information on the state of the production facility several possibilities arise. They range from an estimate of the overall utilization rate to a detailed estimate of the evolution in time of the capacity profile for various workstations up to a planning horizon. Moreover the ways in which uncertainty in this state information is dealt with may vary. Possibilities are that in some way slack is introduced or that at each time the state evolution is probabilistic. An analogous remark holds for the information on future orders. For estimates of opportunity costs some information on the arrival processes of the orders is necessary. If not available yet there is the possibility to activate a learning process to overcome the incompleteness of the information. In practice such sort of information can be obtained by using adaptive statistical procedures or other learning techniques. Altogether this problem area constitutes a new and interesting field for several lines of research. In this thesis we focus on the possible contributions of operations research and learning techniques for the development of decision support tools for order acceptance, from a management perspective, as described above.

1.2 Modeling Order Acceptance and learning

Our modeling approach is in line with the tradition of discrete time Markov Decision Processes (MDP) in operations research. As usual in Markov processes the system dynamics is described in terms of transition probabilities between a set of possible states. At every decision moment we should decide what action a is to be taken in the current state s. The effect of an action a is expressed in terms of the transition probabilities from state s to s', i.e., it is modeled as $p(s, a, s')$. What one is looking for is an optimal decision making policy. Such a policy prescribes an action a for each state s. Optimal refers to some cost or profit criterion which is usually taken as the total expected value of all future yields. A nice optimization property for the so-called value function is available, and this gives rise to efficient algorithms to compute an optimal decision making policy, such as value iteration, policy iteration or reformulation as a Linear Programming problem, see (Puterman, 1994), (Winston, 1994) and (Bertsekas & Tsitsiklis, 1996).

This is a very flexible way of modeling order acceptance in the given context. The state definition combines information on the new arrivals of orders and their attributes with the characteristics of the capacity profile of the production, as mentioned in (i), (iv). The transition probabilities follow from the order arrival process in (ii), the production progress which includes new information due to better knowledge on the detailed design of orders, and the effect of decision making. In order to model the effect of acceptance of orders on production a priority rule for the adaptation of the capacity profile has to be assumed, say in agreement with a first come first serve or earliest due-date principle. In a general setting also the effect of the decision making on capacity flexibility as in (v) can be shown in $p(s, a, s')$. Various sorts of assumptions on profits of accepted orders, as well as costs of order rejection can be accommodated in the total expected value criterion.

Nevertheless the OR approach as sketched has some shortcomings. Firstly, though this sort of modeling is very flexible, it assumes a priori knowledge on many parameters and if this information, say on $p(s, a, s')$, is incomplete, the method has to be adapted. This can be done in a two phases procedure where in the first step the necessary information on parameters is gathered with a stochastic decision policy and next, once the information on parameters is sufficiently accurate, the optimization of the decision policy takes place. Secondly, the Markov decision approach suffers from what is usually referred to as the curse of dimensionality. In a somewhat complex business environment the size of the state space for order acceptance problems may easily be astronomically large. So even when complete information is at hand there is a need for the introduction of approximations. Therefore it also becomes an issue that the learning technique is compatible with effective approximation strategies.

For both these reasons we explore in our research a new alternative for effective process control strategies in order acceptance problems by using computational intelligent techniques, particularly Reinforcement Learning (RL), also known as Neurodynamic Programming (NP). These techniques have been proven to be successful in distilling information from large amounts of data. The distilled information can be used to learn about the opportunity costs or incomplete information and to optimize the process control under study. Reinforcement Learning is a rather new approach that can be interpreted as a conjunction between learning machine problems (automatic goal learning) and Markov decision models (decision making problems). RL focuses on an agent (a virtual decision maker) with a defined goal (optimization criterion) who through trial and error in interaction with its environment (in our case the CRM/PM interface) tries to learn an optimal behavior. This means decision making such that the criterion is optimized in the long run. Once the agent is properly trained it constitutes a decision support tool for the order acceptance management. There are some aspects that make RL appealing to our problem. The idea of learning without the necessity of complete model information and the possibility that the agent learns even from delayed rewards (when the effects of an action can be known only in the future) allow us to consider different degrees of uncertainty and to take into account the opportunity cost problem in a natural way. Moreover, RL combined with the potentialities of neural networks has been claimed to overcome the curse of dimensionality as it appears in many complex problems of planning, optimal decision making, and intelligent control, also in our problem setting. It is worthwhile to investigate this claim.

We compare decision policies found by an RL trained agent with heuristics. It will turn out that an RL trained agent usually outperforms simple greedy heuristics, showing that such agents learned to deal with opportunity costs in a satisfactory way. An obvious advantage of heuristics is the more appealing insight in their structure and performance. As for more advanced heuristics, it can be expected that once suitable structures with sufficiently many parameters are introduced they should perform also very well if properly tuned. Such heuristics are not easily determined in complex situations. An interesting idea is to use an RL trained agent to detect good advanced heuristics, meanwhile interpreting the agent's performance. Our goal is to show that this can indeed be done in certain cases using data mining techniques. Furthermore, we emphasize that RL trained agents are a more flexible and robust approach with respect to incomplete information than tuning parameters in a heuristic to fit a specific situation.

Summarizing, the new elements for order acceptance in this thesis concern:

- introducing some new Markov decision models for OA,
- exploring how to deal with at the onset incomplete information,
- establishing how to use RL for OA,
- evaluating RL as a tool for OA decisions, so as to gain insight and find generalizations,
- obtaining structured heuristics for OA.

For reasons of transparency we focus our research on order acceptance problems with the following structure. We consider orders that can be classified according to a finite number of classes and the production capacity is modeled both as a single server and as a job-shop. In the real world more complexity may arise. Nevertheless the analysis of the models in this thesis provides new insight into the problem of Order Acceptance under uncertainty and the question of how to deal with opportunity costs.

1.3 Literature review

Relatively little attention has been paid to the order acceptance problem in the literature. Nevertheless, some relevant literature can be found in the area of operations management concerning the CRM/PM interface, in the area of Operations Research (OR), in the area of accounting and in the area of Reinforcement Learning (RL).

In the first area some studies have been done about the degree of information required to deal with the coordination mechanism between the order acceptance function and the scheduling function. Three policies were compared by Wester, Wijngaard and Zijm (Wester et al., 1992) in a single machine make-to-order environment. (1) The monolithic policy accepts orders based on a detailed schedule which is built upon an order arrival. (2) The hierarchical and (3) myopic policies take their decisions based on the total workload of all accepted orders. The hierarchical policy makes a detailed schedule with the accepted orders, whereas the myopic policy uses some simpler dispatching rules. The experimental results show that in situations with large setup times and tight due-dates the monolithic approach performs better, probably due to a phenomenon of implicit selective acceptance. Moreover there are no real differences in performance between the hierarchical and myopic policies. The hierarchical and integrated approaches were also compared by Ten Kate (ten Kate, 1995) in process flow industries. He explains why due to uncertainty and complexity of production, hierarchical structures are more widely used. The experimental results show that only for tight situations (short lead-times, high utilization rate) the integrated approach outperforms the hierarchical approach although still in such situations the performance is often bad for both approaches. The problem of accepting orders together with capacity loading decisions is studied in multipurpose batch process industries by Raaymakers (Raaymakers, 1999). Besides the traditional workload and scheduling policies she also considers a makespan estimation policy which uses some aggregate information about the current job mix and the total workload. From the empirical results she obtained that the scheduling policy always realizes a better or equally good service level and capacity utilization performance than the other two policies, however it is very time consuming. Furthermore, the makespan estimation policy realizes higher capacity utilization than the workload policy, particularly in situations with high demand and product variety, so it is a good alternative when detailed information can be difficult to obtain, or when computation time is scarce.

Lead-time flexibility is studied together with order acceptance and scheduling in (Charnsirisakskul et al., 2004). Simultaneous order acceptance and scheduling decisions should be taken in a single-machine deterministic production system where each customer order has a preferred and latest acceptable due-date. Tardiness with respect to the latest due-date is not allowed, and there are tardiness penalties for orders that are not completed before the preferred due-date. The problem is modelled as a mixed integer linear program. The numerical experiments give insight into the benefits in different demand and production environments of three types of flexibility: lead-time, partial fulfillment, and inventory flexibility. The results rank the three types of flexibility as inventory, lead-time, and partial fulfillment in decreasing order of their usefulness.

Uncertainty in order acceptance has started to receive attention just recently. Ivanescu (Ivanescu, 2004) continued Raaymakers' work but under uncertainty, including stochastic order arrivals and processing times. She also considers estimation policies which use aggregate information and compares them to the scheduling policy, now with a fixed slack to account for the stochastic processing times. First with an $\alpha-$ regression policy (makespan estimation) she aims to obtain an α service level (the probability of on-time order set completion). The scheduling policy outperforms the regression policy in scenarios with low order mix variety (higher capacity utilization and closer to the target service level). However, in scenarios with high order mix variety the regression policy gets closer to the target service level than the scheduling policy. Therefore in a hybrid policy she combines detailed scheduling and regression models for the slack estimation. This hybrid policy keeps the benefits of the scheduling policy on capacity utilization but obtains delivery performance closer to the target.

In (Ebben et al., 2005) the authors compared four workload based policies varying from rules based on aggregate information to detailed scheduling methods in a make-to-order job shop with stochastic processing times. To account for inaccuracy in methods and uncertainty, a safety factor is explored with the capacity utilization. (1) The Aggregate Resource Loading (ARL) method looks at aggregate information: an order is accepted when the required capacity is not greater than the total available capacity within the time window for that order. (2) In the Resource Loading per Resource (RLR) method, an order is accepted when for each job of the order capacity is available on the corresponding resource within the time window of the job. (3) The Earliest due-date based order acceptance (EDD) uses the EDD dispatching rule to construct a detailed schedule every time an order arrives. Capacity assignment to time periods is not fixed since orders are rescheduled upon acceptance. (4) The Branch and Price Resource Loading (BPRL) method uses a truncated version of an exact approach for solving the preemptive resource loading problem. Their results show that the sophisticated approaches significantly outperform the straightforward approaches in case of little slack.

Altogether this line of work is oriented mainly towards due-date performance and capacity utilization, not so much towards costs and profits (except (Charnsirisakskul et al., 2004)), certainly not towards opportunity costs.

In the operations research literature the idea of opportunity costs in order acceptance problems is recognized, but not worked out in as full generality as we propose. This problem arises naturally in the context of reservation systems for car rentals, room reservations in hotels or tank capacity rentals. Usually such problems are discussed under interval scheduling (Pinedo & Chao, 1999), but the problem is then deterministic. Garbe (Garbe, 1996) made an interesting deterministic description of some online[1] order acceptance algorithms. Although some algorithms are given, the results are mainly theoretical and the theorems are concerned with a worst case analysis and for a restricted set of problems.

[1] An online algorithm has to decide whether or not to accept an arriving order immediately upon its start time without any further information about possible successors.

An important step in considering opportunity costs is through the use of dynamic programming models. A single server system in continuous time in which opportunity costs play a role is studied by Nawijn (Nawijn, 1985). A decision has to be made between starting a new service for an arriving order or rejecting an arriving order depending on its expected processing time. Orders that arrive while the server is busy, are lost. He proves that there exists an optimal control policy that maximizes the expected average number of customers per unit of time. For the case of Poisson arrivals he obtains an explicit formulation of the optimal policy which in general might be difficult to obtain. An early survey on the application of MDP for control in queueing networks is presented in (Stidham & Weber, 1993). In this survey some references are presented that obtain the structure of optimal policies for admission control and routing in simple networks. (Wijngaard & Miltenburg, 1997) studies the use of short term capacity flexibility for incidental sales opportunities in a single server. Uncertainty is considered on the arrivals of the sales opportunities. A dynamic programming model allows to take into account the uncertainty and the opportunity costs. The optimal policy (maximizing average reward) is partially obtained, a sales opportunity is accepted if its reward per unit of capacity is greater than the average reward per unit of capacity. Although it is not an operational criterion since the average reward is not known, and it is difficult to estimate, it shows that the acceptance of a specific sales opportunity only depends on its reward per unit of capacity. A more recent work is (Brouns & van der Wal, 2000) which incorporates as a new feature termination control in an $M/E_N/1$ queue (Poisson arrivals and Erlang service times). Termination control consists of deciding whether to quit processing jobs in service. The objective is to maximize the expected discounted profit over a time horizon. Under certain regularity conditions on the shape of the reward function, they obtain that there exist optimal threshold policies, e.g. "quit serving if there is too much work waiting and if the job under service has already passed a sufficient number of phases", so there are still unknown parameters to be determined in order to have the exact optimal policies.

In the area of management accounting, opportunity costs related to order acceptance have received some more attention since the eighties, though it has been recognized that opportunity costs are hard to determine (Miller & Buckman, 1987; Gietzmann & Ostaszewski, 1996). In (Gietzmann & Monahan, 1996) an MDP model of a simple stochastic manufacturing process (single server, two products) is modelled in order to assess two heuristic costing rules: direct and absorption costing based acceptance rules. The direct costing (DC) rule accepts an order if its contribution margin exceeds the expected holding costs associated with the order. This rule ignores opportunity costs. The absorption costing (AC) rule seeks to account for opportunity costs and accepts an order if its contribution margin exceeds the sum of its expected holding costs and the allocated costs of providing the capacity. The optimal policy is only partially obtained for some states, this allows to study the behavior of the two heuristics and to reach the conclusion that none of these two heuristic rules is better than the other for all parameters of the problem. The MDP model allows to obtain an "open acceptance" condition on the parameters of the problem under which the AC rule is better than the DC rule. Opportunity costs are analyzed as part of relevant costs for order acceptance decisions by Wouters in (Wouters, 1997). Relevant costs for order decisions are considered to be those avoidable by rejecting an order (incremental costs and opportunity costs). General guidelines on the necessary information from production planning and control function are given to calculate relevant costs. However alternatives to consider opportunity costs are only related to planned capacity, not to future situations. Wouters recognizes the difficulties to calculate relevant costs under uncertainty and gives some suggestions to assess the reliability of the information under uncertainty.

In this thesis we consider a modeling approach for the OA problem under uncertainty. We investigate more general problem settings. In particular we consider more general order attributes and service capacity and different degrees of uncertainty. Generally in the literature a model of the production process is considered to be known beforehand. The problem is that in practice most of the time such a model is hard to find. Sometimes even an approximate model of the process is not easily obtained due to incomplete information. We consider Reinforcement Learning, an approach from Artificial Intelligence, as a remedy.

Artificial Intelligence tools have been widely used in several applications, but hardly for OA problems. However, Wang (Wang, 1994) proposed a multicriteria OA decision tool in which the OA decision rule is based on a prioritization given by a pairwise comparison using a neural network based preference model. Orders are accepted following the priority ranking if capacity is available, but opportunity costs are not an explicit issue there.

Reinforcement Learning (RL) is a simulation based approach which basically consists of agent learning to achieve goals while interacting with the environment. RL has its roots in the 1950s, in the early works on Artificial Intelligence (specifically learning by trial and error), and in the field of Optimal Control (specifically dynamic programming), see (Sutton & Barto, 1998) and (Bertsekas & Tsitsiklis, 1996). These two lines came together during the 1980s to produce the modern field of RL.

RL has been successfully applied in various domains. It has been used in playing games: checkers player (Samuel, 1959), (Samuel, 1967), backgammon player (Tesauro, 1994), in some logistic problems like for example: elevator control (Crites & Barto, 1996), job-shop scheduling (Zhang & Dietterich, 1995), (Riedmiller & Riedmiller, 1999), routing (Boyan & Littman, 1994) and resource allocation (Singh & Bertsekas, 1997) in communication networks. In all these problems the dynamic model was known beforehand, but the size of the problems made them intractable for traditional dynamic programming methods.

The system behavior is simulated, and its performance is improved by means of iterative reinforcement.

RL has also been used in admission control and routing in communication networks (Carlstrom, 2000). Admission control is the known name for order acceptance problems in queueing systems (Puterman, 1994). Admission control in communication networks is to accept or reject an arriving call request. Routing is the set of actions taken to set up a call between two nodes along some path through a network (Carlstrom, 2000). Only unicast routing is considered (one sender, one receiver). For each arriving call if the set of feasible routes is empty the call is rejected. Otherwise one of the routes should be chosen or the call could be rejected in order to reserve resources (e.g. bandwidth) for more profitable future calls. The results present several algorithms based on RL which improve the way a network is shared compared to other conventional routing and admission control algorithms.

The study of order acceptance policies in a production or service environment using RL started recently. Snoek studied a neuro-genetic architecture using RL that aims to optimize OA and scheduling policies in a job-shop environment, (Snoek, 2000). This approach outperformed two simple heuristic policies. In a different study (Mainegra-Hing et al., 2001), an RL policy is shown to converge to the optimal policy for a simple OA case with a single server with at most one job in execution, and it is further discussed here in Chapter 3.

In this thesis we also consider more general cases with several independent order types that compete for capacity on a shared resource or in a multiresource environment. Hence the scarce capacity of the resources should be allocated to a set of concurrent orders planned over a given planning horizon. Further, we allow for stochastic perturbations during job execution resulting in a higher or lower capacity realization than anticipated. Finally we consider Order Acceptance together with other kinds of decisions in an integrated planning approach. The idea is to show how RL may also be useful in a more general and realistic decision making process. Here we consider three possible cases. The first one includes a due-date and price negotiation with the customer, the second one considers choosing a routing in a multiresource shop, and the last one considers outsourcing decisions that may partially outsource an order in service.

It is not very difficult to define reasonably simple heuristics for OA problems (Pinedo & Chao, 1999; Raaymakers, 1999; Mainegra-Hing et al., 2001; Ebben et al., 2005). In this thesis a more general setting for a class of heuristics is presented, which allows for a wealth of advanced heuristics taking system state characteristics into account. This approach to more general heuristics for OA is not addressed in literature to the best of our knowledge.

1.4 An example

To give a first impression of the underlying ideas of this thesis we present here an example. Consider a single resource manufacturing system with one type of resource where arrivals of orders take place as events in continuous time. This kind of problem is thoroughly explained in Chapter 4.

In terms of the characterization (i)-(vi) given above we assume the following:

(i) Capacity is considered as a unique resource. Once accepted, orders are loaded over a planning horizon of 6 stages ($H = 6$). There is a maximum regular capacity of 4 units at each stage of the planning horizon ($C_{max} = 4$). We assume that for each order type the due-time of delivery is at most 6, the size of the planning horizon. The due-time of an order is relative to the stages in the planning horizon and the arrival moment of the order. For example suppose each stage spans one hour and in a situation like in Figure 1.1 at stage 1 of the planning horizon we are at time 7.00 hours and there is an order in the system with the due-date 11.00 which arrived before 7.00 then its due-time is 4, which indicates that the order should be planned not after stage 4 of the planning horizon.

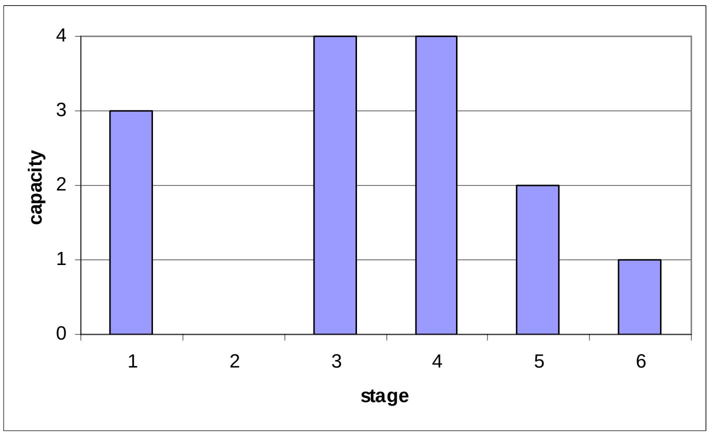


Figure 1.1: *Capacity utilization in a planning horizon with* 6 *stages, 4 hours each*

Here we consider a capacity perturbation that only occurs at the first stage of the planning horizon. This perturbation may be caused by an order being finished earlier or later than scheduled or by a machine problem. We consider in this example a discrete uniform distribution that takes values $(-1, 0, 1)$. When the maximum regular capacity is being used at the first stage of the planning horizon, and a capacity perturbation occurs that implies using more capacity, it is possible to use an extra unit of non-regular capacity at a cost of 2 ($\eta = 2$).

(ii)-(iii) Order definitions are based on ten different types of orders where type i has a due-time t_i, processing time w_i and it generates an immediate reward upon acceptance r_i. Arrivals of orders take place according to a Poisson process, and the probability of arrival is order type dependent, type i has an arrival rate λ_i, see Table 1.1.

(iv) Rejection of an order does not affect any future arrivals.

(v) Order processing requires an order type dependent processing time w_i from the single resource, the actual realization of an order may differ from this value causing a possible capacity perturbation as explained in (i).

(vi) Although arrivals of orders take place continuously, they are only evaluated at discrete equidistant time moments, so we consider several arrivals in each discrete time unit (batch arrivals). At every discrete point in time the decision maker should choose a set of orders from the arrival batch, if there is no available capacity for an order, the only option is rejection for that order. The order acceptance decision is supported by loading procedures. Specifically, to allocate capacity here we use backward loading and least shift back procedures, see Algorithms 4.1 and 4.2 in Chapter 4.

The goal is to decide on the acceptance of arrived orders in order to maximize the total expected average reward per unit of time.

The data of the ten types of orders are provided in Table 1.1. The table also shows a column with the reward per processing time r/w. Violation of due-times is not allowed.

type of order	t	w	r	r/w	λ	
1	6	4	4	1	3	
2	2	7	7	1	2	
3	3	4	4	1	1	$H = 6$
4	6	12	24	2	2	
5	6	10	20	2	3	$C_{max} = 4$
6	4	4	8	2	1	
7	3	10	30	3	1/6	$\eta = 2$
8	2	4	16	4	0.1	
9	1	3	15	5	0.1	
10	1	2	12	6	0.05	

Table 1.1: *Data for an example with 10 different types of orders in a single resource with multiple arrivals*

The case could be that the first six types of orders, which pay off a smaller reward per unit of capacity upon acceptance ($1 \leq r/w \leq 2$), are from the usual costumers, and they are also the more frequent jobs. Suppose a market study shows that there are four more profitable potential clients ($3 \leq r/w \leq 6$) but with smaller arrival frequency. The question could be which rule to follow in order to take advantage of the opportunities that the more profitable costumers could offer. If these more profitable costumers would arrive more often one can think of rejecting some of the usual costumers.

A simulation study of this problem shows different average rewards for different rules similar to some rules previously discussed in Section 1.3. First we study the scenario with the six more frequent type of orders and then with all the ten types. We consider heuristics similar to the direct and the absorption costing rules studied by Gietzmann and Monhanan (Gietzmann & Monahan, 1996) as mentioned in Section 1.3. Similar to the direct costing rule we consider a greedy rule that accepts all orders as long as capacity allows it. Similar to the absorption costing rule we consider rules that accept orders only if their reward per required capacity is over a certain threshold, we call these rules *OrderQuality(b)* where b is the threshold value. Note that this new rule takes opportunity costs into account: even if capacity is available an order with reward per required capacity under b is rejected.

Table 1.2 shows some results of this study. *O6Greedy* and *O10Greedy* are the greedy rules for the problem with six orders and the problem with the ten orders respectively. *O6OrderQuality(2)* and *O10OrderQuality(2)* were the best of the *OrderQuality* rules.

scenario	Average reward
O6greedy	4.082
O6OrderQuality(2)	7.97
O10greedy	4.29
O10OrderQuality(2)	8.1

Table 1.2: *Performance of some heuristics for the example case*

The results show that the rules considering opportunity cost in both problems lead to better results: rules *O6OrderQuality(2)* and *O10OrderQuality(2)* that reject orders with reward per required capacity of one unit (types of orders 1, 2 and 3). Also the problem with 10 types of jobs achieves much better results than the problem with only six.

One question now is whether it is possible to find better rules. The rules above do not take into account arrival frequency of the orders or capacity perturbations for example.

Another question is how to define good rules in case we do not have the complete information on the characteristics of the orders and on the capacity profile?

Using the MDP modeling and the Reinforcement Learning approaches as we propose here (Chapter 2) we obtained a learning agent (decision maker) that improves the previous results obtaining an average reward of **8.62** in the case of 10 types of jobs. We use a simulation based approach where the agent learns to take decisions while interacting with a simulated environment. This agent learns to take decisions based on the composition of the batch of arrivals (i.e., amount of each type of orders), some information on the status of the capacity profile, and some delayed signal evaluating its previous decision. The details of this model and the solutions we propose are discussed in Chapter 4. Here we want to mention that in order to get an effective agent design (see research question 1) we make a non-trivial step by introducing an iterative procedure that allows us to work with a strongly reduced action space. Moreover we remark that parameter tuning is a non-trivial matter, that requires a lot of trial and error before it works satisfactorily (see research question 2).

Since the number of situations that the agent must learn is quite large (in this case approximately 2^{35}), it is difficult to interpret what the agent has exactly learnt. As for average reward, the obtained decision rule - based on well tuned Reinforcement Learning - compares well with respect to the simple heuristics (see research question 3). However it has the disadvantage that it works like a black box for a human decision maker. It would be nice if we could open the black box up to some degree (see research question 4). Using simple data mining techniques we find the following rules that approximate the agent's behavior:

1. If the total used capacity over the planning horizon is in [0,7] then orders should be chosen according to the following priority sequence for order types (10,9,8,6,5,4). Other orders should be rejected

2. If the total used capacity over the planning horizon is in [8,10] then orders should be chosen according to the following priority sequence for order types (10,9,8,5,6,4). Other orders should be rejected

3. If the total used capacity over the planning horizon is in [11,22] then orders should be chosen according to the following priority sequence for order types (10,9,8). Other orders should be rejected

Note that using these rules we never accept orders of type 1, 2 and 3. Orders of type 4, 5 and 6 are only accepted when the total used capacity over the planning horizon is not too high and only when there are no orders of type 10, 9 and 8. This new heuristic obtained an average reward of **9.02** improving all the previous results. A general framework for generating new heuristic rules by extracting the knowledge in RL-agents is presented in Chapter 4.

The learning process for the agent is governed by a set of learning parameters. Based on our experimental results, we define and use a general methodology for the automatic tuning of these parameters. For details we refer to Chapter 5.

1.5 Outline of the thesis

The remainder of this thesis is organized as follows. In Chapter 2 we present the basic theory of Reinforcement Learning with the underlying theory of Markov Decision problems and the main learning methods we deal with in this thesis. We end the chapter with a discussion on the general framework for the application of Reinforcement Learning in Order Acceptance.

To get initial insight into various methods of RL we study in Chapter 3 a simple OA problem. We consider systems with a single server without a queuing facility, and a finite number of types of orders with stochastic dependent arrivals and deterministic processing times. This simple OA problem can be solved exactly to optimality. So we can use an optimal solution to compare its structure with the results from several RL algorithms. Thus this prototype problem allows us to get a better understanding of these RL methods. This chapter is part of a published report, (Mainegra-Hing et al., 2001). In Chapters 4, 5 and 6 we study more complex OA problems and the application of a specific RL method.

Chapter 4 is dedicated to OA on a single resource with batch arrivals of stochastically independent orders. Orders are loaded over a planning horizon at discrete equidistant time moments. Stochastic capacity perturbations are considered here. We also discuss a general class of heuristics for OA that is used to compare with the RL results. Moreover we set up a general framework to interpret the results of RL by extracting heuristic rules from the knowledge learned by the RL-agent. The general class of heuristics and the framework are used throughout the remaining chapters. An earlier and short version of this chapter is published, (Mainegra-Hing et al., 2002). Part of the latest version presented here has been accepted for publication; specificaly the problem description (Section 4.1), the model (Section 4.2), the general class of heuristics (Section 4.3) and six cases of experimental results including the last two cases presented here (Section 4.5), (Mainegra-Hing et al.,).

Chapter 5 extends the model from Chapter 4 by discussing OA in a multiresource shop where the order types define a fixed route through a job shop. The question is whether the RL approach will also work in this more complicated setting. We propose a methodology for tuning the learning parameters of the RL based on experimental experience. To handle the complexity of the problem we use here the concept of Partially Observable Markov Decision problems. In particular we define useful features to facilitate the learning process.

In Chapter 6 we consider Reinforcement Learning supporting Order Acceptance together with other kinds of decisions in an integrated planning approach. The idea is to show how RL may also be useful in a more general and realistic decision making process. Here we consider three possible cases. The first one considers choosing a routing in a multiresource shop, the second one considers outsourcing decisions, and the last one includes a due-date and price negotiation with the customer.

Finally, in Chapter 7 we present the conclusions and propose topics for further research.

Chapter 2

Reinforcement Learning

Reinforcement Learning studies an intelligent system (agent) that learns to achieve a goal through interaction with its environment. The idea of learning from interaction started in the early days of Artificial Intelligence but it was not thoroughly explored until the last 20 years. During that period it has grown into one of the most active areas in machine learning. In previous machine intelligence paradigms such as symbolic processing or supervised learning the intelligent system is supposed to receive some knowledge information beforehand, process such information and use it as its knowledge base. Unfortunately there are many situations where we do not have this a priori information.

In the paradigm of Reinforcement Learning, the agent is supposed to gather such information sequentially. At the same time it is learning from the gathered information. The collection of information and the learning process occur simultaneously by the interaction between the agent and the subject that is to be learned. The information which the agent receives does not need to be in the form of complete rules (the well known IF_THEN rules), as in a symbolic processing paradigm, or like perfect match pairs examples (situations with correct answers), as in supervised learning. The information can be just a simple signal (reward leading to reinforcement) related to the latest interactions. Instead of starting from given examples of desired behavior, the learning agent must discover by trial and error how to behave in order to get the best reward.

An RL-system has mainly two components: the Environment where the process to be studied takes place, and an agent[1] (RL-agent) who should be able to learn to control the process. The Environment in general, is assumed to be a semi-Markov decision process (SMDP). In Section 2.1 we introduce the theory of SMDP. In Section 2.2 we present the general RL-agent we will be working with in this thesis.

[1]Although we focus here on a single agent system, multiagent systems in which a group of agents communicate and cooperate, have also been considered in the context of RL-systems.

In this context we may situate Order Acceptance problems with incomplete information by considering the decision maker as an RL-agent who has to act in an unknown environment where orders arrive and may be accepted or rejected. We consider several degrees of incomplete information through the different models in the next chapters. Using SMDP models we simulate the dynamics of the environment. Then through interaction, with the simulated environment, the RL-agent who does not know about such dynamics, should learn the optimal acceptance behaviour. In Section 2.3 we discuss the general framework for the application of Reinforcement Learning in Order Acceptance.

2.1 The environment: A Semi-Markov Decision Process

A semi-Markov decision process (SMDP) is a model for sequential decision making in dynamic systems under uncertainty. A semi-Markov decision process consists of the following elements:

- The state space characterizing the system: S
- The action space defining the decisions that can be taken: A
- The decision epochs, i.e., the moments in time at which decisions have to be taken. Decision epochs do not necessarily occur at equidistant points in time. When the decision epochs are equidistant in time, the process is called a Markov decision process (MDP). In general the time $d(s,a)$ to the next decision epoch depends on the current state s and the action a taken at the current decision epoch.
- The transition probabilities describing the system dynamics

  $p(s,a,s')$ is the probability that when the action a is chosen in the actual state s, the next state in the next decision epoch, is s'.
- The immediate reward function $rew(s,a,s')$, i.e., the reward received when in state s action a is taken and the next state is s'.

In this thesis we consider that an explicit model for an SMDP is the tuple $\{S, A, d, p, rew\}$ characterizing the process. The term "Markov" is used because the dynamics of the system has the Markov property, i.e., the transition probabilities and the reward function only depend on the current state and the action taken in that state. We consider here finite SMDP, i.e., the state and action space are finite.

The semi-Markov decision problem consists of finding a prescriptive rule to choose an action for each state at every decision epoch such that a certain criterion is optimized. Such a prescriptive rule constitutes a policy. From the theory of finite SMDP it follows that there exists an optimal deterministic decision policy, i.e., a mapping from the state space to the action space $\pi : S \rightarrow A$, see (Puterman, 1994).

The optimality criterion defines a performance measure of the system. The most common optimality criteria are:

- Total expected reward for a finite planning horizon.
- Total expected discounted reward over an infinite planning horizon.
- Average expected reward per unit of time.

In this thesis we focus on an infinite horizon with total expected discounted reward. For the purpose of our research it is not relevant whether we consider a finite or an infinite horizon. However an infinite horizon makes our simulation system simpler. One does not need to consider horizon length or a set of terminal states. Furthermore in infinite horizon problems, unlike in finite horizon problems, one can look for an optimal policy in the set of stationary policies, i.e., independent of the time moment.

The total expected discounted reward has the most complete and general theory, with the fewest requirements, particularly it is not necessary to analyze the chain structure of the Markov chains generated by stationary policies as in the average criterion (Puterman, 1994). A discount factor $0 < \gamma < 1$ corresponds to the idea that rewards are less attractive in the far future. Given a policy π, the state-value function V^π is defined for each state s as the total expected discounted reward starting in state s and following policy π:

$$\begin{aligned} V^\pi(s) &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^{T_k} r_k | s_0 = s \right\} && (2.1) \\ &= \sum_{s'} p(s, \pi(s), s') \left[rew(s, \pi(s), s') + \gamma^{d(s,\pi(s))} V^\pi(s') \right], \\ &= R(s, \pi(s)) + \gamma^{d(s,\pi(s))} \sum_{s'} p(s, \pi(s), s') V^\pi(s') && (2.2) \end{aligned}$$

where r_k is the immediate reward at the decision epoch k and T_k is the elapsed time between $t_0 = 0$ and the $k-th$ future decision epoch. $R(s, \pi(s))$ is the expected immediate reward.

The objective is to find an optimal policy π^* such that $V^{\pi^*}(s) \geq V^\pi(s)\ \forall s \in S, \forall \pi$. This optimal policy can be found by any of the traditional methods from Probabilistic Dynamic Programming in the case of complete information about the model. For more on these methods see (Winston, 1994) and (Puterman, 1994). Linear programming, value iteration and policy iteration methods are the most widely used, and they are basically based on the Bellman Equation (2.1) where all the details about the model are assumed to be known. The value for the optimal policy $V^*(s)$ satisfies the following relation:

$$V^*(s) = \max_a \left[R(s,a) + \gamma^{d(s,a)} \sum_{s'} p(s,a,s') V^*(s') \right]. \tag{2.3}$$

So far we presented the classic SMDP setting. Traditional solution methods for this problem require full knowledge about the model parameters (i.e., transition probabilities, reward function, etc.). However, from previous work of the Artificial Intelligence community on learning optimal decision policies a slightly different concept, the action-value function Q^π appeared to be more advantageous (Watkins & Dayan, 1992). This function is defined for each state-action pair (s,a) as the total expected discounted reward starting in state s , taking action a and thereafter following policy π:

$$\begin{aligned} Q^\pi(s,a) &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^{T_k} r_k | s_0 = s,\ a_0 = a \right\} \qquad (2.4) \\ &= R(s,a) + \gamma^{d(s,a)} \sum_{s'} p(s,a,s') V^\pi(s'). \end{aligned}$$

In terms of this action-value function under an optimal policy π^* the optimality relation is given by

$$Q^*(s,a) = rew(s,a) + \gamma^{d(s,a)} \sum_{s'} p(s,a,s') \max_{a'} Q^*(s',a'). \tag{2.5}$$

The importance of the action-value function $Q^*\,(s,a)$ is to remedy the insufficiency of the state-value function $V^*(s)$ to reconstruct the optimal policy purely from its values. Note from Equations (2.3) and (2.4) that

$$V^*(s) = \max_a Q^*(s,a) \tag{2.6}$$

An optimal policy follows directly from the optimal action-value function through:

$$\pi^*(s) = \arg\max_a Q^*(s,a). \tag{2.7}$$

Note that in this relation there is no reference to other information than $Q^*\,(s,a)$ itself. This relation can easily be reformulated in terms of $V^*(s)$ as

$$\pi^*(s) = \arg\max_a \{ R(s,a) + \gamma^{d(s,a)} \sum_{s'} p(s,a,s') V^*(s') \},$$

but to use this formula we need a lot of model information, since all model parameters enter in the formula besides the value function. In problems with incomplete information this is a disadvantage for formulating transparent learning algorithms. Therefore, in this thesis we focus on learning the function $Q^* (s, a)$. In Section 2.2 we discuss how an RL-agent learns to approximate the action-value function without using an explicit model formulation. Once a good approximation is found the corresponding policy follows directly from the Equation (2.7).

2.2 The agent

RL can be viewed as an incremental method for solving an SMDP without knowing a priori the dynamics of the problem but - instead - by receiving information over time about the states and the reactions (rewards) to the chosen actions. An RL-system has mainly two components, an Agent (RL-agent) and its Environment. The Environment is assumed to be in general a Semi-Markov decision process where the actions are controlled by an Agent. It is characterized by states, rewards and transitions as introduced in Section 2.1.

The RL-agent is characterized by:

- *goal*: the agent's optimality criterion, which should be optimized through its behavior.
- *knowledge*: processed and saved information obtained by communication with the environment that can be used by the agent in order to decide on its behavior;
- *behavior*: the way the agent chooses to interact with its environment in order to achieve its goal. It is also called the agent's policy and it matches the perceived state from the environment and the taken action by the agent;
- *learning method*: the mechanism by which the agent updates its knowledge.

Here we present the general structure of the RL-agent we will be working with in this thesis.

2.2.1 Agent-Environment

Figure 2.1 summarizes the communication between the RL-agent and its environment. At each decision moment the agent (1) observes the *current state* of the environment and (2) performs an *action* selected according to its decision

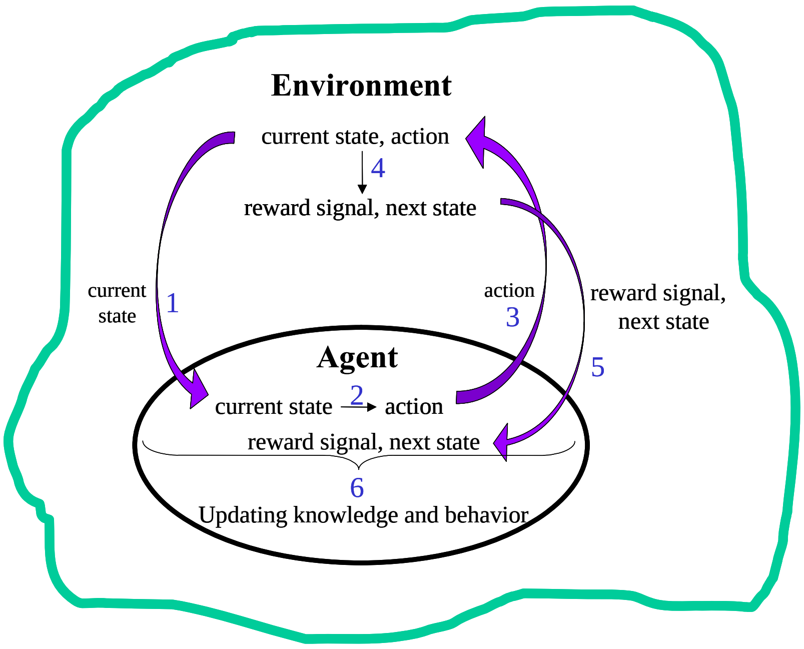

Environment
current state, action
4
reward signal, next state
current state
1
action
3
reward signal, next state
5
Agent
current state
2
action
reward signal, next state
6
Updating knowledge and behavior

Figure 2.1: *Agent-Environment Interaction*

policy. As a result of the (3) received *action* in the environment (4) a transition to a *next state* takes place and a reinforcement, or *reward* signal is generated (5). The reward signal and the new state are received by the agent and (6) may be used through the agent's learning method in order to update its knowledge about the environment, and consequently it can update its behaviour. Rewards and state transition functions may in general be stochastic, and the underlying probability distributions are assumed not to be known to the agent.

Nowadays the RL subject comprises a wide range of problems and related algorithms. As for the problems there are two fundamental categories: the RL evaluation (prediction) and the RL control problems. The first problem concerns the evaluation of a given policy, that means having the agent following a fixed policy to determine a certain performance measure of such a policy (e.g., the total expected discounted reward). The control problem is a more difficult one. It is to find an optimal policy that maximizes a certain performance measure. Our focus here is on RL control using Q-learning (QL) methods that try to learn the optimal action value function Q (Equation (2.5)) as a way to obtain an optimal policy. For semi-Markov decision processes this is a quite natural approach in our opinion (these are also the most developed and used methods) but we should mention that it is not strictly necessary to do Q-learning in order to solve the RL control problem. Methods that search directly in the policy space (genetic algorithms, genetic programming, simulated annealing) have also been used. For a good introduction on RL see (Sutton & Barto, 1998).

In the following subsections we present the agent's structure we will be working with throughout this thesis.

2.2.2 Goal

The agent's goal is to maximize the reward received over time. For the reasons explained before (Section 2.1) we consider here as optimality criterion the expected total discounted reward.

2.2.3 Knowledge representation

Here we consider the RL-agent's knowledge as an estimation of the optimal action-value function Q as in Equation (2.5). The agent can initialize this estimation arbitrarily or by taking into account some prior knowledge. This estimation is updated during the learning process as we explain in Section 2.2.5. The classical representation for the estimation of the action-value function is a backup table representation. In this case for each state-action pair (s, a) there is an entry in the table which is the corresponding approximated action value $Q(s, a)$. In the simple OA problem we present in Chapter 3 we use this type of knowledge representation. However, this way of representing knowledge limits the size and complexity of the solvable problems. It is difficult to represent the estimated Q-values for problems with an extremely large or even continuous state space. Hence function approximations could be used that may generalize and interpolate for states and actions never seen before. Some function approximations include decision trees, artificial neural networks, and various kinds of multivariate regression, see (Sutton & Barto, 1998). An Artificial Neural Network (ANN) is an example of such a function approximation, with a massively parallel distributed structure. Such structure and the capability to generalize make it possible for ANN to solve complex problems. For ANN there is a vast literature, we specifically refer to (Haykin, 1999). Many of the most successful applications of RL use ANN to represent the learning knowledge, see (Tesauro, 1994; Boyan & Littman, 1994; Zhang & Dietterich, 1995; Singh & Bertsekas, 1997).

An ANN is made up of simple processing units typically known as neurons. The most widely used ANNs are of type multilayer perceptron (MLP) where neurons are arranged in layers. For reasons of simplicity we will focus here on multilayer perceptrons with one hidden layer and a single output, which will be referred to as $I - \theta - 1$ perceptrons where I and θ are the sizes of the input and the hidden layer, the 1 is for the single output. This notation is taken from

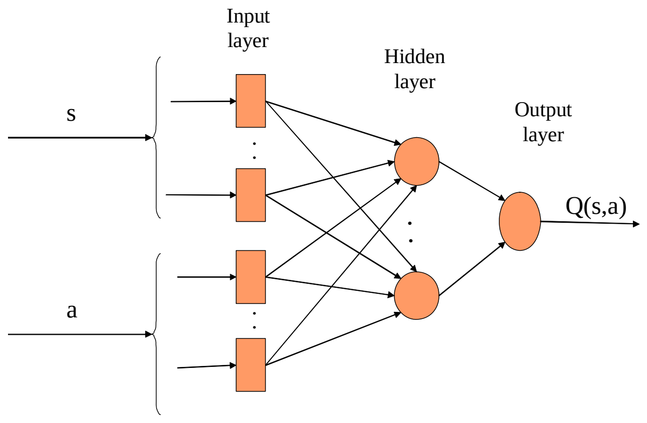


Figure 2.2: *An $I-\theta-1$ perceptron for representing the Q-values, where I and θ are the sizes of the input and the hidden layer*

Haykin (Haykin, 1999). See Figure 2.2 as an example of using an $I-\theta-1$ perceptron for representing the Q-values.

The inputs of the $I-\theta-1$ perceptron are features corresponding to a specific codification for the state s and an action a for that state. The output is a parametric approximation to the corresponding Q-value $Q(s,a)$.

Each input is connected to each neuron in the hidden layer, and each hidden neuron is connected to the output neuron. All these connections are weighted. It is also a common practice to apply an external bias to each neuron expanding the range of functions that the ANN may effectively approximate.

Thus an ANN representing the action-value function Q is parametrized by the weights set w. The set w is defined as $w = \{W, b, W_o, b_o\}$, where W is an Ixθ matrix of the weights from the input layer to the hidden layer, b is a $\theta-$dimensional vector of all the biases to each hidden neuron, W_0 is a θ-dimensional vector of the weights from each hidden neuron to the output neuron, and b_0 is the bias of the output neuron. Each neuron has an associated transfer function which defines the neuron's output as a function of its input.

As transfer functions we use at each hidden neuron the hyperbolic tangent function which defines the output of each neuron given an input x as follows:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The input x_j to the hidden neuron j is $x_j = b_j + \sum_{i=1}^{I} Input_i * W_{ij}$.

For the output neuron we use a linear transfer function. This design is known to work as a universal function approximation such that given a sufficient number of hidden neurons and by properly adjusting the weights, it could approximate any measurable function with arbitrary accuracy. The output of such an $I - \theta - 1$ perceptron for a given $Input$ (1xI dimensional) and a set of weights w is given by

$$Q(Input) = \tanh(Input * W + b) * W_o + b_o.$$

The knowledge of an agent using such an ANN is in the set w. Updating the knowledge consists of adjusting the weights to obtain an appropriate approximation to the optimal Q values. To update the weights we use the traditional backpropagation for a given training pair $(Input, target)$, see (Haykin, 1999). The training pair is obtained at each agent-environment iteration, see (Sutton & Barto, 1998), (Bertsekas & Tsitsiklis, 1996). We explain this in more detail in Section 2.2.5 where we present the learning method we use in this thesis. Next, in Section 2.2.4 we explain how the agent behaves based on its knowledge.

In the application of multilayer perceptrons matters like codification of the inputs and number of hidden nodes are to be defined that may influence the success of the application. They determine the size of the ANN as the number of weights which is $(I + 2)\theta + 1$ for the $I - \theta - 1$ perceptron. We discuss these issues in the following chapters during the presentation of our computational experiments.

2.2.4 Behavior

The behavior defines how the agent chooses the actions (e.g. greedy policy[2], exploratory policy[3]). The ideal action in a given state is the one which maximizes the action-value function, the greedy action. However if we always choose the greedy action based on the actual knowledge, many relevant state-action pairs may never be visited due to the inaccurate estimation of the action-value function. Particularly at the beginning of the learning process we need to explore as much as possible.

Efficient exploration is fundamental for learning. Too much exploration can cause nearly random behavior and too little can lead to non-optimal solutions. This is known as the exploitation-exploration trade-off. Exploitation deals with the use of the available knowledge for example by choosing the greedy actions. Exploration increases experience for example by choosing actions at random. Two well known exploration strategies are the following:

$\epsilon-$**greedy exploration:** With probability 1-ϵ one chooses a greedy action

[2]A greedy policy is one in which all the actions are greedy. An action a is greedy if $a = \arg\max_{a'} Q(s, a')$

[3]An exploratory policy can choose actions at random which can be useful when there is not enough knowledge, for example at the beginning of the learning process.

with respect to the current estimate of the action-value function and with probability ϵ a random action. The parameter ϵ should decrease during the learning process.

Boltzmann exploration: In state s, an action a is chosen with probability

$$\frac{e^{\frac{Q(s,a)}{T}}}{\sum\limits_{a'} e^{\frac{Q(s,a')}{T}}},$$

where T is called the "*temperature*" parameter and T should decrease during the course of the algorithm.

For reasons of simplicity we use in our experiments the $\epsilon-$greedy exploration rule with a parameter ϵ decreasing over time, which is also the most widely used strategy in the literature. See (Thrun, 1992), and (Ratitch & Precup, 2003) for surveys on exploration.

2.2.5 Learning method

The learning method deals with iterative updates of the agent's knowledge. As we mentioned before we focus here on Q-learning methods. The concept of Q-learning was first described by Watkins (Watkins, 1989). Several variants of the Q-learning method have been proposed since then. The classical method and convergence properties were developed using backup tables to represent the knowledge structure. Here we present the most standard tabular methods and their counterpart using neural networks as the parametrized function approximation.

The method is based on estimations of the Q-values which are updated after each agent-environment interaction. The agent starts with some estimation (arbitrarily or using some a priori information in case it is available). At each decision moment t in which the environment is in state s_t the agent chooses an action a_t according to its behavior (see Section 2.2.4). The environment reacts to the taken action by giving a reward $r_{t+1} = r(s_t, a_t)$ to the agent and changing to a new state s_{t+1} in a next decision epoch $t+1$ that occurs after $d(s_t, a_t)$ units of time. With this new information the agent updates the Q-values and decides upon a new action a_{t+1} for the present state s_{t+1}, etc. The update rule for this method, given the experience tuple $< s_t, a_t, r_{t+1}, s_{t+1} >$, is as follows:

$$\begin{aligned} Q_{t+1}(s_t, a_t) &= Q_t(s_t, a_t) + \alpha_t \delta_t \\ \delta_t &= r_{t+1} + \gamma^{d(s_t,a_t)} \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t), \end{aligned} \tag{2.8}$$

where γ is the discount factor, α_t is the learning rate (see below for details) and δ_t is known as the temporal-difference. The idea of this formula is based on the application of the Robbins-Monro stochastic approximation algorithm to estimate an unknown mean (see Equation 2.4) using a single sample, and can also be viewed as a stochastic approximation form of the policy iteration algorithm.

The version of the update rule, when using ANN instead of backup tables to approximate the Q-values, is based on the update of the weight vector w as follows:

$$\begin{aligned} w_{t+1} &= w_t + \alpha_t \delta_t \nabla_{w_t} Q(s_t, a_t, w_t) \\ \delta_t &= r_{t+1} + \gamma^{d(s_t,a_t)} \max_a Q(s_{t+1}, a, w_t) - Q(s_t, a_t, w_t). \end{aligned} \tag{2.9}$$

In this case the update rule can be viewed as a gradient descent method in the ANN weight space that aims to minimize the temporal-difference term.

Q-learning is an off-policy method, meaning that the policy learned about does not need to be the same as the one used to select the actions. In particular it learns about the greedy policy, while the agent follows a policy involving exploratory actions.

In the case of the backup table representation this method converges to the optimal action values with probability 1 as long as all pairs (s, a) are visited infinitely often and the learning rate is reduced over time according to the usual stochastic conditions for the learning rate (see conditions (2.10), see this result in (Watkins & Dayan, 1992)).

Convergence proofs, for the case when using neural networks, are much more restricted than when using a backup table representation. Only for some specific well-structured problems convergence can be guaranteed (see (Bertsekas & Tsitsiklis, 1996), page 337). A problem here is that once the weight vector is updated all the Q-values are also changed and there is no guarantee of reducing the temporal-difference term at each iteration. Thus the algorithm has the risk of divergence. However there is a number of successful applications using RL with ANN. It is an active and open research area. Recent work (Ratitch, 2005) has been done that shows how some attributes of the MDP may be used to predict the performance of the RL algorithms and help with the design of the RL-system.

An RL approach involves an interesting parameter setting dimension. Some tuning is necessary before one finds an efficient parameter setting. Here we consider two sets of different parameters. These parameters are related to learning, and bootstrapping. We discuss these issues in the next two paragraphs.

Learning rate: the parameter α

The learning rate or step-size α_t is a measure of how fast new information is incorporated in the general knowledge. In the previous section $\alpha_t \in [0, 1]$ was introduced, but its behavior over time was not discussed. There are well known conditions for the learning rate from the general stochastic approximation theory that are also used in the convergence proof of the QL methods:

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty. \tag{2.10}$$

The first condition is to guarantee that steps are large enough to overcome initial conditions or random fluctuations and the second is to guarantee that steps become small enough to assure convergence. Sequences of step-size parameters that meet both conditions often converge very slowly or need much tuning in order to obtain satisfactory convergence rates. It is surprising how many successful applications usually do not use step-sizes satisfying these conditions but for example constant step-size. Using a constant learning rate the second condition is not met, and the learned function will never completely converge, but will vary in response to the most recently received rewards. However, this can be desirable in case of a nonstationary environment. The conditions also require separate learning rates for each state-action pair which again could be not very practical for large sized problems. In our experiments we use a decreasing time-varying learning parameter independent of the state-action pair.

Bootstrapping degree: the parameter λ

There is a class of more advanced updating rules. Instead of updating the estimated value function based solely on the approximated value of the immediate successor state (see for example update rule 2.8), one can think of methods that base the updates on weighting the values of next states (bootstrapping) as suggested by the definition of $V^\pi(s)$ in Equation 2.1. This is precisely the idea of methods that introduce the parameter λ as the weighting factor.

The Q-learning updating rule is based on just the one next reward, using the value of the state one step later as a proxy for the remaining rewards. The Monte Carlo methods for episodic tasks ((Bertsekas & Tsitsiklis, 1996),(Sutton & Barto, 1998)) perform a backup for each reward from the current state until the end of the episode. One kind of intermediate method consists of performing a backup based on an intermediate number of steps: more than one, but less than all of them until termination. The updating rule at iteration t using n steps is as follows:

$$\begin{aligned} Q_{t+n}\left(s_t, a_t\right) &= Q_t\left(s_t, a_t\right) + \alpha_t(R_t^{(n)} - Q_t\left(s_t, a_t\right)), \\ R_t^{(n)} &= r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n} + \gamma^n V_t\left(s_{t+n}\right) \end{aligned}$$

However, n-step backups are rarely used because they are inconvenient to implement. Computing n-step returns requires waiting n steps to observe the rewards and states. It is possible to use an average of any set of returns, even an infinite set, as long as the weights on the returns are positive and sum to 1:

$$\begin{aligned} R_t^{\lambda} &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_t^{(n)} \\ &= (1-\lambda)\left[\left(r_{t+1} + \gamma V_t\left(s_{t+1}\right)\right) + \lambda\left(r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t\left(s_{t+2}\right)\right) + ...\right] \end{aligned}$$

$$Q_{t+n}\left(s_t, a_t\right) = Q_t\left(s_t, a_t\right) + \alpha_t\left(R_t^{\lambda} - Q_t\left(s_t, a_t\right)\right).$$

This is a forward view, a theoretical one that is not directly implemented because it needs knowledge of what will happen some steps later. There is an equivalent but mechanistic view: a backward view, based on the following relation:

$$\begin{aligned} &Q_{t+n}\left(s_t, a_t\right) = Q_t\left(s_t, a_t\right) + \\ &\alpha_t\left((1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\left(\sum_{m=1}^{n}\gamma^{m-1}r_{t+m} + \gamma^n \max_{a'} Q_t\left(s_{t+n}, a'\right)\right) - Q_t\left(s_t, a_t\right)\right) \end{aligned}$$

$$Q_{t+n}\left(s_t, a_t\right) = Q_t\left(s_t, a_t\right) + \alpha_t \sum_{m=1}^{\infty}\left(\gamma\lambda\right)^{m-1}\delta_{t+m}$$

Note that we use here that $V_t\left(s\right) = \max_{a'} Q_t\left(s, a'\right)$. The last expression was obtained swapping the order of the two summations and using algebraic transformations, see (Bertsekas & Tsitsiklis, 1996) and (Sutton & Barto, 1998). The backward view provides an incremental mechanism for approximating the forward view. It is like looking back in the time and correcting previous predictions by using the information on actual states. In the backward view there is an additional memory variable associated with each state and action: the eligibility trace $e_t(s, a)$ that can be viewed as a temporary record of the occurrence of an event that makes it eligible for further learning. The updating rule can be as follows:

$$\begin{aligned}
Q_{t+1}(s,a) &= Q_t(s,a) + \alpha_t \delta_t e_t(s,a) \\
\delta_t &= r_{t+1} + \gamma^{d(s_t,a_t)} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \\
e_t(s,a) &= I_{s,s_t}\ I_{a,a_t} + \gamma^{d(s_t,a_t)} \lambda e_{t-1}(s,a),
\end{aligned}$$

where $I_{x,y}$ is a comparison function[4].

This update corresponds to the SARSA method where the Q-values are updated using the real next action a_{t+1} instead of the greedy one used in Q-learning. In each step the eligibility traces for all states and actions decay by $\gamma\lambda$ and - in addition to that - for the current state-action pair the trace is augmented by one. At any time eligibility traces record which states have recently been visited.

For the Q-learning update in case that an exploratory action occurs the eligibility traces are set to zero. Next the Watkins-Q(λ): (backup table representation)

$$\begin{aligned}
Q_{t+1}(s,a) &= Q_t(s,a) + \alpha_t \delta_t e_t(s,a), \\
\delta_t &= r_{t+1} + \gamma^{d(s_t,a_t)} \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t), \\
e_t(s,a) &= I_{s,s_t}\ I_{a,a_t} + \begin{cases} \gamma^{d(s_t,a_t)} \lambda e_{t-1}(s,a), \text{ if } a_{t-1} \text{ was greedy} \\ 0, \text{ otherwise} \end{cases} .
\end{aligned}$$

For the case of a neural network representation of the Q-values eligibility traces do not correspond to each state-action pair but a trace is defined for each component of the parameter vector. In this case it can be interpreted as a smooth parameter change proportional to the gradient function. Next we show the update rule for this case, the Watkins-Q(λ, w): (ANN representation)

$$\begin{aligned}
w_{t+1} &= w_t + \alpha_t \delta_t z_t \\
\delta_t &= r_{t+1} + \gamma^{d(s_t,a_t)} \max_a Q(s_{t+1}, a, w_t) - Q(s_t, a_t, w_t) \\
z_t &= \nabla_w Q(s_t, a_t, w_t) + \begin{cases} \gamma^{d(s_t,a_t)} \lambda z_{t-1}, \text{ if } a_t \text{ was greedy} \\ 0, \text{ otherwise} \end{cases} .
\end{aligned}$$

Eligibility traces provide a link between Monte Carlo ($\lambda = 1$) and one-step RL methods ($\lambda = 0$).

The target values used when $\lambda = 1$ are unbiased samples but may have significant variance since each depends on a long sequence of rewards from stochastic transitions. By contrast when $\lambda = 0$ the target values have low variance since the only random component is the reward of a single transition,

[4] $I_{x,y} = \begin{cases} 1, & \text{if } x = y \\ 0, & \text{otherwise} \end{cases}$

but they are biased by the inaccuracy of the current estimated values.

It is the belief that the parameter λ (for $\lambda \neq 0$) as distributing credit throughout sequences of actions, leads to faster learning. Experimental results (Sutton, 1988) for a specific prediction problem show that intermediate values of λ that are close to 0 give the best results. However a full understanding of how λ influences the rate of convergence is yet to be found. Van Roy (Van Roy, 1998) has suggested that it could be desirable to tune λ as the algorithm progresses but even both directions (starting with $\lambda = 0$ and approaching 1, and the other way around) have been advocated. For reasons of simplicity in our experiments for the prototype type problem we shall take in some cases $\lambda > 0$, λ constant.

Model based learning

An important issue in learning techniques is the learning speed. Besides the bootstrapping idea there are several other QL methods that aim to speed up the standard QL. Here we focus on a model-based approach.

So far we have discussed model-free RL methods where an agent learns about an optimal policy without ever knowing explicitly the model of the environment it interacts with. However, these methods may have been using inefficiently the data they gather and therefore may need too much time to achieve good performance. Model-based methods are alternative methods in which the agent incorporates in its learning process information that it gathers about the model.

There is an open debate about the question whether model-based or model-free methods are better. A fact is that, although model-free methods use less computation time per experience, they do not exploit all the information they gather and they make inefficient use of it. This could be one of the reasons for the large amount of experience these methods need in order to obtain a good performance. In this sense it is believed that incorporating some knowledge from the accumulated experience in the learning procedure can in fact help with the learning speed. But model-free methods are much simpler and are not affected by biases in the design of the model. Recently Kerns and Singh (Kearns & Singh, 1999) argued that both methods have roughly the same sample complexity: rather rapid convergence to the optimal policy as a function of the number of state transitions observed.

Model-based methods, unlike model-free methods, need to store the domain model besides the value function. This is usually done by means of backup tables that scale with state and action spaces. Dynamic Bayesian networks can be used in many cases to represent models in compact form (Tadepalli & Ok, 1998) .

There are also different versions of this model-based approach. In our approach the idea is not to make a separation between the learning phase and the solution phase but to use an idea closer to RL methods. At each time step the model parameters ($\widehat{R}(s,a)$ and $\widehat{\Pr}(s,a,s^{'})$) are estimated and used in the update rule with:

$$\delta_t = \widehat{R}(s_t,a_t) + \gamma^{d(s_t,a_t)} \sum_{s'} \widehat{\Pr}(s,a,s^{'}) \max_a Q(s^{'},a) - Q_t(s_t,a_t). \qquad (2.11)$$

In the case that one knows the transition probabilities this approach is not very difficult. However even in the cases we know the arrival frequency per each type of order, it is not a straightforward matter to compute the summation in δ_t. Therefore we briefly present the Dyna-method which is a very intuitive and simple approach.

The model based Dyna-Q (Sutton, 1990) is an architecture that combines model learning and direct QL. It simultaneously uses experience to estimate a model and uses the experience and the estimated model to learn the optimal policy. The method uses one of the update rules as defined before. However, after the first updates (model and value function) with the data from the interaction between agent and environment, the agent includes additional updates of the estimated value function by simulating data from the estimated model. In our case this means simulating a prescribed number of arrival patterns besides the actual one.

This approach can easily be used for our models. In general we do not need the complete model but only a sample model. We present some experiences using these ideas in the next chapter.

2.3 Reinforcement Learning for Order Acceptance

In the next chapters we work with the application of RL to different models of the OA problem. The models differ in complexity, we start with a very simple prototype problem (single server, single arrivals) and we end up with integrated systems where the RL-agent should learn other kind of decisions besides the OA decision. Here we discuss some general considerations used throughout the rest of the thesis in the application of RL to these OA models.

2.3.1 A simulation model

RL is a simulation based approach to solve SMDP. Our simulation system has two main components: the simulation model of the environment and the RL-agent. The environment represents the system where the OA problem is defined and the RL-agent is the decision maker that should solve the OA problem. Orders arrive requesting some capacity and the RL-agent should decide on the acceptance or rejection of the orders.

OA problems under uncertainty can be modelled as SMDPs. The stochasticity in the models in this thesis are given by different sources of uncertainty: the stochastic arrival of orders (see Chapters 3-6), stochastic capacity perturbation (see Chapter 4), the stochastic processing time of the orders (see Chapters 5 and 6) and the stochastic negotiation of due-dates (see Chapter 6). In practice it is hard to obtain data such as the transition probabilities for the states of the model, even in the case that we know all the distributions of the stochastic processes mentioned above. That is why complete explicit models have limited meaning for applications. The advantage of using RL to solve SMDP is that it is not necessary to have a complete explicit model as described in 2.1 but a simulation model, i.e., a way to generate the transitions of the state instead an explicit formulation of the transition probabilities. Given an action in a current state, the simulation model of the environment should generate a next state and an immediate reward, that is exactly what we need to apply RL. When there is not enough information available to define a simulation model (e.g. we do not know the distributions of the stochastic processes), the RL-agent could interact then with the real environment. In such cases it is recommended to use some model-based methods to speed up the learning process. In the OA problem presented in Chapter 3 we explore some of these model-based methods. Since our focus is not in collecting data but on the RL solutions, in the rest of the thesis we only consider the simulation model.

To define the simulation model of the environment some considerations have to be done. First is to determine the decision moments. For reasons of simplicity we focus here on discrete time moments, even when orders may arrive continuously. However the elapsed time between two consecutive decision moments is not constant. This can be so since we do not consider decision moments when there is a unique possible decision, e.g. there is not enough capacity for the requesting orders, thus the next decision moment will be only when capacity is available (see Chapter 3). Another case is when we use a time-off in order to choose orders one by one from a batch of orders (from 4 on). Secondly we define the states that describe the environment at each decision moment. The state in the simulation model is characterized by two main components: orders requesting service and capacity profile, so there is a general structure for a state $s = (orders\_list, capacity\_profile)$. The information encapsulated in the state depends on the complexity of the environment, and we will discuss it for each model in the next chapters. The dynamic of the environment is assumed in general to be an SMDP.

The RL-agent uses a static structure to represent its knowledge. In the models considering multiple arrivals (4,5 and 6) it is necessary to know beforehand the total number of order types that may arrive to the system. This is because the component *orders_list* in the state should have an entry for each type of order. But the characteristics of each order (e.g., reward upon acceptance, processing time, due-date, etc.) may be only known after processing the order.

At each decision moment, the agent receives information about the state of the system. Depending on the agent's behavior and knowledge an action is taken about the acceptance or rejection of the orders. The agent receives an immediate reward as consequence of the taken action. Using the reward signal and the information about the next state the environment is into, the agent updates the knowledge, and a new cycle starts again.

2.3.2 Tuning RL-agents

One of the most difficult problems in the application of the RL methods is the adjustment of parameters. These parameters are the period of training, which we consider as the number of Agent-Environment iterations; and the learning and exploration rates. A characterizing property of an RL-agent using ANN knowledge representation is the number of hidden neurons (θ). Increasing θ increases the potential of the approximator but also increases the number of weights in the ANN to be updated at each iteration. Initially one could start with a small number of hidden neurons and increase it only when necessary. We combine all these parameters in a set that we call *learning schedule* with six parameters $(T, \alpha_0, \epsilon_0, T_\alpha, T_\epsilon, \theta)$. A learning schedule specifies:

- T: number of iterations of the learning process,
- α_0: initial value of the learning rate,
- ϵ_0: initial value of the exploration rate,
- T_α, T_ϵ: parameters for the learning and exploration rate-decreasing functions:

  $$\alpha_t = \frac{\alpha_0}{1 + \frac{t}{T_\alpha}}, \ \epsilon_t = \frac{\epsilon_0}{1 + \frac{t}{T_\epsilon}}, \qquad t = \overline{1...T},$$

  where T_α and T_ϵ respectively define the decreasing speed.
- θ: number of hidden neurons,

In our implementations these parameters have been chosen through experimental experience and are by no means optimized. It could be possible after extensive numerical experiments to extract regularities that may help to set guidelines for the tuning of the parameters. In 5 we propose a methodology for tuning some parameters based on experimental experience from Chapters 3 and 4.

Algorithm 2.1 sketches the QL method as we described above.

```
Require: learning schedule: $(T, \alpha_0, \epsilon_0, T_\alpha, T_\epsilon, \theta)$
Require: Agent with a knowledge structure Q [with m hidden neurons in case of ANN]
Require: SMD Environment where the Agent is situated
  Init knowledge Q
  Init $s_0$ in Environment
  for $t = 0$ to $T - 1$ do
    $a_t = \epsilon_t - greedyQ.Policy(s_t, \epsilon_t)$
    $(r_{t+1}, s_{t+1}, d_{t+1})$ = Environment$(a_t)$
    $target = r_{t+1} + \gamma^{d_{t+1}} \max_{a'} Q(s_{t+1}, a')$
    Update knowledge $Q(s_t, a_t, target, \alpha_t)$
    Update parameters: $\alpha_{t+1} = \frac{\alpha_0}{1+\frac{t+1}{T_\alpha}}$, $\epsilon_{t+1} = \frac{\epsilon_0}{1+\frac{t+1}{T_\epsilon}}$
```

Algorithm 2.1: *Q-learning*

2.4 Summary

In this chapter, we present the basic theory of Reinforcement Learning with the underlying theory of semi-Markov Decision problems and the main learning methods we deal with in this thesis. The RL methods discussed here can be viewed as suboptimal methods to approximate the action-value function of SMDP through simulation. The methods do not need an explicit model of the problem but a simulation model, which is useful for complex problems that are hard to model.

We also present a general idea on how OA problems under uncertainty can be modelled as SMDPs and solved by using RL methods. In the next chapters we discuss in detail different OA problems and the use of the RL approach.

Chapter 3

A simple Order Acceptance problem

In this chapter we present a simple Order Acceptance problem which is used as a starting point in our study.

In Section 3.1 we describe the problem and how it is modelled as an SMDP in Section 3.2. There we obtain the optimality equations and the threshold structure of an optimal policy. Methods from SMDP theory can easily cope with this problem when the complete model is known. So we can use an optimal solution to compare with the results from the RL algorithms. In Section 3.3 we discuss the application of RL to this simple problem and the experimental results are shown in Section 3.4. Finally in Section 3.5 we draw some conclusions and set guidelines for the more complex cases presented in the next chapters.

3.1 Problem description

In terms of the characterization (i)-(vi) given in Chapter 1 we assume the following:

(i) only one server is available, hence only one order can be processed at any moment in time,

(ii) order definitions are based on a finite number of types of orders, where each type has a specific processing time and immediate reward,

(iii) at most one arrival in each discrete time unit is kept, since a single server can serve only one job at a time. The probability of arrival is order type dependent,

(iv) rejection of an order affects only the immediate reward of that order,

(v) order processing requires an order type dependent deterministic service time from a single server,

(vi) the decision policy does not allow preemption of orders and when the server is busy the only option for a new order is rejection.

Note that in this case, the orders have the smallest possible set of attributes, for example the concept of a due-date does not play a role (conceptually it coincides with the processing time here), and batch arrivals of orders are excluded. Production capacity is modeled as a single server without a queueing facility.

In the context of SMDP theory, models of more realistic order acceptance problems can be studied. As mentioned before generalizations with respect to any of the characteristics (i)-(vi) are interesting from a point of view of applications in practice. In terms of these characteristics we present in the next chapters model formulations which take into account: (i) a more general capacity structure in production, (ii) orders with a larger attribute set including due-dates, (iii) batch arrivals, (iv) rejection of an order affects only the immediate reward of that order, and (vi) some uncertainty in order processing. In general this leads us to multi-server (parallel and/or in series) network problems with waiting lists (queueing) per server (or a common list for a set of servers) with batch arrivals of orders. As for characterization (iv) effects of order loss, we always consider in this thesis that rejection of an order affects only the immediate reward of that order. Nevertheless the case in this chapter is rich enough to demonstrate the complications that arise in finding an optimal decision policy.

3.2 The SMDP model

In this section we formulate the simple prototype order acceptance problem described above as an SMDP. Orders arrive in a single arrival process from a set of n order types. Order type i is characterized by a small set of attributes:

- p_i: processing time,
- r_i: reward for acceptance,
- q_i: probability of arrival.

Production capacity is considered as a single server that can only process one order at a time. Arrivals are checked every fixed period of time, and a decision should be taken immediately at that moment if the server is idle: accept or reject the arrived order. If the server is busy, the arrived order is lost. This prototype problem can easily be modeled as a discrete time Semi

Markov Decision Problem (SMDP) as sketched before. The state space is $S = \{0, 1, 2, ..n\}$ where 0 indicates that no order has arrived, and the action space is $A = \{0, 1\}$. Table 3.1 shows the one step state transition from the current state.

current state	$s \neq 0$		$s = 0$
action (a)	0	1	0
immediate reward $R(s,a)$	0	r_s	0
time until next decision $d(s,a)$	1	p_s	1
transition probability $\Pr(s,a,s')$	$q_{s'}$		

Table 3.1: *Dynamics of transition from current state s in the simple OA model*

Given the current state $s \neq 0$, which identifies the type of order arriving, there are two possible actions: reject $(a = 0)$ or accept $(a = 1)$ the arriving order. Depending on the chosen action the immediate reward may be 0 if the arriving order is rejected, or r_s if the order is accepted. The time until the next decision epoch also depends on the chosen action. If the arriving order is rejected, the next decision epoch would be one time unit ahead, when we check for a new arriving order. If the order is accepted, the server would be busy during the processing time of the accepted order (p_s), so the next decision epoch would be right after the termination of that service. In case the current state is $s = 0$, it means that there is no arriving order so the only option is rejection with zero immediate reward, the next decision epoch would be one time unit ahead. The next state s' does not depend on the current state nor the chosen action, so the transition probability to that state is the probability of arrival of an order of type $s' \neq 0$, or the probability that no arrival occurs $s' = 0$. Note that $\sum_{s=0}^{n} q_s = 1$.

The objective is to find a deterministic policy π:

$$\pi(s) = \begin{cases} 1 & \text{accept the arriving order } s \\ 0 & \text{reject the arriving order } s \end{cases}$$

which maximizes the performance of the system. The performance of the system is measured as the expected value of the total discounted reward. The Bellman equation for the value function V^* corresponding to the optimal policy π^* is given by

$$V^*(s) = R(s, \pi^*(s)) + \gamma^{d(s,\pi^*(s))} \sum_{s'} q_{s'} V^*(s'),$$

with γ the discount factor. The optimal action-value function $Q^*(s,a)$ is defined as

$$Q^*(s,a) = R(s,a) + \gamma^{d(s,a)} \sum_{s'} q_{s'} V^*(s'),$$

and satisfies

$$Q^*(s,0) = \gamma\beta,$$
$$Q^*(s,1) = r_s + \gamma^{p_s}\beta,$$

where $\beta = \sum_{s'} q_{s'} V^*(s')$ is defined by the problem instance. So an optimal policy in this case has a very special form: an order of type s may be accepted if and only if $r_s \geq (\gamma - \gamma^{p_s})\beta$ (namely $Q^*(s,1) > Q^*(s,0)$).

Hence this policy has a nice simple threshold structure. This can be seen as a first example of how this way of modeling guides to heuristic rules. We shall further elaborate on such in Chapter 4. The policy iteration algorithm finds the optimal policy π^* for this problem in at most n steps. Since we want to look at this problem from an incomplete information perspective, in general these traditional methods do not fit into our view. A straightforward way to deal with incomplete information is a two phase method. In the first phase the unknown parameters of the model are estimated. In the second phase, the complete problem with the estimated parameters is solved. In this problem estimation of parameters boils down mainly to estimation of the transition probabilities. Since the essence of the transition probabilities are the arrival intensities, this can simply be done using the historical arrival fraction of orders of type i as an estimator for q_i. Other methods that combine learning and solving without separation in two phases are those which attempt to learn action-value functions (Q-learning).

3.3 Reinforcement Learning approach

Here we present the application of Reinforcement Learning to the prototype order acceptance problem described before. We use QL methods that aim to approximate the optimal Q-value function as discussed in Chapter 2. The RL-agent should learn an OA-policy while interacting with the environment. The RL-agent uses information about the type of order arriving and in case the server is idle a decision will be taken. If an order arrives while the server is busy, it goes out of the system. Thus, given the state $s = (i)$ the agent chooses with certain probability the action a from $A(s)$ which maximizes $Q(s,a)$. The probability of choosing the action with highest Q-value increases during the training of the agent and it is controlled by an exploration strategy as explained in Chapter 2.

The application of Q-Learning to our prototype problem is quite straightforward since it is a simple model with the number of states depending on the different type of orders and only two actions that are possible for each state. The agent has always the possibility of choosing one of the two actions; which one is chosen depends on the current policy. The goal is to maximize the expected total discounted reward. Let us use $\gamma = 0.9$ as discount factor. The behavior of the RL-agent is according to an $\epsilon-$greedy exploration rule: with probability $1 - \epsilon$ the agent chooses a greedy action and with probability ϵ a random action. The parameter ϵ decreases over time according to the following rule $\epsilon_t = \frac{\epsilon_0}{1+\frac{t}{T_\epsilon}}$. Next we discuss the knowledge representation and the learning methods we use.

3.3.1 Knowledge representation

As we start with a very simple problem and a small case (in Section 3.4.1), just to get some insight into the application of the algorithms, it is worthwhile to try the backup table representation. This will allow us to study the influence of the parameter settings without the negative noise of using function approximation, in the QL-algorithms. In Section 3.4.1 we present the application of Q-learning using backup tables to a small case. There we discuss different parameter settings and alternative solutions. Even for this simple problem extensive experimentation was necessary in order to find appropriate parameters.

However, as our main interest is in the possibility of using ANN, we also explore this issue for this small case in Section 3.4.1. The Q-values are stored in an $I - \theta - 1$ perceptron as discussed in Section 2.2.3. The input of the perceptron is a codification of the state-action pair (s, a), in the following way. We use n binary inputs, one per each type of order. A one at input i indicates that an order of type i is requesting service. For the action we use a binary input, one for the acceptance and zero for rejection. Using this codification for the input, the size of the ANN is $(n + 3)\theta + 1$ which is in general (except for $\theta = 1$ and $n \geq 4$) a larger number of unknown parameters to be adjusted, than the number of Q-values to be learned $(2n)$. There are other ways of encoding this information, but this codification easily generalizes to the other models in the next chapters, and it enables us to extend the results obtained with respect to the parameters settings.

The output of the perceptron is a single value that is the approximation of the state-action value function $Q(s, a)$. After each decision, the agent receives information about the state transition: the new state and the reward value. With this information, error backpropagation is used to update the estimation of the Q-values (see Formula 2.8 and 2.9).

In Section 3.4.2 we present experimental results with cases of larger size using also ANN.

3.3.2 Learning methods

As explained before, we focus here on traditional Q-learning as introduced in Section 2.2.5. For all cases we present experiments using backup tables and ANN to represent the knowledge structure. However, as a matter of comparison we also try other alternatives:

1. For the small case of Section 3.4.1 we present the use of the bootstrapping parameter λ as introduced in Section 2.2.5.
2. For all cases, we present the application of the naive two-phases method. In this method the agent keeps an explicit estimated model. At each agent-environment iteration, the agent updates the frequency of arrivals for the arriving order and the respective reward value depending on the decision. Using policy iteration an optimal policy with respect to the current estimated model is found.
3. For all cases we present the application of the model based learning Dyna-method as introduced in Section 2.2.5. Also in this method the agent keeps an explicit estimated model which is updated at each iteration. But in this case, instead of solving the model, the method includes M extra simulated iterations from the estimated model in between two Q-learning iterations. The number M of simulated iterations can be made dependent of the iteration number t and in our experiments we use $M(t) = max(100, t)$ at iteration t.

3.4 Experimental results

In this section we present the application of QL methods, as discussed in Chapter 2, to some cases of the prototype problem presented in this chapter. First we introduce some notation for the performance measures. In Section 3.4.1 we study a small case with three type of orders. This small case introduce us in the use of the RL approach. Therefore we study the influence of the different parameters from the learning schedule. For this small case we discuss the different approaches discussed in Chapter 2: the naive model based method, learning using backup tables, the model based Dyna method, the use of bootstrapping and finally the use of ANN.

In Section 3.4.2 we present two larger cases with 5 and 10 jobs.

In each experiment we compute averaged performance criteria over 10 simulation runs and compare this performance to the optimal solution. We use the following performance criteria:

- MSE_Q_t: The mean square error of the (learned) actual Q_t-values compared to the (optimal) Q^*-values at iteration t.

- $MSE\_Q^{\pi_t}$: The mean square error of the Q$^{\pi_t}$-values corresponding to the (learned) policy π_t at iteration t, compared with the (optimal) Q*-values. The learned policy π_t is defined as the greedy policy with respect to the learned Q$_t$-values. Note that this measure provides information on how close to the optimal policy is the learned policy after t iterations in the learning process.

- $AvToRew_t$: The total average reward accumulated per unit of time up to the time corresponding to iteration t.

The Mean Square Error (MSE) is a common measure to determine proximity betwen two vectors.

$$\begin{aligned} MSE\_Q_t &= \frac{1}{|S\text{x}A|}\sum_{s\in S}\sum_{a\in A}\left(Q_t(s,a)-Q^*(s,a)\right)^2 \\ MSE\_Q^{\pi_t} &= \frac{1}{|S\text{x}A|}\sum_{s\in S}\sum_{a\in A}\left(Q^{\pi_t}(s,a)-Q^*(s,a)\right)^2 \end{aligned}$$

The Q_t vector is obtained from the learned RL-agent at iteration t, and Q^{π_t} is the exact Q-value function for the π_t policy. Note that $\pi_t(s) = \arg\max_{a\in A}(Q^{\pi_t}(s,a))$. The Q^{π_t} values are computed using the value-iteration algorithm.

The Q-learning aims to learn the optimal state-action Q-values, this means that the first measure should be zero. At the end what one is looking for is an approximation of these Q-values that leads to the optimal policy; even if MSE_Q_t is not zero we can have $MSE\_Q^{\pi_t} = 0$, meaning that we have learned an optimal policy. Note that on the other hand $MSE\_Q_t = 0 \Rightarrow MSE\_Q^{\pi_t} = 0$. If any of these two measures is zero, we could say that the RL-agent has learned the optimal policy, so its total average reward must approximate the optimal average reward. Note that the total average reward will not be exactly equal to the optimal average reward because we measure it since the beginning of the learning process, when the RL-agent still has not learned much, and it should be doing some exploration.

3.4.1 A small case

We consider a simple instance with 3 type of orders. Table 3.2 shows the problem definition.

According to the prototype model in Section 3.2 the states of the model are defined by the type of order arriving when the server is idle. At each decision moment the possible actions are to accept or to reject the arriving order. The goal is to find a deterministic policy that maximizes the total expected discounted reward with discount factor $\gamma = 0.9$. Using the complete model information the optimal policy is easily found by policy iteration. Table

order type	probability of arrival	processing time	reward
1	0.4	2	3
2	0.5	4	2
3	0.1	5	8

Table 3.2: *Data set for a small case on the prototype problem*

3.3 shows the optimal policy π^* and the corresponding Q^*-values. In this case the optimal policy π^* rejects an arriving order only if it is of type 2.

state	Q*(state,reject)	Q*(state,accept)	π^*(state)
1	10.78	12.70	1
2	10.78	9.86	0
3	10.78	15.07	1

Table 3.3: *Optimal Q-values and policy for the small case of the simple OA problem*

An order-greedy policy π' that accepts all the orders has a mean square error in the Q-values compared to the optimal Q-values (MSE-Q$^{\pi'}$) of 1.63, see Table 3.4.

state	order-greedy policy π' (always accept)	Q$^{\pi'}$(s,0)	Q$^{\pi'}$(s,1)	greedy Q-values policy
1	1	9.35	11.41	1
2	1	9.35	8.81	0
3	1	9.35	14.13	1

Table 3.4: *Example in the small case of the simple OA problem where non-optimal Q-values lead to the optimal policy by taking the action with the highest Q value*

Notice that in the case the Q-values for policy π' would have been learned at iteration t, looking at the greedy policy with respect to these values ($\pi_t(s) = \arg\max_a Q^{\pi'}(s,a)$) one obtains the optimal policy $\pi_t = \pi^*$ which rejects only orders of type 2. This illustrate a situation where MSE_Q_t ($= 1.63$) differs from $MSE\_Q^{\pi_t}$ ($= 0$).

Figure 3.1 shows the total average reward per unit of time of two different agents that follow these two policies. The "always accept" agent achieves an average total reward of 0.92 and the optimal agent of 1.11. We use these agents to compare with the results of agents that use other learning methods presented here.

Next, we present different approaches to solve this problem with incomplete information. First as a preparation, the naive model-based approach and the

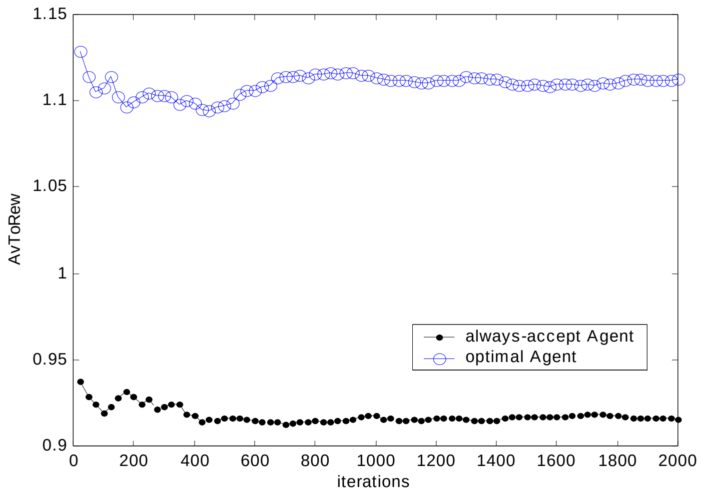


Figure 3.1: *Total average reward of two agents following fixed policies: the* optimal *policy and the* "always accept" *policy, in the small case of the prototype model*

backup-table method are discussed, next we consider neural networks knowledge representation methods. Finally we present a comparison of some of the best learning agents including one using the Dyna learning method.

A two-phase method: naive model based approach

Here at each iteration the agent updates the frequency of arrival of the arriving order and the respective reward value depending on the decision. The estimated model is solved using policy iteration. Due to the structure of the model this learning process is independent of the followed policy.

The results show that 50 iterations were sufficient to learn the optimal policy, $MSE\_Q^{\pi_t} = 0$ with $MSE\_Q_t \rightarrow 0,\ t \geq 50$.

QL results with backup tables

Here we present experiments to analyze the influence of the parameter settings in Q-learning using backup table as the knowledge structure. For this small case it is very easy to use Q-learning with backup table, there are only 6 Q-values to learn. A first study shows the results of using different learning and exploration parameters with the bootstrapping parameter λ set to zero. In a second study we use fixed learning and exploration parameters corresponding with the best performance from the first study, in order to analyze the influence of the bootstrapping parameter.

Study 1: influence of learning and exploration parameters. We use some representative combinations of learning and exploration parameters. The parameters correspond to the *learning schedule* as described in Section 2.3.2. This schedule defines the number of iterations of the training process (T), the initial value of the learning and exploration rates (α_0, ϵ_0), and the parameters for the learning and exploration rates-decreasing functions (T_α, T_ϵ). These parameters are chosen through experimental experience and are by no means optimized.

Parameters α_t and ϵ_t are the learning and exploration parameter at iteration t during the learning process. We use curve decreasing functions for both parameters as follows

$$\epsilon_t = \frac{\epsilon_0}{1+\frac{t}{T_\epsilon}}, \quad \alpha_t = \frac{\alpha_0}{1+\frac{t}{T_\alpha}} \ ,$$

where ϵ_0 and α_0 are initial values, T_ϵ and T_α respectively define the decreasing speed for both parameters. Note that the parameters drop to half its original value when $t = T_\epsilon, T_\alpha$ respectively.

The results from the two-phases method indicate that approximately 50 iterations were necessary in order to estimate the arrival rates accurately enough such that a good estimate of the Q-values leads to the optimal policy. This strongly suggests that we should keep a certain level of exploration in our Q-learning, till such sufficiently accurate estimates are possible.

In the experiments we explore how initial and speeding parameters should be set. After some trial and error experimentation we choose the following combinations of parameters. We use $T = 2000$, $\epsilon_0 = 1$, $T_\epsilon = 100$ and $\alpha_0 = 0.5, 1$ with $T_\alpha = 200, 100, 50$, respectively. Besides we add another value for T_ϵ ($T_\epsilon = 200$) in order to assess our choice for $T_\epsilon = 100$. Summarizing, we use the 7 combinations of parameters as shown in Table 3.5.

combination	1	2	3	4	5	6	7
T_ϵ	100						200
α_0	0.5			1			
T_α	200	100	50	200	100	50	100

Table 3.5: *The 7 combinations of parameters for a first study in the small case of the simple model*

Figures 3.2, 3.3, 3.4 and 3.5 show results from the performance measures MSE-Q, MSE-Q$^\pi$ and AvToRew for these 7 different learning schedules. Each graph represents the average results over 10 independent simulation runs (replications). Figure 3.2 represents the MSE-Q curves for the first six combinations. Combination 2 with $T_\alpha = 100$ leads to the best MSE-Q performance for $\alpha_0 = 0.5$. In case $\alpha_0 = 1$ we find even better results, again for $T_\alpha = 100$ in combination 5. These two combinations (2 and 5) also achieve the best average reward, see Figure 3.3.

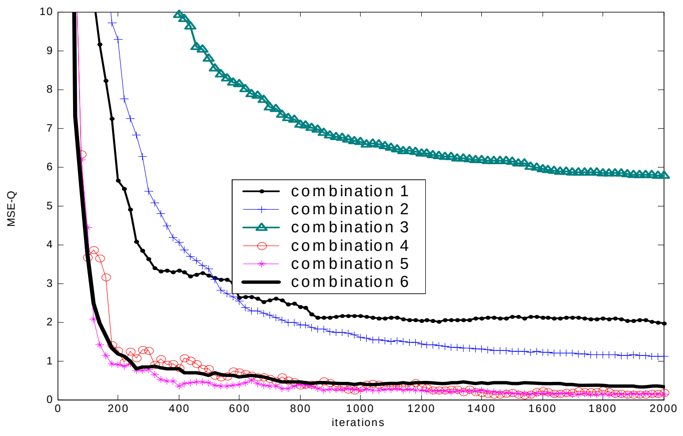


Figure 3.2: *MSE-Q performance in the small case of the simple OA model using six different learning schedules. All combinations use same exploration parameter. The first 3 use* $\alpha_0 = 0.5$ *and* $T_\alpha = 200, 100, 50$*. The last 3 use* $\alpha_0 = 1$ *and* $T_\alpha = 200, 100, 50$

The results from the MSE-Q$^\pi$ performance in Figure 3.4 match pretty well with the results of the MSE-Q performance. It shows that each combination after a hectic start shows rather stable behavior after learning certain optimal policy. Combinations 1 and 3 never learned the optimal policy (MSE-Q$^{\pi_t} \neq 0, \forall t$). Although for combinations 4 and 6 MSE-Q$^{\pi_t}$ achieves the value zero, it is not as stable as in in combinations 3 and 5. These two combinations (3 and 5) seem to be the best options. However, according to the results of the MSE-Q performance in Figure 3.2, and the AvToRew performance in Figure 3.3, we have a preference for combination 5.

Next we analyze whether increasing exploration could improve the results. Figure 3.5 shows the performances for the combinations 5 and 7. The combination 7 uses more exploration than the combination 5. The results show that both cases learn an optimal policy. Note however that the RL-agent follows an exploration policy (ϵ-greedy) instead of the learned policy. This could explain why combination 5 achieves better average reward. So in this case increasing exploration deteriorates the AvToRew performance. These results show the advantage of using no more exploration than necessary.

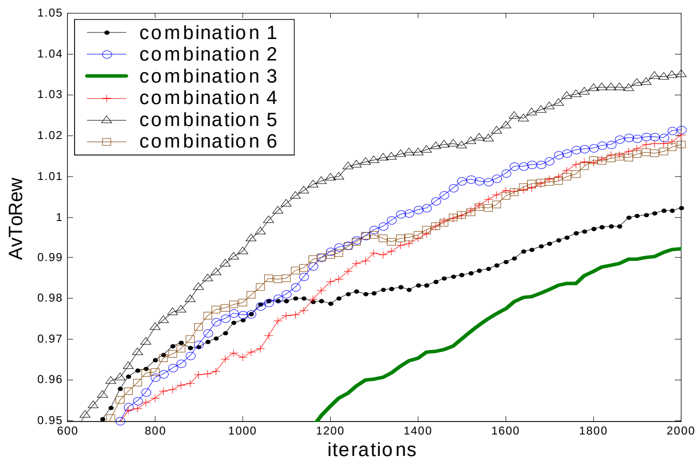


Figure 3.3: *AvToRew performance in the small case of the simple OA model using six different learning schedules. All combinations use same exploration parameters. The first 3 use* $\alpha_0 = 0.5$ *and* $T_\alpha = 200, 100, 50$*. The last 3 use* $\alpha_0 = 1$ *and* $T_\alpha = 200, 100, 50$

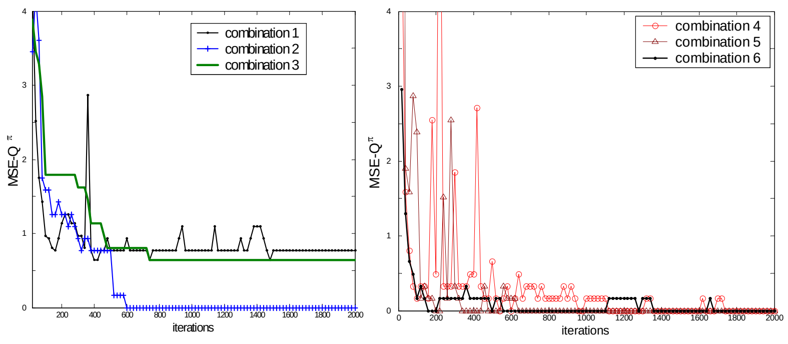


Figure 3.4: *MSE-*Q^π *performance in the small case of the of the simple OA model using six different learning schedules. All combinations use same exploration parameter (*$T_e = 100$*). The first 3 (on the left) use* $\alpha_0 = 0.5$ *and* $T_\alpha = 200, 100, 50$*. The last 3 (on the right) use* $\alpha_0 = 1$ *and* $T_\alpha = 200, 100, 50$

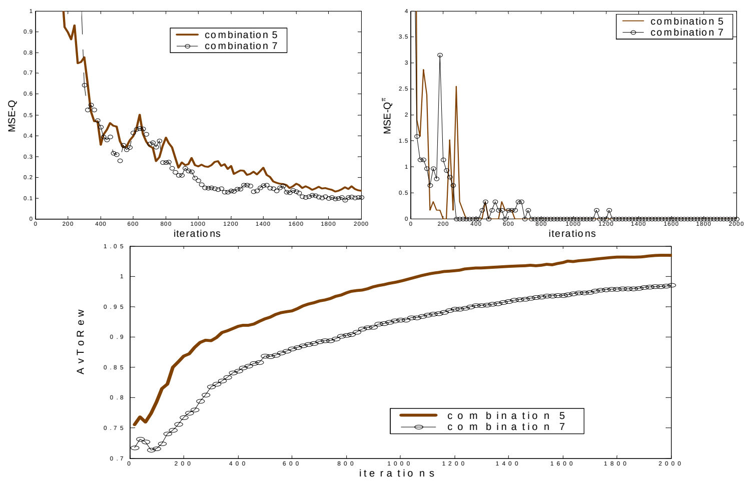


Figure 3.5: *Performance in the small case of the simple OA model using two different learning schedules. Both combinations use same learning parameters ($\alpha_0 = 1, T_\alpha = 100$) but different exploration parameters ($T_\epsilon = 100, 200$)*

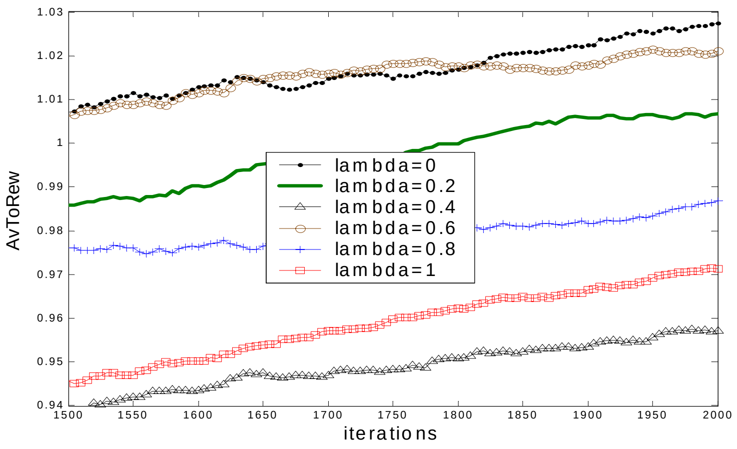


Figure 3.6: *AvToRew performance in the small case of the simple OA model using different values for λ*

Study 2: influence of the bootstrapping parameter λ. Using the combination 5 of parameters ($T = 2000, \epsilon_0 = 1, T_\alpha = T_\epsilon = 100, \alpha_0 = 1$) from the first study, we analyze the influence of different values of the bootstrapping parameter λ $(0, 0.2, 0.4, 0.6, 0.8, 1)$ in the learning process. Bootstrapping introducing $\lambda > 0$, does not help to improve the results, see Figure 3.6. This result support our choice for using $\lambda = 0$.

QL experimental results with neural networks

In this section we present some results for this small case using a neural network approach as explained in Section 3.3.1. As with the backup tables, we need a parameter setting. Whereas previously $\alpha_0 = 1$ was a good choice, it turns out that now working with $\alpha_0 = 0.1$ ($T_\alpha = T_\epsilon = 100$) is a better choice. This smaller value for α_0 compensates for the fact that now by updating the weights w in the perceptron, all Q-values are changed and not a single value as in the backup table method. Some tests on parameter settings were necessary to find out that these settings give an acceptable solution, but we shall not report on that here. Instead we shall focus on the effects of the knowledge structure, mainly the number of hidden neurons. Figures 3.7, 3.8 and 3.9 show the three performance measures previously defined, (MSE-Q, MSE-Q^π and AvToRew) for 5 RL-agents that use different number of hidden neurons $(1, 2, 3, 4, 5\ )$.

The figures show that in general the quality of the results increases with the number of hidden neurons. We want to emphasize that this is rather unexpected, since as discussed in Section 3.3.1, already for one hidden neuron the number of parameters in the NN exceeds the number of Q-values in this simple case. A "decreasing rate of return" effect for these excess parameters would be reasonable in the long run. It is an interesting observation that the ANN is capable to exploit the excess degrees in freedom for finding a more efficient route to good approximations. Notice that except for the case of one hidden neuron, the rest of the cases show an average total reward superior to the "always accept" policy as depicted in Figure 3.1.

An experiment was made with different values of the parameter λ (0, 0.1, 0.2,...,1). In these cases the same λ value was used through each run. Again there was no significance difference in the quality of the learned Q-values, using different values of λ, but values in the range 0.3 to 0.5 speeds up the learning process at early stage.

Final Comparison

Figure 3.10 shows a final comparison between the total average reward for different agents. All the learning agents outperformed the greedy policy and have a tendency to approximate the optimal one. Dyna and Naive that incorporate model base ideas have a faster learning than the direct RL.

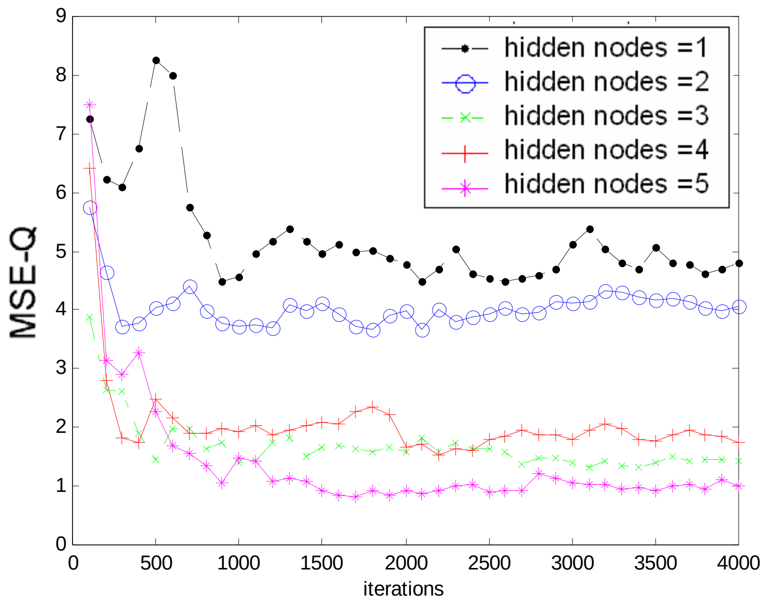


Figure 3.7: *MSE-Q performance of ordinary Q-learning for different sizes of the ANN in the small case of the simple model*

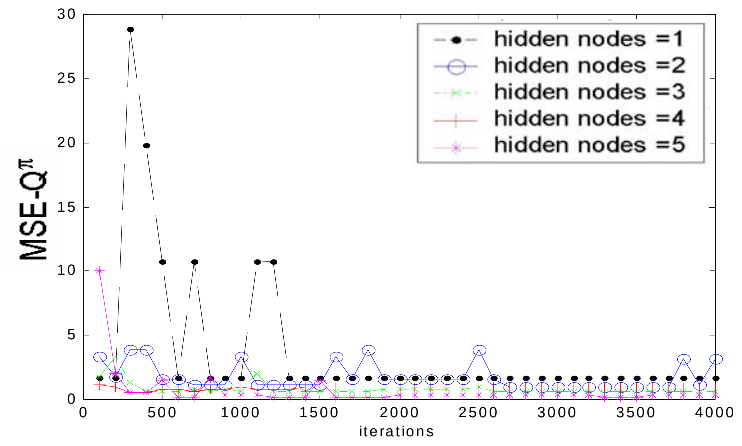


Figure 3.8: *MSE-Q^π performance of ordinary Q-learning for different sizes of the ANN in the small case of the prototype model*

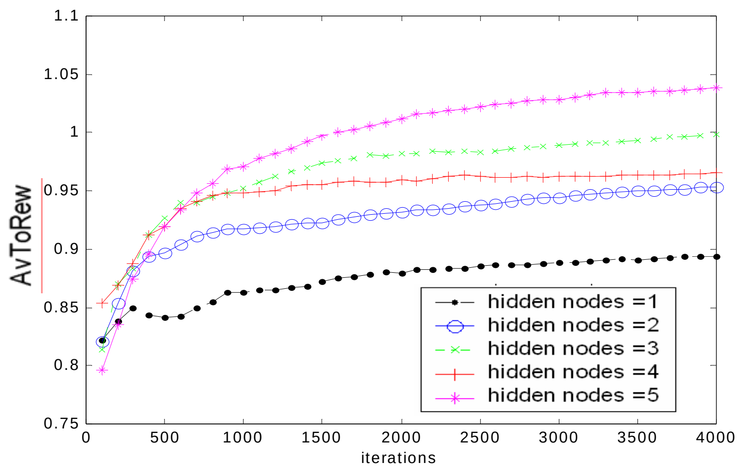


Figure 3.9: *AvToRew performance of ordinary Q-learning for different sizes of the ANN in the small case of the prototype model*

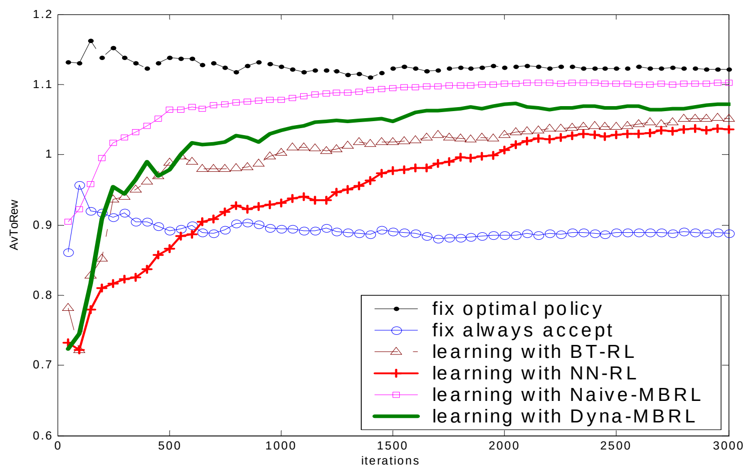


Figure 3.10: *Agents comparison in the small case of the simple model*

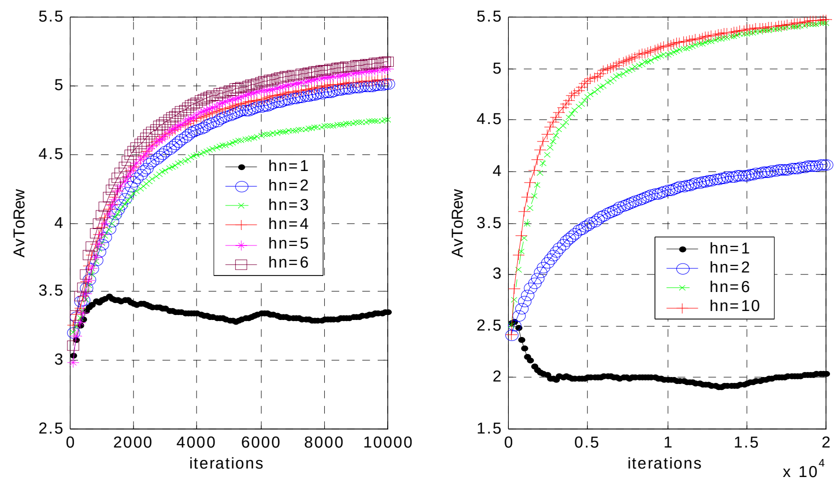


Figure 3.11: *Different RL-Agents in problems with 5 (left graphic) and 10 (right graphic) type of orders*

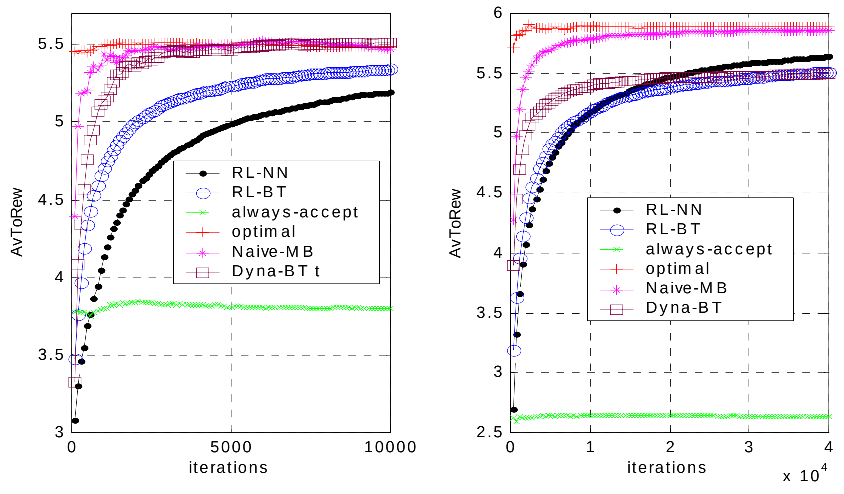


Figure 3.12: *Agent comparison in problems with 5 (left graphic) and 10 (right graphic) type of orders*

3.4.2 Larger cases

In this section we present experimental results of Q-learning using ANN in problems with 5 and 10 orders. The learning and exploration rates for both problems were $\alpha_k = 5/(500 + k), \epsilon_k = 200/(200 + k)$.

Figure 3.11 shows the comparison between agents using different numbers of hidden neurons. The results show that using one hidden neuron gives no good results but from 2 hidden neurons on results get better. For the case with one hidden neuron the ANN loses the generalization capabilities and is not very flexible.

Figure 3.12 shows the comparison of 4 learning agents (RL-NN,RL-BT, Naive-MB, Dyna-MB) with 2 agents using fixed policies ('always accept ', optimal). As in the small case, all the learning methods outperformed the greedy policy 'always accept' and have a tendency to approximate the optimal one. Again as expected the model based ideas speed up the learning process.

3.5 Conclusions

As a simple example, the prototype model helped us as an introduction to the application of the RL approach. In the experiments, backup tables were less sensitive to parameter settings than the neural networks as the knowledge structure for the RL-agents. Including model based ideas is also a trivial way for speeding up the learning process but it increases the computational time per iteration step. No relevant benefits were found in the final performances by using a parameter λ different from zero and constant during the learning process. Increasing the number of hidden neurons generally leads to an increment of the performance. In all the cases the RL-agents outperform the simple heuristic rule *always accept* and approximate the optimal policy. In the next chapters we apply the RL approach to more complex OA problems.

Chapter 4

OA in a single resource

In this chapter we present a simple Order Acceptance problem that is an extension of the problem presented in Chapter 3. The basic difference with respect to the orders is that we consider here a larger attribute set and with respect to the capacity we consider here a unique resource which may consist of several parallel servers that may simultaneously handle more than one order. In Section 4.1 we give a detailed description of the problem, then in Section 4.2 it is modelled as an SMDP.

We define in this chapter a general class of heuristics for OA problems which is presented in Section 4.3.

In Section 4.4 we discuss the application of RL to this problem.The policy learned by the RL-agent is implicit in the weights of the ANN, which works as a black box system not easily interpreted by human reasoning. Therefore we define in this chapter a general framework to interpret what the RL-agent learns providing an explanation in a for humans more comprehensible form.

The experimental results are shown in Section 4.5. Finally in Section 4.6 we draw some conclusions and set guidelines for the more complex cases presented in the next chapters.

4.1 Problem description

Let us now describe the OA situation in detail. Arrivals of orders take place at any continuous time moment. However, arrivals are collected and are only processed at discrete equidistant time moments. The time moments when the collected arrivals are processed, are the decision moments in which orders are rejected or chosen for service. So we consider aggregate batch arrivals, i.e., several arrivals of each order type during the previous time interval.

Each order asks for service on a shared resource, which can process different orders at the same time (i.e., concurrency is allowed). Order acceptance has to be planned over a planning time horizon with H stages. Each stage spans the time between two decision epochs. There is a maximum capacity $C_{\max}$ for every stage that can only be excessed at the cost of a penalty as we explain below (Equation 4.1). The concept of capacity is filled in in a specific way, namely in time units of processing times. The vector c describes the capacity utilization. By c_t we refer to the occupied capacity at stage t of the planning horizon, due to orders in execution accepted previously. Figure 4.1 shows an example of capacity utilization in a planning horizon with $H = 10$ stages and $C_{\max} = 8$ where $c = (4, 2, 8, 7, 5, 2, 4, 8, 4, 8)$.

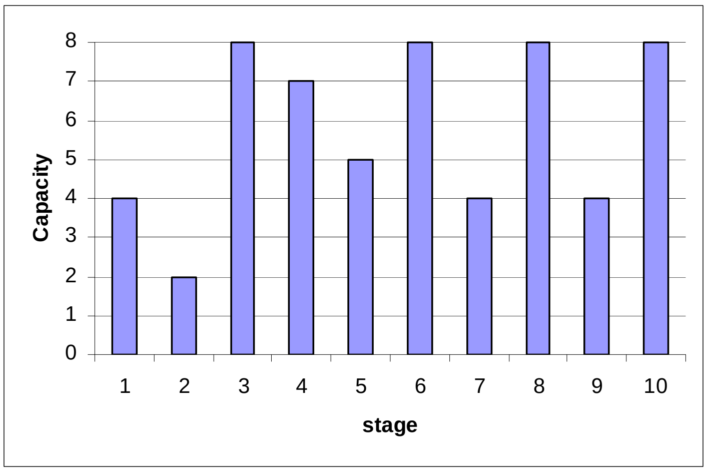


Figure 4.1: *Capacity utilization in a planning horizon with* $H = 10$ *stages and* $C_{\max} = 8$ *where* $c = (4, 2, 8, 7, 5, 8, 4, 8, 4, 8)$

In this chapter we want to consider a larger order's attribute set than in the previous chapter, specifically we include the due-date of the order. But characterizing orders according to their due-dates could create a problem with respect to the amount of different types of orders. For this reason we consider here an analogous attribute: due-time, which is a relative concept with respect to the planning time horizon. The due-time of an order is the time from the first decision moment after the order's arrival until its due-date. For example suppose decision moments are at every one hour, if in a situation like in Figure 4.1 at the stage 1 of the planning horizon we are at time 7.00 hours and the due-date of an order is at 15.00, its due-time is then 9, which indicates that the order should be planned not after stage 9 of the planning horizon.

In general order definitions are based on a finite number n of order types, where type i has an arrival rate (λ_i), due-time (t_i), expected processing time

(w_i) and generates an immediate reward (r_i) upon acceptance.

Order processing requires an order type dependent expected processing time w_i. We allow for uncertainty due to realization differing from the expectation, causing a possible capacity perturbation. For simplicity we consider a random perturbation term $p \in \{-1, 0, 1\}$ which affects the capacity profile c_t only for $t = 1$ with probability $\Pr(p)$. If $c_1 > 0$, then in the realization during the next period the actually required capacity turns out to be $c_1 + p$ instead of the anticipated $c_1 > 0$. If extra capacity is used in realization ($p = 1$) and no regular capacity is available, we introduce a penalty $pen(c, p)$ for using non-regular capacity as follows:

$$pen(c,p) = \begin{cases} -\eta & \text{when } c_1 + p > C_{\max} \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

with $\eta > 0$.

When the available capacity is not sufficient for an arriving order, given its capacity request and its due-date, the only option is rejection. We allow preemption but we do not allow late delivery. Further decision postponement, by putting an order in queue till a next decision, is not allowed. At each decision epoch we assume that arriving orders, occurring after the previous decision epoch, are accumulated into a list. Then, a decision has to be taken about which subset of the orders requesting service will be accepted. In principle each subset of orders should be analyzed, to see whether it fits into the available capacity in such a way that the due-dates are not violated. In order to reduce the number of possible decisions to a polynomial size in the number of order types we impose some restrictions on the structure of the decision rule. Instead of focussing on all possible subsets of orders at once, as possible decisions, we impose that the decision is created sequentially, while we call a *time off*. Each single decision i in the sequence is either the selection of one of the orders from the list ($i \in [1..n]$) or the rejection of all of them ($i = 0$). By definition we put $i = 0$, also if the remaining list is empty. If the available capacity is not enough for an order, then the selection of that order is not an option. If $i > 0$, then capacity is allocated to order type i and the list of the remaining orders is updated. If $i = 0$, we go to the next decision epoch with a *time on* and a list of newly arriving orders.

For the capacity planning we shall use a fixed prescription. Therefore optimizing the capacity planning is not an issue. The non-allocated capacity is available for executing new orders. The processing time for an order can be freely allocated within the available capacity before its due-date. Essential is how the capacity profile is updated given the decision i (I) of accepting an order of type i ($i > 0$), or (II) rejecting the complete job list ($i = 0$) which as a special case always occurs if the job list is empty. For (I) and (II) capacity allocation will be done following different planning rules. (I) and (II) are different from a planning perspective.

In case (I) we choose for a Backward Loading (BL) principle, since allocating capacity as close to the due-date as possible in this model, provides the best conditions for accepting more orders from the order list. Algorithm 4.1 sketches this procedure[1].

```
Require: $c$ and $i$
  $t \leftarrow t_i$
  $requested\_capacity \leftarrow w_i$
  $c'(t+1:H) \leftarrow c(t+1:H)$
  while $t \geq 1$ and $requested\_capacity \neq 0$ do
    $\xi \leftarrow \min(requested\_capacity, Cmax - c_t)$
    $c'_t \leftarrow c_t + \xi$
    $requested\_capacity \leftarrow requested\_capacity - \xi$
    $t \leftarrow t - 1$
  if $requested\_capacity \neq 0$ then
    $return($ not possible $)$
  else
    $return(c')$
```

Algorithm 4.1: *BL (c,i): Backward loading procedure for job type i in a capacity profile c*

For example if in a situation like in Figure 4.1 a new order is accepted with a due-time 8 and a capacity requirement of 5 units, it is allowed to plan the execution of that order with BL, providing 4 capacity units more at stage 7 since stage 8 is already fulfilled, and 1 capacity unit more at stage 5 (stage 8 is already fulfilled), see Figure 4.2. Note from the figure that the new order is allocated in stages 5 and 7 of the planning horizon while there is a different order allocated in the stage 6, in this sense preemption is allowed.

Case (II) is completely different. If there is still available capacity in the first stage ($t = 1$), then it is lost as soon as we enter the next stage unless we adapt the allocation. We may create the best conditions for accepting orders from the new list at the next decision epoch, if we fill the still available capacity at $t = 1$ according to a Least Shift Back (LSB) principle. This boils down to looking for already allocated capacity forward in time, starting at $t = 2$, which is replanned to $t = 1$ until the still available capacity at $t = 1$ is filled as much as possible. Algorithm 4.2 sketches this procedure which also takes into account the capacity perturbation p that may occur at the first stage of the planning horizon.

For example if in a situation like in Figure 4.2 no other arrival is present, we finish the acceptance of orders, and there still are 4 units of capacity available at the first stage. Suppose there is a capacity perturbation on stage 1 of 1 unit

[1]The notation $x(t : H)$ refers to the sub-array of x, from the $t - th$ element up to the $H - th$ one.

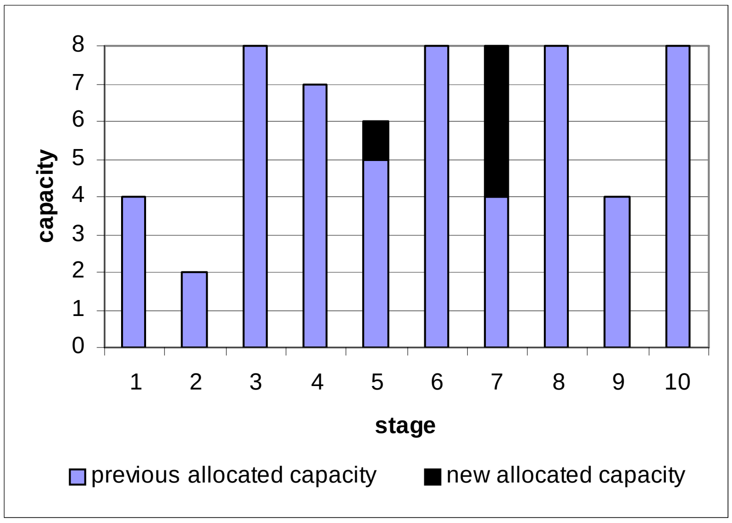


Figure 4.2: *Allocate capacity = 5, due-time = 8*

```
**Require:** $c$ and $p$
  $available\_capacity \leftarrow \max(0, Cmax - \max(0, c_1 + p))$
  $c_{H+1} \leftarrow 0$
  $t \leftarrow 2$
  **while** $t \leq H + 1$ and $available\_capacity > 0$ **do**
    $\theta \leftarrow \min(c_t, available\_capacity)$
    $c'_{t-1} \leftarrow c_t - \theta$
    $available\_capacity \leftarrow available\_capacity - \theta$
    $t \leftarrow t + 1$
  $c'(t-1 : H) \leftarrow c(t : H + 1)$
  $return(c')$
```

Algorithm 4.2: *LSBp(c,p): Least Shift Back plus capacity perturbation allocation procedure for capacity profile c and perturbation p*

of capacity ($p = 1$), then there are 3 units of capacity available (instead of 4), so the 2 units at the second stage and one unit of the third stage can be shifted back to the first stage in order not to lose that capacity, see Figure 4.3.

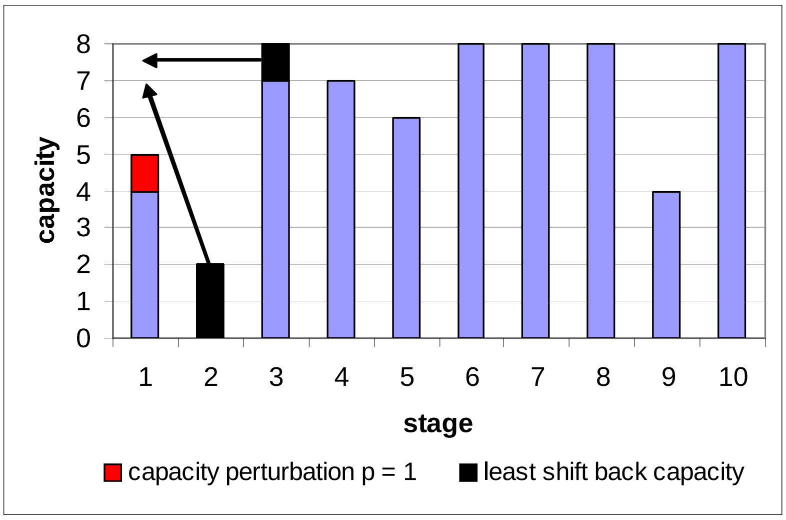


Figure 4.3: *There is a capacity perturbation $p = 1$ at the first stage of the planning horizon. The arrows indicate the capacity that will be shifted back to the first stage in order not to lose it*

Next there is a *time on*, new arrivals will be collected, and the planning horizon is moved one stage ahead as it is shown in Figure 4.4.

This approach is not a limitation for the decision policy. We conjecture that choosing orders one by one, backward loading right after choosing and using LSB after all arrivals are processed, constitutes the best loading strategy for the order acceptance decision.

Cases of more realistic order acceptance problems may be studied. Nevertheless this case is rich enough to demonstrate the complications that may arise in finding an optimal decision policy.

In terms of the characterization (i)-(vi) given in Chapter 1 we have here the following:

(i) Order definitions are based on a finite number of types of orders, where each type has a specific expected processing time, immediate reward upon acceptance, and a due-time. Violation of due-times is not allowed. Orders may be split into segments that can be processed in any sequence.

(ii) Arrivals of orders take place continuously, however, arrivals are only evaluated at discrete equidistant time moments, so we consider several arrivals

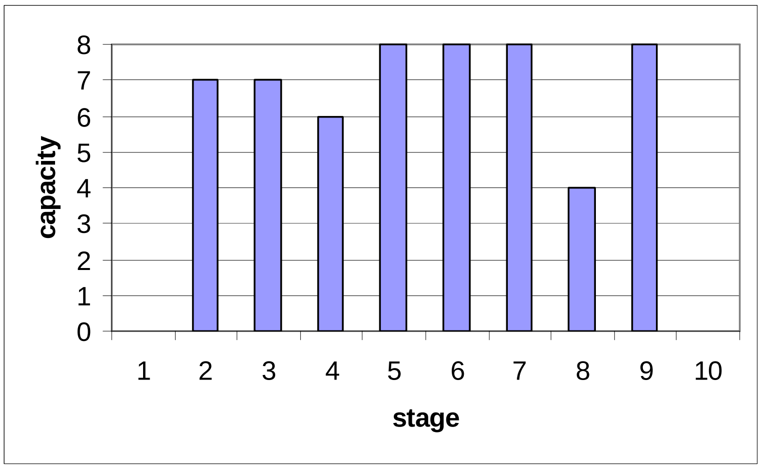


Figure 4.4: *Capacity profile at the beginning of the next planning horizon. Due to a previous LSB the first stage is empty*

in each discrete time unit (batch arrivals), and the probability of arrival is order type dependent.

(iii) Rejection of an order only has an effect on the capacity being free for other orders and does not affect any future arrivals.

(iv) Order processing requires an order type dependent service time from a shared resource, realization of an order may differ from the expectation causing a possible capacity perturbation.

(v) Capacity is considered as a unique resource, which may consist of several parallel-servers. Orders are loaded over a planning horizon. There is a maximum regular capacity. It is possible to use non-regular capacity at a certain cost, in order to avoid possible violations of due-times for the accepted orders that may be caused by the capacity perturbations.

(vi) The decision policy should choose a set of orders from the arrival batch: if there is no available capacity for an order, the only option is rejection for that order.

4.2 The SMDP model

We now formally describe this problem as a semi-Markov decision problem (SMDP). The decision moments occur during a *time off* and an action may be selecting one order from the order list, or rejecting the remaining order list. Note that in this way the action space is reduced considerably compared with what one should expect at first. Originally, each subset of orders from the list that fits into the open capacity is a potential action. In this reduction the number of actions is limited to $n+1$ possibilities. However, an extra iteration procedure with *time-off* has to be introduced to facilitate this. The state at each decision moment is defined by $s=(k,c)$ where:

$k=(k_1,...,k_n)$ is the order list and k_i represents the amount of orders of type i requesting service.

$c=(c_1,...,c_H)$ is the capacity profile and c_j is the total capacity already allocated in stage j.

The presence of a list with k_i orders of type i at a certain decision epoch has the following joint probability distribution

$$\Pr(k)=\Pr\{new\_orders=(k_1,......,k_n)\}=\frac{\lambda^k}{k!}\exp(-\Lambda)$$

where we use the shorthand notation $\lambda^k=\Pi\lambda_i^{k_i}$, $k!=\Pi k_i!$ and $\Lambda=\Sigma\lambda_i$.

For each order type i we restrict the maximum amount of orders in the order list to m_i. This number may be determined by the arrival probabilities or by the limited capacity. The state space S is the set of all possible states and its cardinality is $(C_{max}+1)^H\prod_{i=1}^{n}(m_i+1)$ in the general case.

The action space is $A=\{0,1,..,n\}$. The set of allowed actions $A(s)$ for a state $s=(k,c)$ is defined as follows. Action $i\in[1..n]$ is allowed if order type i is present in the order list and capacity is available for at least one order of type i, rejection $(i=0)$ of the complete order list is always an option (i.e., $A(s)=\{i\in[1..n]\ |k_i\neq 0, BL(c,i)\text{ is possible}\}\bigcup\{0\}$).

The one step state transition from current state s is described in Table 4.1. The first column represents all the characteristics defining a state transition from the current state s. The other two columns represent these characteristics given that the action i in the current state is different from rejection (second column), and when the action is rejection (third column).

current state (s)	(k,c)	
action (i)	$i\in A(s),\ i\neq 0$	0
next state (s')	$(k-e_i, BL(c,i))$	$(k',\ LSBp(c,p))$
$d(s,i)$	0	1
$rew(s,i,s')$	r_i	$pen(c,p)$
$Pr(s,i,s')$	1	$\Pr(k')\Pr(p)$

Table 4.1: *Dynamics of transition from current state s in the shared resource model.*

In case an order of type i is chosen $(i\neq 0)$, an immediate reward r_i is received and a transition to the next state s' occurs deterministically $(Pr(s,i,s')=$

1) with the *time off*, i.e., the elapsed time $d(s,i) = 0$. The order is then removed from the order list so the new order list is given by $k - e_i$ where e_i is a vector with 1 in position i and zeros otherwise. The capacity profile is updated with the procedure $BL(c,i)$ as described in Section 4.1.

In case the order list is rejected ($i = 0$) the immediate reward is stochastically dependent on the capacity perturbation (p) and is given by the penalization function (Formula 4.1). Now we have a *time on* situation with the elapsed time $d(s,0) = 1$ and we are in situation (II), where a new list of arriving orders is considered. The new arrivals vector k' is determined by the order arrival process with statistics as described before and the function $LSBp(c,p)$ updates the capacity profile given the current capacity profile c and the perturbation term p, so the transition probability depends on the arrival process and on the capacity perturbation.

The objective is to find a deterministic policy π:

$$\pi(s) = \begin{cases} i & i \in \overline{1...n} \text{ select order type } i \\ 0 & \text{reject the arrival-list,} \end{cases}$$

which maximizes the performance of the system. The performance of the system is measured as the expected value of the total discounted reward. The corresponding Bellman equation for the state value function V^π is given by:

$$V^\pi(s) = \sum_{s'} Pr(s,\pi(s),s') \left[rew(s,\pi(s),s') + \gamma^{d(s,\pi(s))} V^\pi(s') \right], \qquad (4.2)$$

where γ is the discount factor and $d(s,i)$ is the elapsed time as introduced before. The optimal action value function Q^* in this case satisfies:

$$Q^*(s,i) = \sum_{s'} Pr(s,i,s') \left[rew(s,i,s') + \gamma^{d(s,i)} \max_{i'} Q^*(s',i') \right]. \qquad (4.3)$$

Having Q^*, an optimal policy $\pi^*(s)$ can be determined by choosing the action $i \in A(s)$ that maximizes $Q^*(s,i)$:

$$\pi^*(s) = \arg \max_{i \in A(s)} Q^*(s,i). \qquad (4.4)$$

In this problem even in the case that the parameters of the model are known (exactly or by some statistical estimation) solving the problem is still a difficult task due to what is usually referred to as the "curse of dimensionality". It is known that finding an optimal policy requires an overwhelming computational effort if the dimension of the state space increases. Here the state space may be tremendously large. As a consequence $\pi^*(s)$ may be a complex rule. A simple structure "*accept if capacity is available and the reward per unit of processing time is sufficiently high*" as found in the prototype problem in Chapter 3 may no longer be expected for the general case. What one should expect is that, in case of low utilization of the capacity, an optimal decision rule will be inclined to accept more orders which generate low profit per unit of processing time than in case of high utilization. Further, one should expect that even in case of high utilization certain small rush orders, that fit well into a gap in the capacity profile, are still attractive for acceptance, also if they are not so profitable, simply because the gap is too small to accommodate other orders. An interesting aspect of this research is to see up to what degree the learning approach leads to an approximate decision rule representing such effects. In Section 4.3 we introduce a general class of heuristics that is used for reasons of comparison but also to help interpreting the policy learned by the RL-agent as we explain in Section 4.4.1.

4.3 A general class of heuristics

Recall that we defined for each state s in S an associated decision set $A(s)$ of all allowed actions when in state s. However, this set $A(s)$ is very general. To facilitate the deduction of heuristic rules from state-action pairs properties we introduce the following two concepts. Firstly, we consider special subsets $A^k(s)$ of $A(s)$ which display certain measures like quality of orders or capacity assurance. For example, motivated by one of the following considerations:

- orders with rewards under a certain threshold may always be rejected, so choosing them is not an option,
- if the total capacity is occupied up to a certain level certain orders may be rejected.

Secondly, we consider a partition of the state space S, $S = \bigcup_{j=1}^{N} S_j$, with $S_j \bigcap S_i = \emptyset$, $j \neq i$. For each S_j we define $A^k(S_j) = \bigcup_{s \in S_j} A^k(s)$. On each $A^k(S_j)$ we introduce a linear preference relation $\succ$ that defines a preference relation ϕ_j of all the actions in the set. We denote with i_l the action in position l with respect to this preference relation: $\phi_j = (i_1, i_2, ..., i_n, i_{n+1})$ means that $i_1 \succ i_2 \succ ... \succ i_n \succ i_{n+1}$ so order type i_1 is preferred over order type i_2, etc. A heuristic policy π can be defined that assigns to each state $s \in S_j$, $j = 1...N$, an allowed action from $A^k(s)$ considering the defined preference relation, as follows:

$$\pi(s) = \max_{\phi_j} A^k(s), \tag{4.5}$$

i.e., $\pi(s)$ is the maximum action from all the actions in $A^k(s)$ according to the ordering ϕ_j. Note that the given structure of the heuristic is very powerful. On one hand the heuristic is completely defined by the subsets $A^k(s)$, the partition $\{S_j\}$, the orderings $\{\phi_j\}$ and the expression (4.5). On the other hand any policy π can be represented in this form by defining unitary subsets $A^k(s) = \{\pi(s)\}$, in which case the ordering and the partition of the state space do not matter. This means that the given structure is a characterization of MDP policies. The RL trained agent policy can be represented in this way by putting $A^k(s) = \left\{\arg\max_a Q(s,a)\right\}$ and again the ordering and the partition of the state space are not important. But, of course, finding the subsets A^k and the partition of the state space is just as difficult as solving the Markov decision problem, with its curse of dimensionality.

To illustrate the flexibility of this approach for describing (and as we discuss in Section 4.4.1, finding) heuristics we consider some useful examples constructed by relating the ordering and subsets to relevant criteria, such as the reward per requested unit of capacity for a job (i.e., $\frac{rj}{wj}$ for job type j) and the utilization defined as the occupied capacity as a percentage of the full capacity. These examples do not consider the partition of the state space (for a partition example see e.g., the set of rules in Section 4.5.1 for case 1). An example of this sort is the optimal policy for the simple OA problem in Chapter 3.

Heuristic: Order Quality

- allowed actions for each state $s \in S$:

$$A^1(s) = \left\{ i \in A(s)\backslash\{0\} | \tfrac{r_i}{w_i} \geq b \ and \ \ cap(s,i) \leq HC_{\max} \right\} \bigcup \{0\}$$

- ordering for all actions in A:

$$i_{n+1} = 0, \ i \succ j \text{ if } \left(\tfrac{r_i}{w_i} \geq \tfrac{rj}{wj}\right) \ or \ \left(\tfrac{r_i}{w_i} = \tfrac{rj}{wj} \ and \ r_i > r_j\right).$$

Here $cap(s,i)$ denotes the total occupied capacity after accepting an order of type i when in state s. $HC_{\max}$ is the total capacity in the planning horizon. According to this policy only orders with a reward per processing time above a threshold b may be accepted. This threshold is related to the avoidance of opportunity losses. In our experiments this sort of heuristic will be referred to as $OrderQuality(b)$.

Improvements of this heuristic are possible. Note that, in case of penalization for excess capacity, it might be better to fill capacity only up to a certain level of utilization. So there is a safety margin for dealing with perturbations due to non-anticipated extra capacity demand during job execution. This remark leads us to requiring that after acceptance of an order of type i the capacity utilization of the total capacity in the planning horizon ($HC_{\max}$) is still under a capacity level threshold (ρ_i). This leads us to the following action subsets and ordering:

Heuristic: Capacity level

- allowed actions for each state $s \in S$:

$$A^2(s) = \{i \in A(s) \backslash \{0\} \mid cap(s,i) \leq \rho_i HC_{\max}\} \bigcup \{0\}$$

- ordering for all actions in A:

$$i_{n+1} = 0,\ i \succ j \text{ if } \left(\frac{r_i}{w_i} > \frac{rj}{wj}\right) \ or\ \left(\frac{r_i}{w_i} = \frac{rj}{wj}\ and\ r_i > r_j\right).$$

In our experiments this sort of heuristic will be referred to as $CapacityLevel(\rho)$. Note that the previous heuristic can be included here by using

$$\rho_i = \begin{cases} 0, & \frac{r_i}{w_i} < b \\ 1, & \frac{r_i}{w_i} \geq b \end{cases}.$$

Note that a good *CapacityLevel(ρ)* should incorporate information about the due-time t_i into ρ_i. Finding a good parameter vector ρ for these heuristics could be a difficult task. Hence we use the same parameter for all types of orders as an initial reference to evaluate the results of the RL-agents; these will be called one-parameter-heuristics. Then, using the policy learned by the RL-agents, we will try to identify better parameters. It would be interesting to see to what extent the RL-agent is able to find good parameters and even more, to find heuristic rules that fit in the more general class of heuristics that consider the partition of the state space. In the next section we present the RL-approach, with an explanation on how to interpret the policies learned by the RL-agents, and then we present some computational experiments.

4.4 Reinforcement Learning approach

Here we present the application of Reinforcement Learning to the order acceptance problem described before. We use QL methods that aim to approximate the optimal Q-value function as discussed in Chapter 2. The RL-agent should learn an OA-policy while interacting with the environment. We focus here on traditional Q-learning as introduced in Chapter 2.

The application of this method to our model is not as straightforward as in the previous model discussed in Chapter 3. Here the size of the state space does not only depend on the different types of orders, but also on the number of possible arrivals, the size of the planning horizon, and the maximum capacity.

The size of the action space depends on the number of order types, and not all actions being possible for each state. Here we can think of two variants.

1. The agent receives exact information about s and $A(s)$ according to the model description in Section 4.2. Together with the information about the state, the agent receives the environment information about the possible actions for that state. This implies that at each decision moment the possibility of capacity allocation in the current capacity profile, for each type of order present in the order list, has to be checked.

2. A change in the definition of $A(s)$ and consequently a change in the reward function and the transition to the next state. The agent is allowed to choose any of the present orders, either it fits in the capacity or not (i.e., $\widetilde{A}(s) = \{i \in [1..n] \; |k_i \neq 0\} \bigcup \{0\}$). Hence the original $A(s)$ is enlarged such that it is not necessary to check whether there is available capacity for each type of order present in the order list. In case the agent chooses an order that does not fit in the current capacity, the agent will be punished with a negative reward $(-\zeta, \, \zeta > 0)$. In this case a change has to be made in the environment reward function to consider this punishment term:

$$rew(s,i,s') = \begin{cases} r_i & \text{when } i \neq 0, \text{ and } BL(c,i) \text{ is possible,} \\ -\zeta & \text{when } i \neq 0, \text{ and } BL(c,i) \text{ is not possible,} \\ pen(c,p) & \text{when } i = 0. \end{cases}$$

   Another change is in the transition to the next state. In case the agent chooses an order that does not fit in the current capacity, the order list for the next state would not have such an order. Table 4.2 shows the new state transition.

current state (s)	(k,c)		
action (i)	$i \neq 0$,and $BL(c,i)$ is possible	$i \neq 0$, and $BL(c,i)$ is not possible	0
$rew(s,i,s')$	r_i	$-\zeta,$	$pen(c,p)$
$d(s,i)$	0	0	1
next state (s')	$(k-e_i, BL(c,i))$	$(k-k_ie_i, c)$	$(k', \; LSBp(c,p))$
$Pr(s,i,s')$	1	1	$\Pr(k')\Pr(p)$

Table 4.2: *State transition with extended $A(s)$ definition in the shared resource model*

In our approach we choose for the second variant since it is computationally more efficient for this model.

The behavior of the RL-agent is according to an $\epsilon-$greedy exploration rule as explained in Chapter 2: with probability $1-\epsilon$ the agent chooses a greedy action (the action a from $A(s)$ which maximizes $Q(s,a)$) and with probability ϵ a random action. The parameter ϵ decreases over time according to the following rule $\epsilon_t = \frac{\epsilon_0}{1+\frac{t}{T_\epsilon}}$.

As for the knowledge structure, the agent uses an $I-\theta-1$ perceptron to store the $Q-$values as discussed in Section 2.2.3. The input of the perceptron is a codification of the state-action pair (s,a), in the following way:

- n integer inputs showing the numbers of orders of each type $(k_1, ..., k_n)$.
- H integer inputs showing the already allocated capacity at each of the capacity profile stages $(c_1, ..., c_H)$.
- 1 integer input representing the action.

Using this codification for the input, the size of the ANN is $(n+H+3)\theta+1$ which is a much lower dimension than the state and action space has in general, $|S\text{x}A| = (n+1)(Cmax+1)^H \prod_{i=1}^{n}(m_i+1) \geq (n+1)2^{n+H}$. The output of the perceptron is a single value that is the approximation of the state-action value function $Q(s,a)$.

The goal is to maximize the expected total discounted reward. Again we use 0.9 as discount factor. It is reasonable to expect that the agent learns to avoid the negative rewards, i.e., to avoid the orders that do not fit in the capacity profile. For the parameters of the agent we use the *learning schedule* as described in Section 2.3.2.

4.4.1 Interpreting the RL-agent's policy

The policy learned by the RL-agent is implicit in the weights of the ANN and by means of these it can be applied as a decision maker, obtaining for a given state the corresponding action. However, it works as a black box system, not easily interpreted by human reasoning. Having this policy explicitly, mapping an action for each state, could computationally be very expensive since the size of $S\text{x}A$ grows exponentially with respect to the problem data. Furthermore, the learning policy could be affected by noises due to infrequent states and the use of function approximation. Therefore, a better alternative is to obtain the general rules that the RL-agent has learned over some subsets of states.

There has been some effort to provide an explanation in a for humans comprehensible form of the knowledge embedded in a trained ANN, by extracting rules that mimic the behaviour of the ANNs, see (Towell, 2000; Towell & Shavlik, 1993; Andrews et al., 1995; Tickle et al., 1998).

As stated in (Tickle et al., 1998), "no compelling evidence has emerged which mandates the use of a particular type of ANN architecture and/or a particular type of rule extraction technique in a given class of problem domains."

In our case we use a simple data mining technique to obtain a set of rules which match the behaviour of the trained RL-agent up to a certain degree that defines the quality of the rules. We use as the quality of a set of rules, the percentage of the data covered by the rules.

Based on a number of iterations using the trained RL-agent, we construct a descriptive decision-tree splitting the iterations. Recall that we can collect one state-action pair per iteration. Decision trees are one of the most popular techniques in data mining because they are easily built and understood, see (Han & Kamber, 2000). The basic idea is to make a partition of all the collected data into nodes, and for each node to find rules describing the data on the node. As we are using heuristics from the general class defined in Section 4.3 to compare with the RL-agent, we consider a preference relation on the type of orders a suitable structure to describe the RL-agent's policy at each node of the decision tree. The partition of the set of all the data can be done based on some characterization of the states, say for example the total amount of loaded capacity. A set of rules with the structure of the general class of heuristics discussed in Section 4.3, can be derived from the tree, i.e., a preference relation of the actions over subsets of the state space. What we look for is a set of rules like [if $s \epsilon S_j$ then use φ_j to decide an action for s].

We define the overall tree-error E of the set of rules defined by the decision tree as the number of iterations misclassified by the rules. We allow for a tolerance of τ in the error E. Consequently we may measure the error E^{φ} of each node with a tolerance of τ'. Algorithm 4.3 sketches the general framework we use here to interpret the RL-agent's policy.

The root node (zero level) represents all the iterations. If at that level, a quality criterion is met, we have a unique rule defined by the preference relation. Otherwise, a new level is added to the tree. For the OA problem considered in this chapter we choose the total occupied capacity as a branching variable. In our problems we generally split the root node according to possible values of the total occupied capacity C_T on the planning horizon ($C_T = \sum_{i=1}^{H} c_i$). There is a maximum of $HC_{max} + 1$ possible nodes at the first level. Again we define a preference relation at each node on the first level. Now the rules will be of type "IF $C_T = x$ THEN ϕ_x", where ϕ_x is the preference relation at the node corresponding to the value x of the total capacity C_T. If this set of rules meets the quality criterion we stop, otherwise a new level is added to the tree. The first level is split according to possible combinations in the distribution of capacity over the planning horizon, and the same procedure is repeated. If necessary a last third level can be added attending to possible combinations of number of arrivals per each type of order.

As in every branching method, it is not necessary to fully expand every level. We could choose to split the leaf representing the rule with the lowest quality every time. We could also choose not to split by single cases but by a

```
  Reinforcement Learning
  Perform the learning procedure and obtain a policy $\pi$
  For the learned policy $\pi$ perform a specific number of RL-iterations and
  collect for each iteration the state $s$, the set $A(s)$ of possible actions and the
  chosen action $a$ for that state

  Data Mining: building a decision tree
Require: Levels of tolerance $\tau, \tau'$
  Choose a preference relation $\phi$ over all actions in A
  Initialize a decision tree by taking into the root node all the RL-iterations.
  Consider the root node as the current node. $E \leftarrow \tau + 1$
  while $E > \tau$ and there are alive nodes do
    For each pair of actions $(a, a')$ determine the number of iterations in the
    current node where action $a$ was chosen when action $a'$ was also possible.
    Determine the error $E^{\phi}$ in the node
    Iteratively improve $E^{\phi}$ by swapping two actions in $\phi$
    Compute E
    if $E^{\phi} < \tau'$ then
      backup $\phi$, $E^{\phi}$ in the current node and kill the node
    else
      if $E > \tau$ then
        choose a model parameter and branch the node. To every new node
        assign $\phi$ as preference relation. Choose an alive node as the current
        node.

  Detection of heuristic: Extracting rules from the decision tree
  For each leaf (killed node) of the final decision tree formulate an if-then
  rule according to the settings in the leaf for the model parameter and the
  preference relation
```

Algorithm 4.3: *General framework to interpret the RL-agent's learned policy*

combination of cases (interval values for total capacity for example instead of single values). This is the same as combining similar nodes into one. Next we explain how to define the preference relation from a set of iterations.

Preference relation

Given a number of iterations, we want to define a preference relation ϕ among all the actions $\{0, 1, 2, ..., n\}$ aiming to minimize the number of iterations not covered by such a preference relation. The number of iterations not covered by the preference relation is called the error of the preference relation. We use an iterative approximate procedure which at every step reduces the error. To explain the procedure, a basic concept of "matrix of actions for a given preference relation ϕ", A_ϕ is first introduced.

The element a_{ij} of the matrix of actions A_ϕ for the preference relation ϕ, represents the number of iterations where the action i in ϕ was chosen when action j in ϕ was also possible, i.e., the number of times action i was preferred over action j. From this definition it follows that the elements under the diagonal (a_{ij}, $i > j$) represent violations of the preference relation. Note however that the sum over all these elements $\xi = \sum_{i=2}^{n+1} \sum_{j=1}^{i-1} a_{ij}$ is an upper bound on the total number of erroneous iterations E^ϕ. If the preference relation ϕ would represent all the iterations, the matrix A_ϕ would be an upper triangular matrix (i.e., $a_{ij} = 0$, $i > j$).

Our iterative procedure moves from one preference relation to another so that the upper bound ξ is decreased. A new preference relation is obtained swapping the actions in position i and position j when

$$\sum_{k=i+1}^{j} a_{ik} < \sum_{k=i+1}^{j} a_{ki} \text{ for } i < j. \tag{4.6}$$

In this case the new upper bound would be $\xi' = \xi + \sum_{k=i+1}^{j} (a_{ik} - a_{ki})$. No further improvement is possible if one of the following conditions holds:

- condition (4.6) does not hold for any two i, j
- all the elements above the main diagonal in A_ϕ are greater than their symmetric counterpart.

The complete swapping procedure is sketched in Algorithm 4.4.

Since the successive values of ξ are always decreasing there is guarantee that there are no cycles. Although the total number of iterations is not polynomial, since in the worst case we should analyze all permutations of the actions, in

```
Require: an arbitrary initial preference relation $\phi$
Require: the matrix of actions $A_\phi$ of size $(n+1,\, n+1)$
  repeat
    for $i = 1$ to $n$ do
      for $j = i+1$ to $n+1$ do
        if $\sum_{k=i+1}^{j} a_{ik} < \sum_{k=i+1}^{j} a_{ki}$ then
          swap action in position $i$ with action in position $j$ and update $A_\phi$
  until all elements under the diagonal are smaller than their symmetric coun-
  terpart or no any change is made in a cycle
```

Algorithm 4.4: *Swapping procedure to find a preference relation ϕ with small errors*

practice we use an initial preference relation that guarantees the stopping condition of the swapping procedure in few iterations (see Algorithm 4.5). In the procedure the sum over all elements in the column j except a_{jj} represents the number of iterations in which the action in position j of the preference relation ϕ' was possible but not chosen. The smaller this value, the higher preference is given to the action.

```
Require: an arbitrary initial preference relation $\phi'$
Require: the matrix of actions $A_{\phi'}$ of size $(n+1,\, n+1)$
  for $i = 1$ to $n+1$ do
    $\phi(i) \leftarrow \arg\min_j \sum_{k \neq j} a_{kj}$
    Eliminate column and row $\phi(i)$ from $A_{\phi'}$
```

Algorithm 4.5: *Procedure to define an initial preference relation ϕ before swapping*

4.5 Experimental results

In this section we present the application of RL methods, as discussed in Chapter 2, to four cases of the OA in the single resource model presented in this chapter. In all cases we consider Poisson arrivals, and discount factor $\gamma = 0.9$. Table 4.3 shows the data of the four cases.

The first column represents all the characteristics defining an instance of the model, characteristics defining the type of orders and the capacity profile. Each case is in a column. When some cases share the same characteristics, the cells corresponding to those characteristics have been merged in the table.

All cases have 5 types of orders. In the first case all types of orders are very similar, they only differ in the reward for acceptance; and there is no capacity perturbation. The second case is a variation of the first case where

	Case 1	Case 2	Case 3	Case 4
n	5			
λ	$(0.2, 0.2, 0.2, 0.2, 0.2)$	$(0.4, 0.4, 0.4, 0.4, 0.4)$	$(0.1, 0.1, 0.1, 2, 2)$	
m	$(2, 2, 2, 2, 2)$		$(1, 1, 1, 6, 6)$	
t	$(5, 5, 5, 5, 5)$		$(3, 3, 3, 5, 5)$	
w	$(3, 3, 3, 3, 3)$		$(3, 6, 1, 1, 2)$	
r	$(3, 6, 9, 12, 15)$		$(12, 18, 5, 1, 2)$	
r/ω	$(1, 2, 3, 4, 5)$		$(4, 3, 5, 1, 1)$	
H	5			
C_{max}	2		3	
p	-	$(0, 1)$	-	$(-1, 0, 1)$
$\Pr(p)$	-	$(0.2, 0.8)$	-	$(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$
η	-	6	-	9
$\|S\|$	59049		401408	
$\|S \times A\|$	354294		2408448	

Table 4.3: *Data of the cases in the shared resource model*

there is an increase in the flow of orders (higher arrival frequency) and there is also capacity perturbation. The last two cases introduce a higher variation between the types of orders, no one characteristic is the same for all types of orders. Both cases are very similar but the third case does not consider capacity perturbation. Since these two last cases consider more arrivals, and more capacity, they have larger state spaces than the first two cases.

In all cases we compare the results of the RL-agents against the general one-parameter-heuristics as presented in Section 4.3. As a performance measure we use a different measure than the measures we used in Chapter 3. Since in that case we have the optimal policies to compare with, we use the MSE-Q^{π_t} to evaluate the learned RL-policy. Here we want to use an average reward related measure to evaluate the learned RL-policy, but the measure AvToRew_t as used in Chapter 3 says more about the evolution of the learning process than about the final result of the learning. In order to evaluate the learned policy in this chapter we use AR_{20000}, defined as the total average reward accumulated per unit of time up to the time corresponding to iteration 20000 following *only the learned RL-policy.* Furthermore, we apply the technique explained in Section 4.4.1 in order to interpret the policy learned by the RL-agents.

We use one-way ANOVA test to compare the performance of different policies. We obtain for each policy the performance on five independent samples of this case. One-way Anova tests are useful to compare the means of independent sample distributions. We look at two indicators: F and Pr. Large values of F indicates that the variation among the sample means is large relative to the variation within the sample and hence the null hypothesis of equal means should be rejected. Pr indicates whether the value of F is large enough to reject the null hypothesis, it gives the probability that we are mistaken when rejecting the hypothesis of equal means. So here we consider that $Pr \leq 0.05$ shows significant evidence against the equality of the means.

4.5.1 Case 1

In this case all types of orders are equal except for the reward upon acceptance. Table 4.4 shows the data for this case.

$order_i$	λ_i	w_i	t_i	m_i	r_i	$\frac{r_i}{w_i}$	*Capacity profile*
1	0.2	3	5	2	3	1	
2	0.2	3	5	2	6	2	$H = 5$
3	0.2	3	5	2	9	3	$C_{\max} = 2$
4	0.2	3	5	2	12	4	
5	0.2	3	5	2	15	5	

Table 4.4: *Data of case 1 in the shared resource model*

Good policies in this case could be like the type of parametric heuristics we introduced before. As we mentioned, we first try these one-parameter heuristics that use the same parameter for all types of orders. Table 4.5 summarizes the results of these heuristics. The best of these heuristics is *OrderQuality*(2) which chooses orders with reward per capacity-request over 2, so this heuristic never chooses orders of type 1.

scenario	AR_{20000}	scenario	AR_{20000}
greedy	6.28	*CapacityLevel(0.8)*	6.16
OrderQuality(2)	**6.61**	*CapacityLevel(0.7)*	5.99
OrderQuality(3)	6.42	*CapacityLevel(0.6)*	5.82
OrderQuality(4)	5.26	*CapacityLevel(0.5)*	5.58
OrderQuality(5)	2.97	*CapacityLevel(0.4)*	5.18
CapacityLevel(0.9)	6.22	*CapacityLevel(0.3)*	3.94

Table 4.5: *Performance of some heuristics for case 1 in the shared resource model*

Table 4.6 shows the results and the parameters for the RL-agents trained in this case. The parameters correspond to the *learning schedule* as described in Section 2.3.2. This schedule defines the number of hidden neurons (θ), the number of iterations of the training process (T), the initial value of the learning and exploration rates (α_0, ϵ_0), and the parameters for the learning and exploration rates-decreasing functions ($T\alpha, T\epsilon$). These parameters are chosen through experimental experience and are by no means optimized.

	θ	T	α_0	T_α	ϵ_0	T_ϵ	AR_{20000}
1	50	2×10^4	10^{-3}	2×10^3	1	2×10^3	6.572
2	100	3×10^4	10^{-3}	3×10^3	1	3×10^3	**6.658**
3	150	5×10^4	10^{-3}	5×10^3	1	5×10^3	**6.703**
4	150	10^5	10^{-3}	10^4	1	10^4	**6.711**
5	250	10^5	10^{-3}	10^4	1	10^4	**6.732**
6	300	10^5	10^{-3}	10^4	1	10^4	**6.733**
7	350	10^5	10^{-3}	10^4	1	10^4	**6.751**

Table 4.6: *Parameters and performance of some RL-agents in case 1 of the shared resource model*

Except the first RL-agent, all the others outperform the heuristic *OrderQuality*(2). Table 4.7 shows the results of a one-way ANOVA test with the *OrderQuality*(2) heuristic and the last four RL-agents, each simulated using five independent samples of this case.

The table shows significant evidence against the equality of the performance of the RL-agents and the *OrderQuality*(2) heuristic (Pr=0.0439). However, there is not sufficient evidence to reject the equality in the performance of the

RL-agents (Pr=0.415). This reveals that adding more hidden neurons than 150 does not really help to improve the performance of an RL-agent in this case.

OrderQuality(2)+RL-agents		RL-agents	
F	Pr	F	Pr
2.985	0.0439	1.007	0.415

Table 4.7: *Results of a one-way ANOVA test for some RL-agents and the heuristic OrderQuality(2) in case 1 of the shared resource model*

The best RL-agent outperforms the heuristic *OrderQuality*(2) by 2.12%, but what has this RL-agent learned? In order to answer this, we applied the simple data mining technique we explained in Section 4.4.1.

Interpreting the learned RL-policy

Using the preference relation $\phi = (5, 4, 3, 2, 1, 0)$ defined by the profitability $\frac{r_i}{w_i}$ of the types of orders and 20000 iterations from the trained agent, we obtain the matrix A_ϕ as shown in Table 4.8.

Selected	**Possible Actions**					
Actions	5	4	3	2	1	0
5	**2859**	135	209	438	500	2859
4	368	**2738**	210	374	485	2738
3	276	284	**2513**	365	485	2513
2	89	104	98	**1453**	336	1453
1	0	0	1	0	**312**	312
0	60	19	30	734	1768	**10125**

Table 4.8: *Selected action against possible actions in 20000 iterations of the RL-agent in case 1 of the shared resource model*

To illustrate the meaning of the table let us look at the number 284 at entry $(3, 2)$. It means that action 3 was selected 284 times when action 4 was also possible. The number 284 also represents violations of the actual preference relation in which action 4 has a higher preference than action 3.

Using the procedure to obtain an initial preference relation in the root (Algorithm 4.5) we obtain the preference relation (4,3,5,2,1,0) and in some steps of the swapping procedure (in Algorithm 4.4) we obtain the best preference relation for the zero level of the decision tree ϕ_0 = (3,4,5,2,0,1). Given higher priority to action zero (rejection) than to action 1 means that action 1 would never be taken following a heuristic defined by that preference relation since action zero is always possible. In any case the type of order 1 is the lease profitable type of orders. In the sequel we will not write in the preference relation the actions with less preference than action 0.

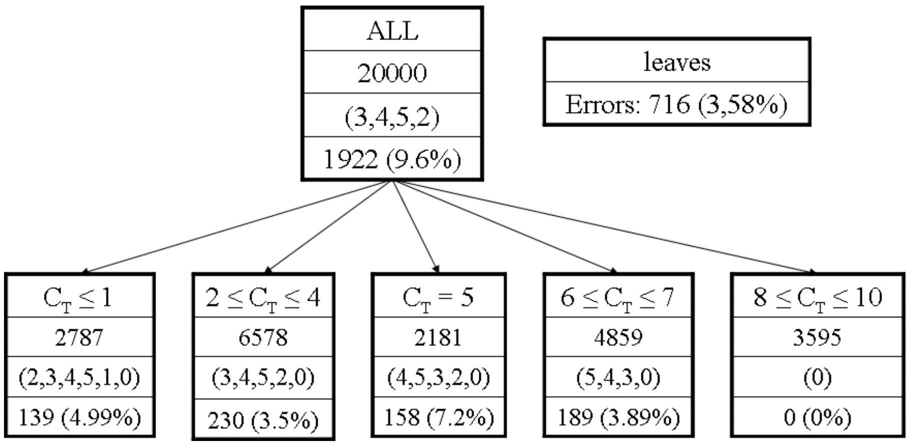


Figure 4.5: *One level decision tree representing an approximation to the RL-policy in case 1 of the shared resource model. The set of rules represented by this tree has a quality of 96.42% and the correspondent heuristic achieves a performance of 6.70*

In this case one may think that a better preference relation than ϕ_0 is for example ϕ' =(5,4,3,2,0) which accepts all orders except orders of type 1 using their profitability as a preference relation. This is true in the class of heuristics defined only by a preference relation. Let us call these heuristics $Priority(\phi)$ when they are just defined by the preference relation ϕ. Note that $Priority(\phi')$ is exactly the heuristic *OrderQuality(2)*. Table 4.9 shows the performance of the heuristics and the errors of the rules with respect to the RL-policy when following these two preference relations.

preference relation	AR_{20000}	errors
$Priority(\phi_0)$	6.43	1922
$Priority(\phi')$	6.61	2289

Table 4.9: *Results of 2 preference orders in case 1 of the shared resource model*

Although $Priority(\phi')$ has a better performance, $Priority(\phi_0)$ has fewer errors with respect to the RL-policy (90,39% of quality). $Priority(\phi_0)$ represents only the level zero of the tree, next we can continue splitting the tree using the total occupied capacity C_T as the splitting parameter. Figure 4.5 shows a decision tree with another level. Each node has four rows; the description of the node, the number of transitions covered by the node, the preference relation, and the number of errors (between parentheses the percentage of the error).

The leaves of the tree represent the following set of rules:

1. If $C_T \leq 1$ then the preference relation is $(2,3,4,5,1,0)$,
2. if $2 \leq C_T \leq 4$ then the preference relation is $(3,4,5,2,0)$,
3. if $C_T = 5$ then the preference relation is $(4,5,3,2,0)$,
4. if $6 \leq C_T \leq 7$ then the preference relation is $(5,4,3,0)$,
5. if $8 \leq C_T \leq 10$ then reject.

This set of rules has an error of 716 which gives a quality of 96.42% representing the RL-policy. Note that the preference relation is mainly related to the relation in which orders are chosen one at a time. Choosing less profitable orders when the utilization of capacity over the planning horizon is low, in the first rules, is the way of choosing these orders up to some point, it does not mean that the more profitable orders will be rejected since according to the other rules, they will be chosen in the sequel.

According to the values of C_T, the heuristic represented by the decision tree is similar to the heuristic *CapacityLevel*(0.4,0.8,1,1,1). This heuristic chooses orders of type 1 if the capacity is kept up to a 40% of the total capacity and there is no other type of order. Orders of type 2 are chosen if the capacity is kept up to an 80% of the total capacity and there are no orders of types 3, 4 and 5. For the other three types of orders there is no restriction on capacity, they will be chosen according to a preference relation (5,4,3). This heuristic *CapacityLevel*(0.4,0.8,1,1,1) achieves an AR_{20000} of 6.76. A one-way ANOVA test with the heuristic *CapacityLevel*$(0.4, 0.8, 1, 1, 1)$ and the RL-agent on five independent samples of this case shows that there is not sufficient evidence to reject the equality of the average rewards for both agents (F=0.03, Pr=0.87).

This case shows how an RL-agent is able to learn a more complex policy than in the previous chapter, not only a preference relation for the types of orders but a recognition of the different situations according to the utilization of capacity over the planning horizon.

4.5.2 Case 2

In this case we introduce capacity perturbation and in order to be able to appreciate the effects of the capacity perturbation, we also increase the flow of orders.

$order_i$	λ_i	w_i	t_i	m_i	r_i	$\frac{r_i}{w_i}$	Capacity profile
1	0.4	3	5	2	3	1	H=5
2	0.4	3	5	2	6	2	$C_{\max} = 2$
3	0.4	3	5	2	9	3	$\Pr(p) = \begin{cases} 0.2 & p=0 \\ 0.8 & p=1 \end{cases}$
4	0.4	3	5	2	12	4	
5	0.4	3	5	2	15	5	$\eta = 6$

Table 4.10: *Data case 2 in the shared resource model*

Table 4.11 summarizes the results of the one-parameter heuristics. The best of these heuristics is $OrderQuality(4)$ which only chooses orders with reward per capacity-request from 4, so this heuristic only chooses orders of type 4 and 5.

scenario	AR_{20000}	scenario	AR_{20000}
greedy	2.738	$CapacityLevel(0.8)$	3.79
$OrderQuality(2)$	3.254	$CapacityLevel(0.7)$	4.12
$OrderQuality(3)$	3.877	$CapacityLevel(0.6)$	4.33
$OrderQuality(4)$	**4.614**	$CapacityLevel(0.5)$	4.49
$OrderQuality(5)$	4.597	$CapacityLevel(0.4)$	4.4
$CapacityLevel(0.9)$	3.21	$CapacityLevel(0.3)$	4.07

Table 4.11: *Performance of some heuristics for case 2 in a shared resource*

Table 4.12 shows the results and the parameters for the RL-agents trained in this case.

	θ	T	α_0	T_α	ϵ_0	T_ϵ	AR_{20000}
1	100	4×10^4	10^{-3}	4×10^3	1	4×10^3	**4.719**
2	100	5×10^4	10^{-3}	5×10^3	1	5×10^3	**4.740**
3	100	6×10^4	10^{-3}	6×10^3	1	6×10^3	**4.753**
4	100	7×10^4	10^{-3}	7×10^3	1	7×10^3	**4.766**
5	100	8×10^4	10^{-3}	8×10^3	1	8×10^3	**4.722**
6	250	12×10^4	10^{-3}	12×10^3	1	12×10^3	**4.769**

Table 4.12: *Parameters and performance of some RL-agents in case 2 of the shared resource model*

All RL-agents outperform the heuristic $OrderQuality(4)$. Table 4.13 shows the results of a one-way ANOVA test on five independent samples of this case for the best three RL-agents and the Heuristic $OrderQuality(4)$. The first part of the table shows strong evidence against the equality of all these agents' average reward. However the second part of the table show that there is not sufficient evidence to reject the equality of the average reward from these best three RL-agents.

$OrderQuality(4)+$ best three RL-agents		best three RL-agents	
F	Pr	F	Pr
57.05	7.423345×10^{-7}	0.0265	0.8840896

Table 4.13: *Results of a one-way ANOVA test for the best three RL-agents and the heuristic OrderQuality(4) in case 2 of the shared resource model*

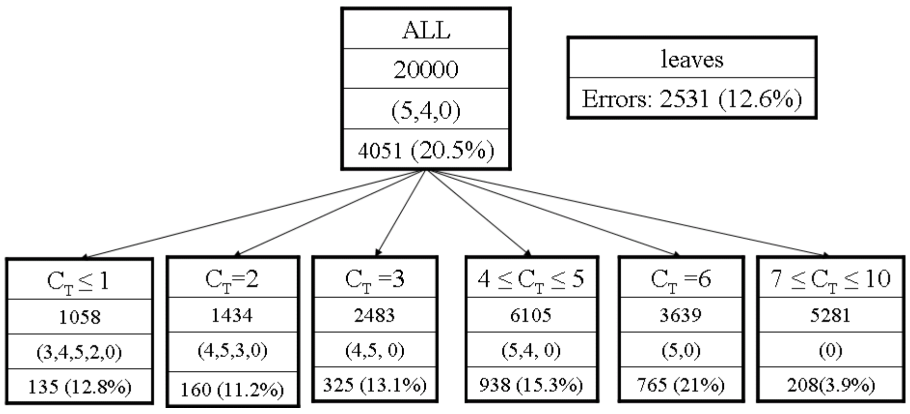


Figure 4.6: *One level decision tree representing an approximation to the RL-policy in case 2 of the shared resource model. The set of rules represented by this tree has a quality of 87.42% and the heuristic achieves an average reward of 4.84*

Interpreting the learned RL-policy

Here we analyze the policy learned by the last RL-agent represented in Table 4.12 using 20000 iterations from this trained agent. After the initialization procedure for the preference relation, no further improvement was possible and we obtained at zero level the preference relation (5,4,0) with 4051 errors which makes 79.5% of quality. This is exactly the heuristic $OrderQuality(4)$. Adding a new level according to the total capacity, we obtain 2531 errors and 87.4% of quality, see Figure 4.6.

However from this level one can notice how the tendency of the RL-agent is to be more careful using capacity. Note that utilization of capacity over the planning horizon is kept under 90% which helps avoiding penalization for using extra capacity when capacity perturbation occurs. A heuristic following the set of rules represented by this tree achieves an AR_{20000} of 4.84 which is 1.77% better than the performance of the RL-agent. According to the values of C, this heuristic is similar to the heuristic *CapacityLevel*$(0, 0.4, 0.5, 0.8, 0.9)$ which achieves an AR_{20000} of 4.96. A one-way ANOVA test for the heuristic *CapacityLevel*$(0, 0.4, 0.5, 0.8, 0.9)$ and the RL-agent on five independent samples of this case shows that there is very strong evidence to reject the equality of the average rewards for both agents ($F = 270.13$, $\Pr = 1.9 \times 10^{-7}$).

This case shows how an RL-agent is able to learn a good policy even under capacity perturbations, when some capacity must be kept free not only waiting for better opportunities to accept the more profitable orders, but also to avoid penalizations for using extra capacity. The analysis of what the RL-agent had learned, guided us to find better heuristics. We believe that by training other

RL-agents we can continue improving the performance.

4.5.3 Case 3

In this case there is higher variation between orders of different types. The more profitable orders are of type 3, 1 and 2 in that order. However these are also the less frequent type of orders and with the smallest due-time.

$order_i$	λ_i	w_i	t_i	m_i	r_i	$\frac{r_i}{w_i}$	Capacity profile
1	0.1	3	3	1	12	4	
2	0.1	6	3	1	18	3	H=5
3	0.1	1	3	1	5	5	
4	2	1	5	6	1	1	$C_{\max} = 3$
5	2	2	5	6	2	1	

Table 4.14: *Data case 3 in the shared resource model*

Table 4.15 summarizes the results of the one-parameter heuristics for this case. The best of these heuristics is *CapacityLevel*(0.6) which only chooses orders if the capacity utilization is kept under 60% of the total capacity in the planning horizon.

scenario	AR_{20000}	scenario	AR_{20000}
greedy	3.138	*CapacityLevel*(0.7)	4.217
OrderQuality(3)	3.192	*CapacityLevel*(0.6)	**4.255**
OrderQuality(4)	1.646	*CapacityLevel*(0.5)	4.252
OrderQuality(5)	0.479	*CapacityLevel*(0.4)	4.242
CapacityLevel(0.9)	3.612	*CapacityLevel*(0.3)	4.113
CapacityLevel(0.8)	4.098	*CapacityLevel*(0.2)	3.749

Table 4.15: *Performance of some heuristics for case 3 in a shared resource model*

Table 4.16 shows the results and the parameters for the RL-agents trained in this case.

All RL-agents outperform the heuristic *CapacityLevel*(0.6) and the best of these RL-agents improves it by 21,1%. Table 4.17 shows the results of a one-way ANOVA test on five independent samples of this case for the best three RL-agents and the heuristic *CapacityLevel*(0.6). The results show strong evidence against the equality of all these agents average reward. However there is not sufficient evidence to reject the equality of the average reward of the best three RL-agents.

	θ	T	α_0	T_α	ϵ_0	T_ϵ	AR_{20000}
1	50	3×10^4	10^{-3}	3×10^3	1	3×10^3	**4.627**
2	100	7×10^4	10^{-3}	7×10^3	1	7×10^3	**5.103**
3	150	10×10^4	10^{-3}	10×10^3	1	10×10^3	**5.107**
4	200	12×10^4	10^{-3}	12×10^3	1	12×10^3	**5.161**
5	250	14×10^4	10^{-3}	14×10^3	1	14×10^3	**5.152**
6	300	16×10^4	10^{-3}	16×10^3	1	16×10^3	**5.174**

Table 4.16: *Parameters and performance of some RL-agents in case 3 of the shared resource model*

CapacityLevel(0.6)+ best three RL-agents		best three RL-agents	
F	Pr	F	Pr
1148	5.107×10^{-15}	0.8892	0.448

Table 4.17: *Results of a one-way ANOVA test for the best three RL-agents and the heuristic CapacityLevel(0.6) in case 3 of the shared resource model*

Interpreting the learned RL-policy

Here we analyze the policy learned by the last RL-agent represented in Table 4.16 using 20000 iterations from this trained agent. After the initialization procedure, no further improvement was possible and we obtained at zero level the preference relation (2,3,1,4,0) with 6327 errors which makes 68.4% of quality. Adding a new level according to the total capacity levels to the decision tree, it makes 3306 errors and 83.5% of quality, see Figure 4.7.

A heuristic following the set of rules represented by this tree achieves an AR_{20000} of 5.179, very close to the performance of the RL-agent. According to the values of C, this heuristic is similar to the heuristic *CapacityLevel*(0.8, 0.9, 1, 0.4, 0.4) which achieves an AR_{20000} of 5.175. A one-way ANOVA test for the heuristic *CapacityLevel*(0.8, 0.9, 1, 0.4, 0.4) and the RL-agent on five independent samples of this case shows that there is not significative evidence to reject the equality of the AR_{20000} for both agents ($F = 2.73$, $\text{Pr} = 0.1371$).

This case shows that an RL-agent is able to learn a good acceptance policy in a case with more variation in the characteristics of the different types of orders.

4.5.4 Case 4

This case is a variation on case 3 introducing capacity perturbation, see the data in Table 4.18.

Table 4.19 summarizes the results of the one-parameter heuristics. The

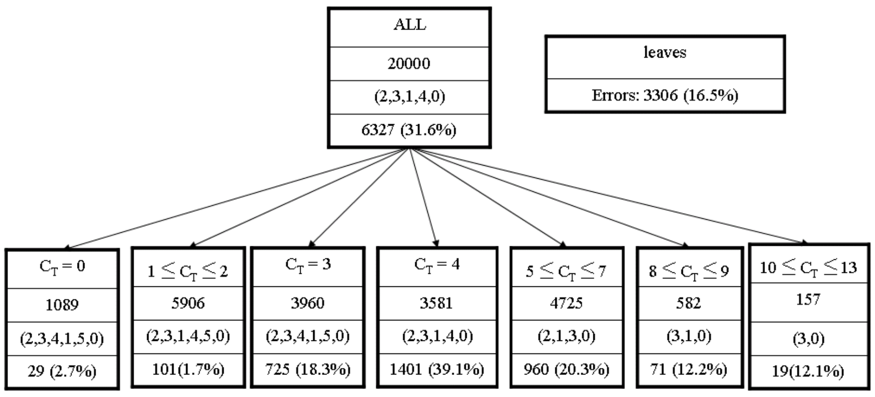


Figure 4.7: *One level decision tree representing an approximation to the RL-policy in case 3 of the shared resource model. The set of rules represented by this tree has a quality of 83.52% and the heuristic achieves a performance of 5.179*

$order_i$	λ_i	w_i	t_i	m_i	r_i	$\frac{r_i}{w_i}$	Capacity profile
1	0.1	3	3	1	12	4	H=5
2	0.1	6	3	1	18	3	$C_{\max} = 3$
3	0.1	1	3	1	5	5	$\Pr(p) = \begin{cases} \frac{1}{3} & p=-1 \\ \frac{1}{3} & p=0 \\ \frac{1}{3} & p=1 \end{cases}$
4	2	1	5	6	1	1	
5	2	2	5	6	2	1	

Table 4.18: *Data case 4 in the shared resource model*

best of these heuristics is *CapacityLevel*(0.6) which only chooses orders if the capacity utilization is kept under 60% of the total capacity in the planning horizon.

Table 4.20 shows the results and the parameters for the RL-agents trained in this case.

All RL-agents outperform the heuristic *CapacityLevel*(0.6). Table 4.21 shows the results of a one-way ANOVA test on five independent samples of this case for the best three RL-agents and the Heuristic *CapacityLevel*(0.6). The table shows strong evidence against the equality of all these agents average reward. However there is not sufficient evidence to reject the equality of the average reward from the RL-agents.

Interpreting the learned RL-policy

Here we analyze the policy learned by the last RL-agent represented in Table 4.20 using 20000 iterations from this trained agent. After the initialization pro-

scenario	AR_{20000}	scenario	AR_{20000}
greedy	1.231	$CapacityLevel(0.7)$	3.71
$OrderQuality(3)$	3.083	$CapacityLevel(0.6)$	**3.735**
$OrderQuality(4)$	1.646	$CapacityLevel(0.5)$	3.727
$OrderQuality(5)$	0.479	$CapacityLevel(0.4)$	3.718
$CapacityLevel(0.9)$	3.037	$CapacityLevel(0.3)$	3.636
$CapacityLevel(0.8)$	3.392	$CapacityLevel(0.2)$	3.37

Table 4.19: *Performance of some heuristics for case 4 in a shared resource model*

	θ	T	α_0	T_α	ϵ_0	T_ϵ	AR_{20000}
1	50	2×10^4	10^{-3}	2×10^3	1	2×10^3	**4.437**
2	100	6×10^4	10^{-3}	6×10^3	1	6×10^3	**4.617**
3	150	8×10^4	10^{-3}	8×10^3	1	8×10^3	**4.668**
4	200	11×10^4	10^{-3}	11×10^3	1	11×10^3	**4.728**
5	250	13×10^4	10^{-3}	13×10^3	1	13×10^3	**4.712**
6	300	16×10^4	10^{-3}	16×10^3	1	16×10^3	**4.794**

Table 4.20: *Parameters and performance of some RL-agents in case 4 of the shared resource model*

cedure, no further improvement was possible and we obtained at zero level the preference relation (1,2,3,4,0) with 6362 errors which makes 68.2% of quality. Adding a new level according to the total capacity levels to the decision tree, it makes 2022 errors and 89.9% of quality, see Figure 4.8.

A heuristic following the set of rules represented by this tree achieves an AR_{20000} of 4.77. According to the levels of capacity, this heuristic is similar to the heuristic *CapacityLevel(0.8, 0.75, 0.4, 0.35, 0.4)* which also achieves an AR_{20000} of 4.77. A one-way ANOVA test for the heuristic *CapacityLevel(0.8, 0.75, 0.4, 0.35, 0.4)* and the RL-agent on five independent samples of this case does not show significative evidence to reject the equality of the average rewards for both agents ($F = 3.46$, $P = 0.1$).

This case shows that an RL-agent is able to learn a good acceptance policy in a case with more variation in the characteristics of the different type of orders and with capacity perturbation.

4.6 Conclusions

This chapter discussed the application of RL for OA in a shared resource model. We have defined a general class of heuristics for the OA problem. These heuristics are also used in the next chapters for more complex models. Particularly we used two type of parametric heuristics $OrderQuality(b)$ and $CapacityLevel(\rho)$.

CapacityLevel(0.6)+ best three RL-agents		best three RL-agents	
F	Pr	F	Pr
157.6	6.66×10^{-10}	0.02	0.82

Table 4.21: *Results of a one-way ANOVA test for the best three RL-agents and the heuristic CapacityLevel(0.6) in case 4 of the shared resource model*

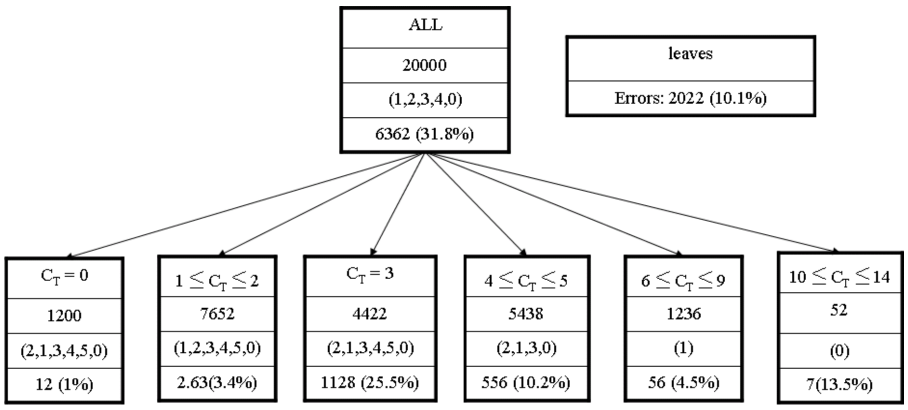


Figure 4.8: *One level decision tree representing an approximation to the RL-policy in case 4 of the shared resource model. The set of rules represented by this tree has a quality of 89.1% and the heuristic achieves a performance of 4.77*

The heuristics are used as a means of measuring the RL performance. However, unlike the RL, the heuristics need to have complete information in order to be defined. We have shown that training RL-agents is a good alternative to obtain good OA policies. We presented four different cases, with different degrees of complexity. In all the cases we obtained RL-agents that outperform the simple heuristic rule *always accept* and the best of the one-parameter-heuristics. Mining data from the trained RL-agents' knowledge, we could interpret the solution found by the RL-agents. We define here a general framework to interpret the learned RL-policy. Using decision trees we obtained an interpretation of the learned RL-policy as the general parametric heuristics we previously defined. So the RL-approach helped as well to find good parameters for the general class of parametric heuristics.

It is questionable whether using some more sophisticated data mining algorithms we could find better interpretations of the RL-policies, but it goes beyond the boundaries of this research. Furthermore we believe that training other RL-agents we could continue improving the performance. Using the results of the trained RL-agents in this model, we will define a methodology for tuning the RL-parameters in the next chapter.

Chapter 5

OA in a multiresource server

In this chapter we consider an extension of the model from the previous chapter. The difference with respect to the orders is that each order consists of a set of jobs which need to be performed on different resources following a specific route. So the difference with respect to the capacity is that here the server system is a multiresource server.

Since the allocation of capacity is a more complex task here, we use an integrated simulation environment defined by Ebben, Hans and Olde Weghuis (OldeWeghuis, 2002; Ebben et al., 2005). In such an environment, a tentative loading plan is used for reactive scheduling. Reactive scheduling is commonly used under uncertainty, when the existing schedule is updated using new available information. The tentative loading plan is also used to support the OA decisions. In Section 5.1 we explain these ideas in more detail and in Section 5.2 we present the SMDP approach. In Section 5.3 we discuss the application of RL to this problem and present a methodology for tuning the RL-agents. The experimental results are presented in Section 5.4 where the performance of the best RL-agents is compared with some OA Heuristics. Data Mining is used for a better understanding of what the RL-agents learn. Descriptive rules help to interpret the learned knowledge and to elaborate more sophisticated OA heuristics. Finally in Section 5.5 we draw some conclusions.

5.1 Problem description

The arrival and collection of orders occur in similar ways as in the previous chapter. The orders arrive continuously but they are collected upon arrivals and they are only processed at discrete equidistant moments: decision moments. Here again we consider aggregate batch arrivals.

In this chapter we consider a larger order's attribute set than in the previous chapter, specifically we include the route of the order. In general order definitions are based on a finite number n of order types that arrive continuously over time. Each order asks for service on a server with M different resources. Order i consists of a set O_i of n_i jobs which have to be performed in a predetermined routing. The jobs in O_i ($O_i = \{J_{ij}|j = 1, 2..., n_i\}$) define a fixed routing σ_i as a linear precedence relation. A routing σ_i is a sequence of pairs (R_{ij}, B_{ij}) where R_{ij} is the resource and B_{ij} is the processing time distribution with mean β_{ij}, of job J_{ij} ($\sigma_i = ((R_{i1}, B_{i1})..(R_{in_i}, B_{in_i}))$). Order i is characterized by the arrival rate (λ_i), due-time (t_i), a routing (σ_i) and generates an immediate reward (r_i) upon acceptance. Here we use the same due-time concept as explained in Chapter 4. Note that for reasons of transparency t_i can be measured in different units, e.g. in hours or in stages over the planning horizon. In this chapter we express t_i in the same time units as we express the processing times.

Allocation of orders in this chapter is considered in a different way than in the previous chapters. Again we use a fixed prescription, but in order to give more flexibility with respect to the capacity planning and the uncertainty here in, we consider the resequencing problem. The resequencing problem is about rescheduling: every time an order is accepted a new *tentative loading plan* is made with all the accepted orders, see (Pinedo & Chao, 1999). In the previous chapters capacity is reserved when an order is accepted and this assignment is not changed afterwards. Here capacity allocation is made following the tentative loading plan, but when a new order is accepted we construct a new loading plan including the new order. The actual realization of the plan is one job at a time per resource, see (Ebben et al., 2005). We believe this gives more flexibility for the future allocation of high priority orders.

Order acceptance has to be planned at each resource over a planning time horizon with H stages. Each stage spans the time between two decision epochs. There is a maximum capacity $C_{\max}$ for all stages at each resource that can only be excessed at the cost of a penalty as we explain below. The MxH matrix L describes the capacity utilization according to the tentative loading plan. L is called the *loading profile*. Figure 5.1 shows an example of capacity utilization according to a loading plan in a planning horizon with $H = 5$ stages, $M = 2$ and $C_{\max} = 8$ where $L = \begin{pmatrix} 3 & 4 & 4 & 8 & 3 \\ 6 & 3 & 3 & 6 & 5 \end{pmatrix}$.

The orders in the system are disaggregated into their jobs. In order to compute the tentative loading plan we keep a list of all accepted jobs which are waiting to be processed: the *job list (JL)*. Per each job we keep a record with the following information (Job_id, release date, due-date, predecessor, successor, resource, expected processing time, active).

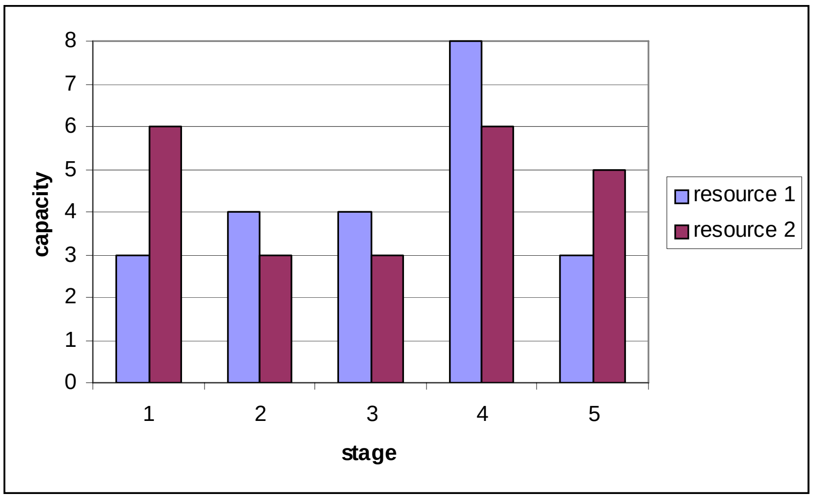


Figure 5.1: *Capacity utilization according to a loading plan in a planning horizon with $H = 5$ stages, $M = 2$ and $C_{\max} = 8$*

We compute the release date and the due-date of job J_{ij} given the current time using the following formulas:

$$J_{ij}.release\_date = current\_time + \sum_{k=1}^{j-1} \beta_{ik}$$

$$J_{ij}.due\_date = current\_time + t_i - \sum_{k=j+1}^{n_i} \beta_{ik}$$

The *active* entry is a boolean which indicates whether a job is available to be processed. This entry is true for all first jobs of an order and for all jobs of which the predecessor has been already served.

To compute the loading plan we use an Earliest due-date (EDD) rule with the jobs in the *job list* and information about the jobs that are being processed at each resource. In the sequel the MxH matrix EP describes the capacity utilization of the jobs that are being processed at the current stage at each resource, see Figure 5.2. EP is called the *execution profile*. Jobs in EP can not be rescheduled. The jobs in the *job list* are prioritized according to their due-dates. Whenever a resource is freed, the job with the earliest due-date from the active jobs, is selected to be processed next. Due to its simplicity the EDD rule is very popular, furthermore "this rule tends to minimize the maximum lateness among the jobs waiting for processing", see (Pinedo, 2002).

Since the resequencing problem cares for the utilization of capacity in a flexible way we use here a forward loading approach instead of the backward loading used in the previous chapter. Forward loading means: start loading at the first stages of the planning horizon and continue forward.

5.1.1 OA decision

At every decision moment a decision must be taken on the collected orders. By contrast with Chapter 4 we do not allow preemption of jobs. Late delivery is also not an option. Given the list of arrivals we impose that the OA decision is created sequentially instead of focussing on all possible subsets of accepted orders at once. The orders are chosen from the arrival list one by one while we call a *time off* at each decision moment. The tentative loading plan is also used to support the OA decision. In order to know if a chosen action fits in the capacity profile, the order is temporarily added to the *job list*. If a new loading plan can be made with all the previously accepted orders and a new arriving order such that no order is late, then the given order fits in the capacity, thus accepting the order is an option. When the available capacity is not sufficient for an arriving order, the only option is rejection for that order and the order is deleted from the job list.

Each single decision in the sequence during the *time off* is either the selection of one of the orders from the list or the rejection of all of them. If we choose an order, it is moved from the list of arrivals to the list of accepted orders into the job shop. If we reject, the decision process at the current time is finished with the consequent *time on*. The rejected orders are out of the system.

5.1.2 The tentative loading plan

The tentative loading plan is built during the *time off* in order to support the OA decision. In constructing the tentative loading plan jobs are removed one by one from the job list according to the EDD priority. The jobs removed from the job list are planned in the loading profile.

As in a simulation approach we consider the occurrence of events that change the system. We define two events: the release of a job (release event) and the completion of a job on a resource (completion event). The *event list* handles the occurrence of the events.

The release of a job means that this job becomes *active* in the *job list*: it can be loaded from now on. When a job is released we have to check whether it can be started. We can start job j when the corresponding resource is available and job j is the job with the earliest due-date. If a job is released but can not be started we handle this event by doing nothing.

The completion of a job j on a resource causes the resource to be available again; a new job can be started on this resource. At the same time, the possible successor of job j can be released. Here we describe the procedure in more detail. First some notation:

lt: loading time, i.e., the simulation clock, it keeps track of the value of simulated time while building the tentative loading plan.

J: the subset of the job list consisting of all unplanned jobs that still have to be processed.

J_A: the subset from the unplanned jobs J which are already released at loading time, i.e., the set of active jobs at loading time.

The following steps describe the making of the loading plan.

Step 0. Initialization:

- Set lt equal to 0.
- Set the *event list* empty
- Set J equal to the *job list.*
- Determine the initial set of active jobs, J_A: all jobs in J with release_date $\leq lt$ for which the preceding jobs have been finished.
- Add the completion times of the jobs that are in process to the event list. If no job is in process on a resource, add a completion event of a dummy job in the resource to the event list with time lt (this is equivalent to having a resource idle event).
- Add the release times of the first job of all orders to the event list (only if release time $> lt$, i.e., release is still in the future).

Step 1: Handling events:

Sort the event list on time. Take the first event from the event list and set the loading time lt equal to the time of this event. When there are more events scheduled at the same time, add all jobs that will be released as a result of these events to the active job set J_A. Otherwise the loading might depend on the order of appearance of the events in the event list. In case of a release event go to Step 2, in case of a completion event go to Step 3, else (no more events) go to Step 4.

Step 2: Release event

Suppose job J_{ij} is released on resource R_{ij} :

Add job J_{ij} to J_A if not added already in Step 1 or 3. Determine whether resource R_{ij} is available and whether job J_{ij} is the highest priority job in J_A on resource Rij (earliest due-date). When the resource R_{ij} is available and job

J_{ij} is the highest priority job, plan job J_{ij} on resource R_{ij}, add a completion event to the event list with time $lt + \beta_{ij}$ and remove job J_{ij} from J and J_A. Update the profile of resource R_{ij}. Remove current event from event list. Go to Step 1.

Step 3: Completion event

Suppose job Jij is completed on resource R_{ij}.

When job J_{ij} has a successor, add release event of the successor of job J_{ij} to the event list with time lt and add the successor directly to J_A. Select highest priority job from J_A that has to be processed on resource R_{ij}. If such a job is found, plan this job J_{kl} on resource R_{kl} (Note that $R_{kl} = R_{ij}$), add a completion event to the event list with time $lt + \beta_{kl}$ and remove job J_{kl} from J and J_A. Update the profile of resource R_{kl}. Remove current event from event list. Go to Step 1.

Step 4: Stop:

All jobs have been loaded (sets J and J_A are empty). Check the completion time of all orders. If there is at least one late order conclude that the EDD loading with no late orders is not possible.

In making the tentative loading plan we mention the update of the profile. When job J_{ij} is planned the loading profile of the corresponding resource is updated according to a forward loading procedure, see Algorithm 5.1.

```
Require: $J_{ij}$ and $L$
  $t \leftarrow lt$
  $remaining\_time = \beta_{ij}$
  $L' \leftarrow L$
  while $t \leq t_i$ and $remaining\_time \neq 0$ do
    $\xi \leftarrow \min(remaining\_time, Cmax - L(R_{ij}, t))$
    $L'(R_{ij}, t) \leftarrow L'(R_{ij}, t) + \xi$
    $remaining\_time \leftarrow remaining\_time - \xi$
    $t \leftarrow t + 1$
  if $remaining\_time \neq 0$ then
    $return($ not possible $)$
  else
    $return(L')$
```

Algorithm 5.1: *FL (L,J_{ij},lt): Forward loading procedure for job type J_{ij} in a loading profile L at time lt*

The tentative loading plan is used to support the OA decision during the *time off*, but it might also be used during the *time on.* This may happen because the actual realization of the orders during the *time on* could cause a capacity perturbation which requires a new loading plan. In the sequel we refer to the EDD loading plan as the EDDL plan.

5.1.3 Realization of an order

In the job shop the jobs are processed one at a time per resource when the resource is available and the jobs do not need to wait for predecessors to be finished. For the realization of the orders we use the tentative EDDL plan constructed during the OA. Note that such a plan considers the estimated processing time of the jobs. We allow for uncertainty due to realization differing from the expectation, causing a possible capacity perturbation.

The actual realization of the job may be different than expected. When the processing time is larger than the expected value, we compute a new tentative loading plan with the actual realization. If all the orders remain on time, we take the new loading plan as the current tentative loading plan; otherwise we keep the old loading plan and consider the use of non-regular capacity for the exceeded capacity at a certain cost. The cost of a unit of non-regular capacity is η.

5.1.4 Characterization of OA

In terms of the OA characterization (i)-(vi) given in Chapter 1 we assume the following:

(i) Order definitions are based on a finite number of types of orders where each type has a specific expected processing time, immediate reward upon acceptance, and a due-time. Violations of due-times are not allowed. The processing time of each order consists of a specified set of jobs which should be processed in a specific sequence on specific resources.

(ii) Arrivals of orders take place continuously, however, arrivals are only evaluated at discrete equidistant time moments, so we consider several arrivals in each discrete time unit (batch arrivals), and the probability of arrival is order type dependent.

(iii) Rejection of an order affects only the immediate reward of that order.

(iv) Order processing is type dependent, following the precedence relations defined by the set of jobs. Realization of an order may differ from the expectation causing a possible capacity perturbation.

(v) Capacity is considered as a job-shop, which may consist of different resources. Accepted orders are loaded over a planning horizon. There is a maximum regular capacity. It is possible to use non-regular capacity at a certain cost, in order to avoid possible violations of due-times for the accepted orders, that may be caused by the capacity perturbations.

(vi) The acceptance policy should choose a set of orders from the arrival batch. This policy is supported by a tentative loading plan which determines whether an order fits into the available capacity. If there is no available capacity for an order, the only option is rejection for that order.

5.2 The SMDP model

In this problem as described in the previous section the capacity profile has a different structure from what is used in the previous chapters. Here we should take into account the list of accepted jobs that are waiting to be served: the job list (JL) and information about which jobs are in service and how they are allocated: the execution profile (EP). The job-list is a complex structure, it contains the jobs of the accepted orders that are waiting their turn for realization. One might think that a structure with no more than $\sum\limits_{i=1}^{n} n_i$ entries should be enough to handle all the possible jobs in the job-list, but, for example, the first job of an order should have a different characterization (e.g. different release time) than the first job of an order of the same type which was accepted at a different time moment. The job list and the execution profile constitute the capacity profile $C = (EP, JL)$.

Having said this, the problem can be modeled as a Semi Markov Decision Problem similar to those in the previous chapters in the sense that the state may be characterized by the arrivals and the capacity profile. At every decision moment there is information about the orders requesting service, the execution profile and the list of accepted jobs which together define the state of the system $s = (k, C), C = (EP, JL)$. Here $k = (k_1, ..., k_N)$ is the order list and k_i represents the number of orders of type i requesting service. For each order type i we restrict the maximum number of orders in the order list to m_i. This number may be determined by the arrival probabilities or by the limited capacity. $EP = (e_{jt}) \in Z^{MH}$ is the execution profile and by e_{jt} we refer to the occupied capacity in the resource j at stage t of the planning horizon. JL is a table containing information of all the jobs of the accepted orders which are still waiting to be served. The information in the job-list table includes the release-time, the due-time, the expected processing time, the resource, the predecessors and successors of each job and a field indicating if the job is active or not.

The action space is $A = \{0, 1, .., N\}$. The set of allowed actions $A(s)$ for a state $s = (k, C)$ is defined as follows. Action $i \in [1..N]$ is allowed if order type i is present in the order list k and capacity is available for at least one occurrence of that type of order. Rejection of the complete order list is always an option (i.e., $A(s) = \{i \in [1..N] \; | k_i \neq 0, EDDL(C, i) \text{ is possible}\} \bigcup \{0\}$).

The one step state transition from the current state s is described in Table 5.1. The first column represents all the characteristics defining a state transition from the current state s. The other two columns represent these characteristics given that the action i in the current state is different from rejection (second column), respectively is rejection (third column). Recall that O_i refers to an order of type i.

current state (s)	$(k, C) = (k, EP, JL)$	
action (i)	$i \in A(s)$, $i \neq 0$	0
next state (s')	$(k - e_i, EP, JL + O_i)$	$(k',\ EP', JL')$
$rew(s,i)$	r_i	$-\eta \sum\limits_{r=1}^{M} (PT_r - C_{\max})^+$
$d(s,i)$	0	1
$Pr(s,i,s')$	1	$Pr\{k'\}Pr\,(\ EP', JL'/EP, JL)$

Table 5.1: *Dynamics of transition from current state s in the job-shop model*

In case an order of type i is chosen ($i \neq 0$), an immediate reward r_i is received and a deterministic transition to the next state s' occurs ($Pr(s,i,s') = 1$). Note that there is a *time off* at this point, i.e., the elapsed time $d(s,i)$ is 0. The order is then removed from the arrival list so the new arrival list is given by $k - e_i$ where e_i is a unit vector with 1 in the position i. The new job-list is updated including the jobs of the accepted order ($JL + O_i$), and the execution profile remains unchanged.

In case the action is rejection ($i = 0$), the decision process at the current time is finished; there is a *time on*; orders may start service on the available resources according to the EDDL plan until the next decision epoch. Remind that the system is changing continuously but we are considering only discrete moments. The transition to the next state is highly stochastic. At the next decision moment the clock is stopped again and the order arrival processes determine the new orders list k'. The jobs being processed in between the decision epochs stochastically depend on the job-list and the execution profile, they determine the new job-list and the new execution profile (EP', JL') and also the total immediate reward. This immediate reward is a random variable and in order to compute it we use the following steps:

step 1 Per each resource r compute the summation PT_r of all processing times of the jobs that were completed on resource r during the time on

step 2 $rew(s,0) = -\eta \sum\limits_{r=1}^{M} (PT_r - C_{\max})^+$

This negative reward is a penalization for each unit of non-regular capacity that was necessary to use in order to keep all accepted orders in time.

The objective is to find a deterministic policy π:

$$\pi(s) = \begin{cases} i & i \in \overline{1...n} \text{ select order type } i \\ 0 & \text{reject the arrival-list,} \end{cases}$$

which maximizes the performance of the system. The performance of the system is measured as the expected value of the total discounted reward. The corresponding Bellman equation for the state value function V^π is given by:

$$V^\pi(s) = rew(s, \pi(s)) + \gamma^{d(s,\pi(s))} \sum_{s'} Pr(s, \pi(s), s') V^\pi(s'),$$

where γ is the discount factor and $d(s,i)$ is the elapsed time as introduced before. The optimal action value function Q^* in this case satisfies:

$$Q^*(s,i) = rew(s,i) + \gamma^{d(s,i)} \sum_{s'} Pr(s,i,s') \max_{i'} Q^*(s',i')$$

and the optimal policy $\pi^*(s)$ can be determined by: $\pi^*(s) = \arg\max_i Q^*(s,i)$.

This problem is more complex than the problems analyzed in the previous chapters. The definition of the state is more complex since the capacity profile has a new component (JL). The transition probabilities and the immediate reward depend on the probability distributions of the processing time of the jobs in the job-list and on the probability of the jobs causing capacity excess.

A description of the detailed model, in particular the transition probabilities, is not necessary since our approach is simulation based. Next we discuss the applicability of our RL-approach to this model.

5.3 Reinforcement Learning approach

Here we discuss the application of Reinforcement Learning to the order acceptance problem described before. We use QL methods that aim to approximate the optimal Q-value function as discussed in Chapter 2. The RL-agent should learn an OA-policy while interacting with the environment. We focus here on traditional Q-learning using ANN as introduced in Chapter 2.

In this problem, the state representation has a complex structure, as explained above. Using the complete state information makes the input of the backpropagation ANN (see Section 2.2) very complex, demanding a more intensive learning process. Making an analogy with the state representation from the previous models, and taking into account that we use a simulation model instead of a detailed model, we decided to simplify the structure of the state representation. In order to do this we use the concept of *feature*, see (Bertsekas & Tsitsiklis, 1996). Features are used to summarize the most important characteristics of the state, they are usually motivated by problem insight. Below we simplify the state $s = (k, C)$ by defining on the one hand a feature for the order list k and on the other hand a feature for the capacity profile C.

We use the loading profile L as a feature of the capacity profile $C = (EP, JL)$. The loading profile combines the information in the job-list JL

Execution-profile + **Job-list** $\longrightarrow$ **Loading-profile**

$$\begin{pmatrix} 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Job	R	β	t	active
J_{12}	2	8	24	1
J_{21}	1	8	16	1
J_{22}	2	8	24	0

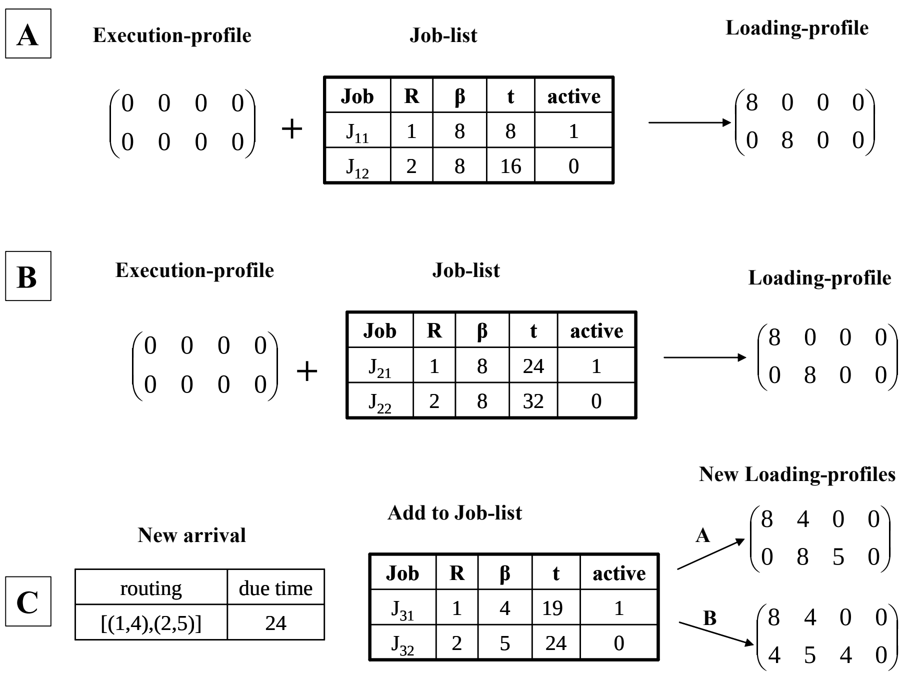
$$\begin{pmatrix} 8 & 8 & 0 & 0 \\ 8 & 0 & 8 & 0 \end{pmatrix}$$

Figure 5.2: *The loading-profile combines information from the execution-profile and the job-list by loading the jobs from the job-list into the execution profile. Here is an example in a case with $H = 4$ stages, $M = 2$ and $C_{\max} = 8$. The job list is simplified here for the example*

and the execution profile EP into a single structure according to the tentative loading plan. Figure 5.2 shows an example with $M = 2$, $H = 4$ and $C_{max} = 8$ hours. There is currently a job in execution using all the capacity at the first stage of the planning horizon on resource 1. Assume that one stage of the planning horizon spans 8 hours. The job-list as in the figure is a table, the first column gives the identification of the jobs waiting for service, the next three columns describe the resource (R), expected processing time (β) and due-time (t in hours). The last column specifies which jobs are ready to go in service ($active = 1$) and which should still wait ($active = 0$). There are 3 jobs in the job list. Job J_{22} should wait for its predecessor job J_{21} to be finished in order to be active. The loading-profile combines information from the execution profile and the job list by loading the jobs from the job list into the execution profile. In the example, J_{12} is loaded at stage 1 in resource 2, J_{21} is loaded at stage 2 in resource 1 and J_{22} is loaded at stage 3 in resource 2. In the figures in this section the due-time unit is stage over the planning horizon, and the job list has a simplified structure for reasons of transparency.

Basically we use the loading procedure as a feature extracting mapping from the capacity profile (EP, JL) to the loading profile L. In order to give more information, and since it does not imply an extra effort, instead of giving the complete arrival list k, we give to the RL-agent a feature kr of k which only considers the type of orders that do fit in the capacity profile.

The set of features (kr, L) is what the RL-agent receives at each interaction instead of receiving the current state. In this case, it is said that the RL-agent receives an observation $obs = (kr, L)$ of the state of the environment. In a way one may say that the overwhelming amount of information from the environment is filtered before it reaches the agent in the same fashion as bright sunlight is filtered by sun glasses before it reaches the human eye. The learning problem is now an extension of the basic SMDP explained in Section



A

Execution-profile

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} +$$

Job-list

Job	R	β	t	active
J_{11}	1	8	8	1
J_{12}	2	8	16	0

Loading-profile

$$\longrightarrow \begin{pmatrix} 8 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \end{pmatrix}$$

B

Execution-profile

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} +$$

Job-list

Job	R	β	t	active
J_{21}	1	8	24	1
J_{22}	2	8	32	0

Loading-profile

$$\longrightarrow \begin{pmatrix} 8 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \end{pmatrix}$$

C

New arrival

routing	due time
[(1,4),(2,5)]	24

Add to Job-list

Job	R	β	t	active
J_{31}	1	4	19	1
J_{32}	2	5	24	0

New Loading-profiles

$$\xrightarrow{A} \begin{pmatrix} 8 & 4 & 0 & 0 \\ 0 & 8 & 5 & 0 \end{pmatrix}$$

$$\xrightarrow{B} \begin{pmatrix} 8 & 4 & 0 & 0 \\ 4 & 5 & 4 & 0 \end{pmatrix}$$

Figure 5.3: *An example of the same loading profile coming from two different job lists (A and B) that produces two different loading profiles when a new arrival occurs (in C)*

2.1 known as *Partially Observable MDP* (POMDP), see (Kaelbling et al., 1996) and (Kaelbling et al., 1998). An agent in a POMDP takes actions and receives reward, as in an MDP, but the agent never directly sees the identity of the current state. Rather, the agent has access only to the current observation which complicates the problem of learning, see Figure 5.7.

A POMDP has an underlying SMDP (the real environment where the agent is situated), but the POMDP (the environment as perceived by the agent) itself is not Markovian. A new observation does not only depend on the previous observation and action, but on the preceding experience. It could happen that the same observation and the same action but coming from two different past histories, always lead to two different new observations. Figure 5.3 shows an example of the same loading profile coming from two different job lists (cases A and B) that produces two different loading profiles when a new arrival occurs (in C) that fits in the capacity profile. The acceptance of the order arriving in C will always produce two different loading profiles for cases A and B, Figures 5.4 and 5.5 illustrate the build up of both loading profiles.

"The most naive strategy for dealing with POMDP is to ignore it. That is

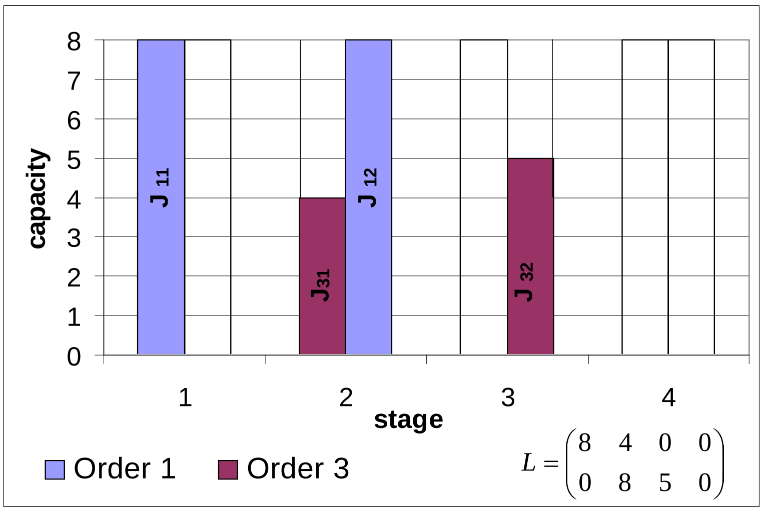


Figure 5.4: *Building up Loading Profile L when in case A the new arrival from case C is loaded*

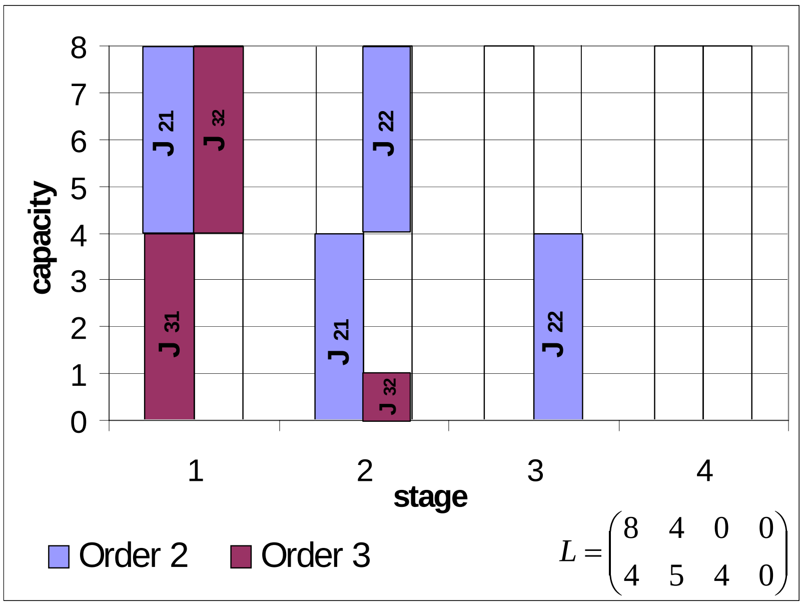


Figure 5.5: *Building up Loading Profile L when in case B the new arrival from case C is loaded*

to treat the observations as if they were the states of the environment and try to learn to behave", (Kaelbling et al., 1996). Furthermore practice has shown that:

- Small violations of the Markovian properties are well handled by Q-learning algorithms, see (Kaelbling et al., 1996).
- When the RL-agent handles information about a combination of features of the state it is able to find a better policy than heuristic agents that only consider a few features, see (Riedmiller & Riedmiller, 1999).

There are other ways to deal with POMDP though, see (Kaelbling et al., 1998),(Kearns et al., 2000), (Bakker et al., 2002), (Bakker, 2002), (Bakker et al., 2003). In general they work with an internal memory of previous experiences (observations and actions) for the RL-agent that helps to disambiguate the current state. But this implies a more complex structure for the learning procedure.

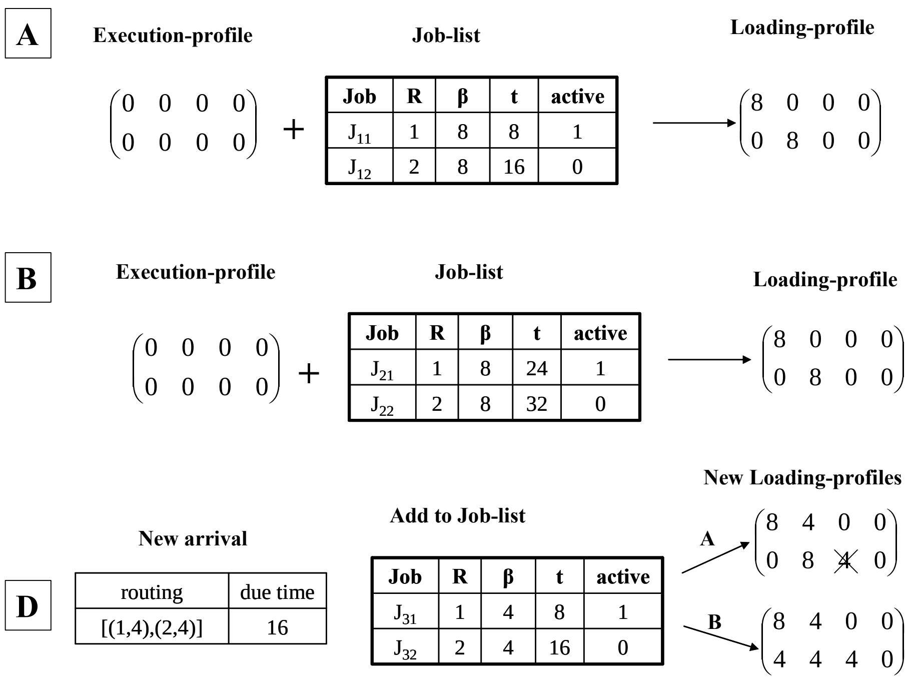
Therefore we decided to use (kr, L) as a set of features of the state to feed to the RL-agent. We believe the set of features we use constitutes a rather good observation. Note that in building kr we need the set of possible actions which incorporates knowledge from the job list and the execution profile that is not in the loading profile. See Figure 5.6 for an example in which the same loading profile in two different cases (A and B) may lead to two different sets of possible actions. In our opinion this is an intermediate approach between the naive and the more sophisticated approaches for solving POMDP.

Note that the features are only used as the input vector to the RL-agent. The transition of the sates is made as in the SMDP discussed in Section 5.2, keeping all the information of the states. Now the agent-environment interaction is not as in Figure 2.1 but as in Figure 5.7. In this case the agent is working with a different perspective of the environment, its sensors are different. It can be thought of as an observer which translates the state into observations for the agents.

The behavior of the RL-agent is according to an $\epsilon-$greedy exploration rule as explained in Chapter 2: with probability $1-\epsilon$ the agent chooses a greedy action (the action a from $A(obs)$ which maximizes $Q(obs, a)$) and with probability ϵ a random action. The parameter ϵ decreases over time according to the following rule $\epsilon_t = \frac{\epsilon_0}{1+\frac{t}{T_\epsilon}}$.

As for the knowledge structure, the agent uses an $I - \theta - 1$ perceptron to store the $Q-$values as discussed in Section 2.2.3. The input of the perceptron is a codification of the observation-action pair (obs, a), in the following way:

- n integer inputs showing the numbers of orders of each type $(kr_1, ..., kr_n)$.



A

$$\begin{pmatrix}0&0&0&0\\0&0&0&0\end{pmatrix} +$$

Job	R	β	t	active
J_{11}	1	8	8	1
J_{12}	2	8	16	0

$$\longrightarrow \begin{pmatrix}8&0&0&0\\0&8&0&0\end{pmatrix}$$

B

$$\begin{pmatrix}0&0&0&0\\0&0&0&0\end{pmatrix} +$$

Job	R	β	t	active
J_{21}	1	8	24	1
J_{22}	2	8	32	0

$$\longrightarrow \begin{pmatrix}8&0&0&0\\0&8&0&0\end{pmatrix}$$

D

routing	due time
[(1,4),(2,4)]	16

Job	R	β	t	active
J_{31}	1	4	8	1
J_{32}	2	4	16	0

$$A \nearrow \begin{pmatrix}8&4&0&0\\0&8&\cancel{4}&0\end{pmatrix}$$

$$B \searrow \begin{pmatrix}8&4&0&0\\4&4&4&0\end{pmatrix}$$

Figure 5.6: *A new arrival with a tight due-time comes in (D). In the case A it is not possible to accept it and in case B it is possible*

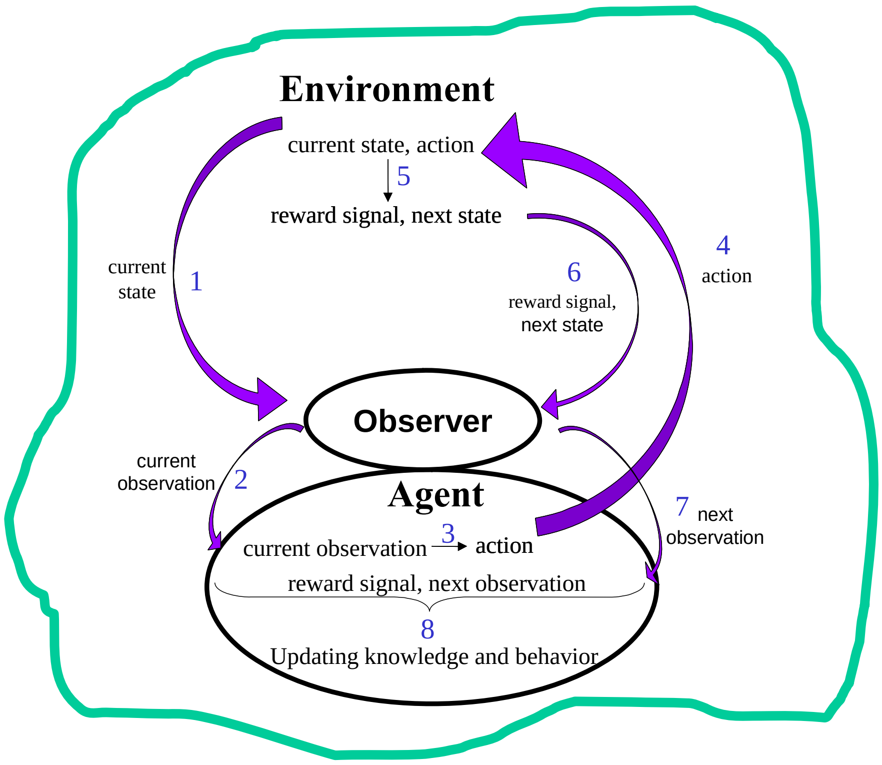


Figure 5.7: *Agent-environment interaction in POMDP*

- MxH integer inputs showing the already allocated capacity at each of the loading profile stages $\begin{pmatrix} L_{11} & L_{12} & \cdots & L_{1H} \\ \vdots & \vdots & & \vdots \\ L_{M1} & L_{M2} & \cdots & L_{MH} \end{pmatrix}$.
- 1 integer input representing the action.

Using this codification for the input, the size of the ANN is $(n+MH+3)\theta+1$ which is a much lower dimension than the state and action space has in general, $|S\text{x}A| \geq (n+1)(Cmax+1)^{MH} \prod_{i=1}^{n}(m_i+1) \geq (n+1)2^{n+MH}$. The output of the perceptron is a single value that is the approximation of the state-action value function $Q(obs, a)$.

The goal is to maximize the expected total discounted reward. Again we use 0.9 as discount factor. Next we discuss the tuning of the parameters of the agent.

5.3.1 Tuning RL-agents

In Section 2.3.2 we defined a general *learning schedule* with six parameters $(\theta, T, \alpha_0, \epsilon_0, T_\alpha, T_\epsilon)$. This schedule defines the number of hidden neurons (θ), the number of iterations of the training process (T), the initial value of the learning and exploration rate (α_0, ϵ_0), and the parameters for the learning and exploration rates decreasing functions $(T\alpha, T\epsilon)$. In our implementations these parameters have been chosen through experimental experience and are by no means optimized. However from the numerical experiments with the different problems discussed in Chapters 3 and 4, we observed the following regularities that may help to set guidelines for the tuning of the parameters, see also Appendix B.

1. A satisfactory behavior of the RL-agents was obtained for certain θ, using $\alpha_0 = 10^{-3}$, $\epsilon_0 = 1$, $T = \mu 10^4$, $T_\alpha = \mu 10^3$, $T_\epsilon = \nu 10^3$ for a certain value μ, and $\nu \in [1, ..., \mu]$. We call this combination of parameters the (θ, μ)-scheme when we take the ν that achieves the best observed performance, and use the notation $AR\,(\theta, \mu)$ for the Average Reward (AR) of a trained agent using a (θ, μ)-scheme. By a satisfactory behavior we mean that the $AR\,(\theta, \mu)$ approximates or outperforms the AR of the best known heuristic.
2. For a given θ, $AR\,(\theta, \mu)$ is monotonically increasing with respect to μ, up to a certain value $\mu_1(\theta)$ from where it starts to oscillate.
3. For a given μ, $AR\,(\theta, \mu)$ is monotonically increasing with respect to θ, up to a certain value $\theta_1(\mu)$ from where it starts to decrease.

4. The values $\mu_1(\theta)$ and $\theta_1(\mu)$ depend on problem properties.

The reason for regularity 2 could be that increasing training (iterations, learning, exploration), allows the RL-agent to gather more information obtaining a better performance. But when the knowledge structure (ANN) is not enough for that information, the agent could be overtrained and exploratory actions could deteriorate performance causing the oscillations.

On the other hand, the reason for regularity 3 could be that when increasing the size of the ANN, we increase the potentiality of the ANN as a function approximator, allowing the agent to make a better use of the gathered information, and improve performance. But increasing the size of the ANN also requires extra effort to adjust all the weights, so it could be that there is a limit to the size of the ANN for which the gathered information is not sufficient.

On the basis of regularity 1 we have reduced the *learning schedule* with six parameters $(\theta, T, \alpha_0, \epsilon_0, T_\alpha, T_\epsilon)$ to a *(θ, μ, ν)-learning schedule* with three parameters. In addition, on the basis of all the observed regularities, we have designed a methodology to tune the three parameters (θ, μ, ν). The idea is, starting with small values for parameters θ and μ, to increase them until the RL-agent using the (θ, μ)-scheme outperforms the best known heuristic (i.e., $AR(\theta, \mu) > AR_{Heu}$). For a fixed θ, we increase μ until the $AR(\theta, \mu)$ starts to decrease or gets greater than AR_{Heu}. If $AR(\theta, \mu)$ is not yet greater than AR_{Heu} we increase θ and start to increase μ again. For a fixed θ and μ we use the (θ, μ)-scheme, which finds the best exploration parameter ν under this scheme. To find the best ν we do not try all the possible values in $[1, ..., \mu]$ but we start with the value of the best exploration parameter obtained before. This is because when we start a new (θ, μ)-scheme with a higher learning structure or more learning transitions, it does not make sense to explore less than in the previous (θ, μ)-scheme. Although this tuning process may stop when the RL-agent outperforms the best known heuristic we should also add other stopping criteria. It could happen for a specific problem that the heuristic at hand gives already a good policy. Algorithm 5.2 sketches this tuning procedure in more detail.

The increase-steps for the parameters θ and μ are chosen on the basis of experience, we found out that a larger step-size for θ leads us to a more radical change in performance but it also requires more training iterations. In the figure, $QL(\theta, \mu_i, \nu)$ represents the Q-Learning using the *(θ, μ, ν)-learning schedule*, it returns the Average Reward of the trained RL-agent after 20000 iterations (AR_{20000}), which is our performance measure.

```
$i \leftarrow 0, \nu_i \leftarrow 0, \theta_0 \leftarrow 0, \mu_0 \leftarrow 0$
$\Delta\theta \leftarrow 50, \Delta\mu \leftarrow 1$
**repeat**
  $\theta \leftarrow \theta + \Delta\theta$
  **repeat**
    $i \leftarrow i + 1, \mu_i \leftarrow \mu_{i-1} + \Delta\mu$
    **for** $\nu = \nu_i$ to $\mu_i$ **do**
      $AR(\nu) \leftarrow QL(\theta, \mu_i, \nu)$
    $(AR(\theta, \mu_i), \nu_i) \leftarrow \max(AR(\nu_i : \mu_i))$
  **until** $AR(\theta, \mu_i) < AR(\theta, \mu_i - 1)$ or $AR(\theta, \mu_i) > AR_{HEU}$
  **if** $AR(\theta, \mu_i) < AR(\theta, \mu_i - 1)$ **then**
    $i \leftarrow i + 1$
**until** $AR(\theta, \mu_i) > ARHeu$ or STOP
```

Algorithm 5.2: *Tuning RL-agent to outperform a known AR_{HEU}*

5.4 Experimental results

In this section we present the application of QL, as discussed in Chapter 2, to two cases of the OA in the job-shop model presented in this chapter. The first case uses deterministic processing times and the second case is a version of the first case with stochastic processing times.

In both cases we compare the results of the RL-agent against the general one-parameter-heuristics as presented in Section 4.3. The heuristics *OrderQuality(b)* and *CapacityLevel(ρ)* here are the analogues of those defined for the single resource case. The only difference is in the parameter ρ, which here is a matrix $P = (\rho_{ij})$ that indicates the maximum level of capacity that can be occupied in resource j after the acceptance of an order of type i. The processing time for orders of type i as used in previous models is here the sum of the processing times of all the jobs of the order, $w_i = \sum\limits_j \beta_{ij}$. The heuristic *CapacityLevel*(P) is then defined as follows:

Heuristic: Capacity level

- allowed actions for each state $s \in S$:

$$A^2(s) = \{i \in A(s) \backslash \{0\} \mid cap_j(s,i) \leq \rho_{ij} HC_{\max},\ j = \overline{1..M}\} \bigcup \{0\}$$

- ordering for all actions in A:

$$i_{n+1} = 0,\ i \succ j \text{ if } \left(\frac{r_i}{w_i} > \frac{rj}{wj}\right) \mathit{or} \left(\frac{r_i}{w_i} = \frac{rj}{wj} \mathit{\ and\ } r_i > r_j\right).$$

Here $cap_j(s,i)$ denotes the total occupied capacity in resource j after accepting an order of type i when in state s. $HC_{\max}$ is the total capacity in the planning horizon.

All these heuristics use partial information about the problem like rewards, capacity request, capacity profile; they do not take into account the arrival rates, and specific routings for example. Defining heuristics that consider all these factors is a difficult task. Unlike these heuristics, the RL-agent has a learning mechanism, which we expect to be useful extracting the implicit information from the state transitions in order to learn a good OA-policy.

For a trained RL-agent that was using the *(θ,μ,ν)-learning schedule* we use the notation: $NN\theta A\mu B\nu C$, where $\theta = A$ is the number of hidden neurons; $\mu = B$ defines the training period and the learning rate as follows: $T = \mu 10^4$, $\alpha(t) = \frac{\mu}{10^3\mu+t}$; and $\nu = C$ defines the exploration rate $\epsilon(t) = \frac{10^3\nu}{10^3\nu+t}$. For finding good parameters we use the procedure explained in Section 5.3.1. Furthermore, we apply the techniques incorporated in the framework in Section 4.4.1 in order to interpret the policy learned by the RL-agents, see Algorithm 4.3. As a performance measure we use AR_t, the total average reward accumulated per unit of time up to the time corresponding to iteration t.

5.4.1 The case with deterministic processing times

This is a case with 4 types of orders and 2 types of resources. The planning horizon is 5 days ($H = 5$), i.e., one stage in the planning horizon spans 24 hours. There are 8 working hours a day ($C_{max} = 8$). All types of orders have 2 jobs, one at each resource. Table 5.2 shows the data of this case. The orders of type i follow a Poisson arrival with parameter λ_i, which indicates the expected amount of arrivals in between decision moments. The routing $\sigma_i = [(R_{ij}, \beta_{ij})]$ indicates the resource and processing time needed by the job j in the order i. The table also indicates the reward upon acceptance r_i, the due-time t_i, and the maximum number of arrivals per day m_i for each type of order. Decisions should be taken daily, on which orders to accept from the pool of orders that have arrived in the last 24 hours. Orders that are not accepted in that day will go out of the system. The order type 1 is the least frequent but the more profitable one, so a good policy might always accept this type of order. The orders of type 2 and 3 have the same ratio of profitability (2 units of reward per unit of total processing time) and the same arrival rate, but order type 3 has smaller processing time, and a shorter due-time. The order type 4 is the least profitable order, but since it has the smallest processing time, and it is the most frequent, it could be convenient to accept it sometimes.

Note that in this problem the components of the capacity profile would take only the values 0, 4 and 8 because of the values of the processing times. Therefore the cardinality of the observation space is potentially $10 * 3^{13} = 15\,943\,230$.

i	λ_i	$\sigma_i = [(R_{ij}, \beta_{ij})]$	r_i	t_i	m_i	*Capacity Profile*
1	0.5	$[(2,12),(1,12)]$	96	120	2	$H = 5$
2	1	$[(1,16),(2,8)]$	48	120	2	$M = 2$
3	1	$[(2,8),(1,8)]$	32	72	2	$C_{max} = 8$
4	2	$[(1,4),(2,4)]$	8	120	9	

Table 5.2: *Description of the deterministic case in the job shop model*

Heuristics

The most profitable order is the order of type 1 with four units of reward per total capacity; and since there is no stochasticity on the order realization, there is no reason not to accept orders of type 1 when capacity is available, no other better order would come in the future. Thus a good policy should always accept orders of type 1. It is not the case with the other order types which are more frequent but receive lower reward per capacity. We consider for simplicity *CapacityLevel*(ρ) policies with the same safety level for both resources. In the following we named these policies *CapacityLevel*$(1, \rho)$, where ρ is the level of capacity that could be used for the order types other than order type 1. Table 5.3 shows the heuristics that consider level of safety 1 for order type 1 and 0.9, 0.8, etc. for the rest of the orders. That means that when capacity is available, always an order of type 1 is accepted.

scenario	AR_{20000}	scenario	AR_{20000}
CapacityLevel$(1, 0.9)$	22.82	*CapacityLevel*$(1, 0.4)$	**46.48**
CapacityLevel$(1, 0.8)$	28.51	*CapacityLevel*$(1, 0.3)$	44.65
CapacityLevel$(1, 0.6)$	44.13	*CapacityLevel*$(1, 0.2)$	44.31
CapacityLevel$(1, 0.5)$	46.46	*CapacityLevel*$(1, 0.1)$	43.04

Table 5.3: *Heuristics CapacityLevel(1,ρ) for the deterministic case in the job-shop model*

The best heuristic is *CapacityLevel*$(1, 0.4)$ which always accepts orders of type 1 but accepts the others only if capacity is kept under 0.4 of the total capacity on each resource. That means that in a planning horizon of 40 working hours only 16 hours are occupied when accepting orders of type different from 1.

RL-agents

Table 5.4 shows results of some RL-agents trained for this case. RL-agents are trained sequentially according to the (θ, μ, ν)-methodology described in Section 5.3.1. The table only shows the results of the RL-agents for the best value of

ν for each pair (θ, μ).

scenario	AR_{20000}	scenario	AR_{20000}
$NN\theta150\mu6\nu6$	27.76	$NN\theta350\mu14\nu14$	42.27
$NN\theta200\mu8\nu6$	43.34	$NN\theta400\mu16\nu16$	45.65
$NN\theta250\mu11\nu10$	43.66	$NN\theta450\mu22\nu22$	**46.17**
$NN\theta300\mu12\nu10$	43.67		

Table 5.4: *Results of RL-agents in the deterministic case in the job-shop model*

The best RL-agent ($NN\theta450\mu22\nu22$) approximates *CapacityLevel*$(1, 0.4)$ with a 0.66% of error in the average reward.

Learning the RL-policy

Here we analyze the policy learned by the RL-agent $NN\theta450\mu22\nu22$ using 20000 iterations from this trained agent. Unlike in the previous model, here we have simplified the decision making by given to the agent the set of possible actions (we do this by means of the feature k_r), so there are transitions where rejection is the only possible action, which we call trivial decision. To analyze the policy we would not take into account the trivial decisions. The trivial iterations are easily filtered out by a quick scan of the total list of 20000 iterations. For this case we have 17546 useful iterations.

Using the simple data mining technique we explained in Section 4.4.1 we obtain the general preference relation for the zero level of the decision tree $\phi_0 = (1, 3, 2, 0, 4)$ with 1757 errors, which makes 90% of quality. The matrix of selected/possible actions A_θ is shown in Table 5.5.

Selected Actions	**Possible Actions**				
	1	3	2	0	4
1	4283	1604	1251	4203	3689
3	6	2412	1140	2412	2080
2	14	229	1073	1073	903
0	1	640	436	9220	9082
4	1	81	67	558	558

Table 5.5: *Selected action against possible actions in 17546 iterations of the RL-agent in case 1 of the job-shop model*

Giving higher priority to action zero (rejection) than to action 4 means that action 4 would never be taken following a heuristic *Priority*(ϕ_0) defined by that preference relation. In any case the type of order 4 is the least profitable type of order. Although such a heuristic makes an average reward of 46.735, which is the best so far, it does not take into account the capacity profile.

Taking into account information on the capacity profile is more complex in this model. Branching according to the distribution of capacity on each resource is very expensive computationally since the number of combinations of such distributions in this case could be up to 3^{10} (5 stages per each of the 2 resources, each taking values $0, 4$ or 8). Therefore we only consider the total amount of loaded capacity at each resource respectively. Recall that L is a MxH matrix representing the loading profile according to the tentative loading plan, so we define the total amount of loaded capacity in resource i by $TL_i = \sum_{j=1}^{H} L_{ij}$.

Table 5.6 describes the first level of the decision tree with eleven nodes. These nodes were obtained after a preprocessing phase during which we grouped together observations with the same preference relation. The columns specify the number of the node, the description of the node, the preference relation, the number of transitions covered by the node, the number of errors, and the percentage of the error. In general we denote by Heu$k$NNθ the k-th heuristic obtained from the trained RL-agent with θ hidden neurons. We call Heu1NN450 to refer to the heuristic described by the rules in Table 5.6.

N_0	(TL_1,TL_2)	Preference relation	# of cases	errors	% errors
1	$TL_1 \leq 4$	1,3,4,2,0	340	60	17.6
2	$TL_1 = 8 \wedge TL_2 \leq 8$	1,3,2,0	638	92	14.4
3	$TL_1 = 8 \wedge TL_2 \geq 12$	1,2,0	11	0	0
4	$TL_1 = 12 \wedge TL_2 \leq 12$	1,3,2,0	2161	312	14.4
5	$TL_1 = 12 \wedge TL_2 \geq 16$	1,0	10	1	10
6	$TL_1 = 16 \wedge TL_2 \neq 8$	1,3,0	1817	242	13.3
7	$TL_1 = 16 \wedge TL_2 = 8$	1,3,2,0	700	13	1.8
8	$(20 \leq TL_1 \leq 24) \wedge TL_2 \leq 8$	1,3,0	2030	141	6.9
9	$(20 \leq TL_1 \leq 24) \wedge TL_2 = 12$	1,3,2,0	3141	128	4.1
10	$(20 \leq TL_1 \leq 24) \wedge TL_2 \geq 16$	1,0	1641	144	8.8
11	$TL_1 \geq 28$	1,0	5057	104	2

Table 5.6: *Branching the $NN\theta450\mu22\nu22$-policy by total capacity per resource. It shows the priority policy and the errors with respect to the $NN450\mu22\nu22$ of these priorities for the different combinations of capacity. We call this policy Heu1NN450*

This heuristic has quality 93% with respect to the policy learned by the RL-agent $NN\theta450\mu22\nu22$ and makes an average reward of 47.97. The policy learned by the RL-agent is not an optimal policy as we may see, but the idea is that it is approaching a good policy during the training procedure while its potentiality as an approximator is increased with the increasing number of hidden neurons, and the length of the training period. That means that some learned decisions may be wrong, mainly related to those infrequent states, as for example in the states where orders of type 2 are accepted for high levels of TL_1 and TL_2. Table 5.7 shows, besides the previous agents $NN\theta450\mu22\nu22$, and Heu1NN450, a collection of other agents following heuristics obtained from Heu1NN450 by successive elimination of the acceptance of orders of type 2 in cases with high levels of TL_1 and TL_2.

Agent	Description	AR_{20000}	% Error
$NN\theta450\mu22\nu22$		46.17	0.0
Heu1NN450	Table 5.6	47.97	7.0
Heu2NN450	in Table 5.6 at row 9 eliminate action 2	48.53	7.5
Heu3NN450	in Table 5.6 at rows 9 and 7 eliminate action 2	49.19	8.4
Heu4NN450	in Table 5.6 at rows 9,7 and 3 eliminate action 2	49.20	8.4

Table 5.7: *Results from the the RL-agent and the heuristics obtained from an analysis of the policy learned by the RL-agent in the deterministic case of the job-shop model*

The best result is obtained with Heu4NN450 which has quality 91.6% with respect to the policy learned by the RL-agent $NN\theta450\mu22\nu22$ and an average reward of 49.2 which is 6.56% higher than the average reward of the RL-agent. We could continue defining other heuristics, but it seems to be a general learned rule that orders of type 4 are only accepted when capacity utilization is very low, and only if there are no orders of type 1 and 3. For high levels of capacity utilization, only orders of type 1 are accepted. In general orders of type 3 are preferred over orders of type 2.

Table 5.8 shows the results of three one-way ANOVA tests between the *CapacityLevel*$(1, 0.4)$,the RL-agent $NN\theta450\mu22\nu22$, and the heuristic Heu4NN450, each simulated using five independent samples of this case. The table shows significant evidence against the equality of the performances obtained by using the three policies.

Agents	F	Pr
CapacityLevel$(1, 0.4)$+$NN\theta450\mu22\nu22$	7.38	0.0264
CapacityLevel$(1, 0.4)$+Heu4NN450	1416.58	$2.72\text{x}10^{-10}$
$NN\theta450\mu22\nu22$+Heu4NN450	2172.12	$4.96\text{x}10^{-11}$

Table 5.8: *Results of one-way ANOVA tests between the CapacityLevel(1,0.4), the RL-agent $NN\theta450\mu22\nu22$, and the heuristic Heu4NN450 in the deterministic case of the job-shop model*

5.4.2 The case with stochastic processing times

This is the same case as before, but the processing times follow a discrete distribution with three parameters (v, a, p) such that the expected value is $v = \beta_{ij}$ as shown in Table 5.9. Parameter a determines the deviation from the expected processing time of a job and parameter p the probability of such deviation.

value	prob
$v - \alpha$	$2p$
v	$1 - 3p$
$v + 2\alpha$	p

Table 5.9: *Discrete distribution for the processing times in the stochastic case of the job shop model*

Table 5.10 shows the data of this case. The jobs in an order have the same parameters a and p for their processing time distribution. The most profitable order (type 1) is the one with higher deviation in the processing time of the jobs ($a = 8$, $p = 1/4$). And the least profitable order does not suffer any deviation in the processing time of its jobs.

$order_i$	λ_i	*routing* $[(R_{ij}, \beta_{ij})]$	time distribution		r_i	dt_i	m_i	*Capacity Profile*
			α	p				
1	0.5	$[(2, 12), (1, 12)]$	8	1/4	96	120	2	$H = 5$
2	1	$[(1, 16), (2, 8)]$	2	1/8	48	120	2	
3	1	$[(2, 8), (1, 8)]$	1	1/16	32	72	2	$C_{max} = 8$
4	2	$[(1, 4), (2, 4)]$	0	0	8	120	9	

Table 5.10: *Description of the stochastic case in the job shop model*

Note that in this problem, due to the stochasticity of the processing times, the components of the capacity profile would take all the possible values in $[0, ..., 8]$. Therefore the cardinality of the observation space is potentially $10 * 3^{23} = 941\,431\,788\,270$.

Heuristics

Table 5.11 shows the results from the heuristics.

The best heuristic is *CapacityLevel*$(1, 05)$ which always accepts orders of type 1 and acts greedily with respect to the orders keeping a 50% of the capacity free at each resource.

scenario	AR_{20000}	scenario	AR_{20000}
greedy	21.02	$CapacityLevel(1,05)$	**37.12**
random	20.79	$CapacityLevel(1,04)$	36.90
$CapacityLevel(1,09)$	22.06	$CapacityLevel(1,03)$	35.27
$CapacityLevel(1,08)$	25.92	$CapacityLevel(1,02)$	34.91
$CapacityLevel(1,07)$	34.28	$CapacityLevel(1,01)$	33.19
$CapacityLevel(1,06)$	36.17		

Table 5.11: *Heuristics for the case with stochastic processing times in the job-shop model*

RL-Agents

First we run the best RL-agent trained in the deterministic case which was the $NN\theta450\mu22\nu22$. Then we train an RL-agent with the same learning schedule for this case. To differentiate it from the RL-agent in the deterministic case, we add at the end of the name, the suffix -spt (from stochastic processing time). Table 5.12 shows the results of the agents in this case including the best heuristic from the deterministic case (Heu4NN450).

scenario	AR_{20000}
$NN\theta450\mu22\nu22$	37.45
$Heu4NN450$	38.21
$NN\theta450\mu22\nu22spt$	35.03
$NN\theta550\mu20\nu20spt$	37.22

Table 5.12: *Results of RL-related agents in the stochastic case in the job-shop model. The first agent was trained in a deterministic environment with the expected processing times (as in case 1), and the second agent is the best of the heuristics obtained also in the deterministic environment. The last two agent are RL-agents trained for this case with stochastic processing times*

Learning the RL-policy

Here we analyze the policy learned by the RL-agent $NN\theta550\mu20\nu20spt$ using 17171 non-trivial iterations (where rejection was not a trivial decision). Using the simple data mining technique we explained in Section 4.4.1 we obtain the general preference relation for the zero level of the decision tree $\phi_0 = (1,3,2)$ with 3012 errors, which makes 82.5% of quality. The matrix of selected/possible actions A_{θ_0} is shown in Table 5.13.

Taking into account information on the capacity profile is more complex in this case. Remember that the total amount of occupied capacity at each resource could get any value in [0,...,40]. Table 5.6 describes the first level of

Selected	**Possible Actions**				
Actions	1	3	2	0	4
1	3693	1465	1143	3693	3166
3	1	2221	914	2221	1907
2	80	536	1635	1635	1404
0	0	617	271	8000	7813
4	0	193	137	1622	1622

Table 5.13: *Selected action against possible actions in 17546 iterations of the RL-agent in the stochastic case of the job-shop model*

the decision tree branching according to the total amount of occupied capacity TL_1 and TL_2 at each resource respectively. We call the heuristic described by these rules Heu1NN550SPT.

This heuristic has quality 84.2% with respect to the policy learned by the RL-agent $NN550\mu20\nu20$ and makes an average reward of 38.01. The errors are due to the differentiation that the RL-agent learns according to the distribution of capacity. Branching according to the distribution of capacity on each resource is very expensive computationally, since the number of combinations of such distributions in this case could be up to 3^{20} (5 stages per each of the 2 resources, each taking 9 values). Table 5.15 shows both these agents and the collection of other heuristics obtained from a consecutive analysis of Heu1NN550SPT.

The best result is obtained with Heu3NN550SPT which has an Average Reward of 38.24 and differs from the RL-policy in 17.8% of the cases.

Table 5.16 shows the results of three one-way ANOVA tests between the *CapacityLevel*(1, 0.4),the RL-agent $NN\theta450\mu22\nu22$, and the heuristic Heu4NN450, each simulated using five independent samples of this case. The table shows significant evidence against the equality of the performances obtained by using the three policies.

5.5 Conclusions

This chapter discussed the application of RL for OA in a job-shop model. Based on experimental results from previous chapters we presented a learning scheme to train RL-agents using a reduced set of parameters. Here we analyzed two cases, one with deterministic processing times, while the second is the same problem but with stochastic processing times. In both cases we train RL-agents until we find a good approximation to the best known heuristic.

Using simple data mining we could gain some insight into the learned RL-policies and approximate these policies by sets of logical rules which are easier

N_0	TL_1	TL_2	preference relation	# of cases	errors
1	[0,4)	[0,8)	1,2,3,4	589	144
2	[0,4)	≥8	1,4,0	29	10
3	[4,8)	[0,4)	1,2,3,0	162	12
4	[4,8)	≥4	1,2,0	247	46
5	[8,12)	[0,4)	1,3,2,4	362	37
6	[8,12)	≥4	1,3,2,0	817	204
7	[12,16)	[0,16)	1,3,2,0	2831	612
8	[12,16)	≥16	1,3,0	251	106
9	[16,20)	[0,20)	1,3,2,0	2422	302
10	[16,20)	≥20	1,2,0	126	27
11	[20,24)	[0,12)	1,3,0	900	52
12	[20,24)	[12,16)	1,3,2,0	778	16
13	[20,24)	[16,24)	1,2,3,0	993	132
14	[20,24)	[24,28)	1,2,0	65	8
15	[20,24)	≥28	1,0	15	2
16	[24,28)	[4,12)	1,3,0	540	35
17	[24,28)	[12,16)	1,3,2,0	649	60
18	[24,28)	[16,20)	1,0	756	72
19	[24,28)	[20,24)	1,2,30	618	123
20	[24,28)	[24,28)	1,3,2,0	287	62
21	[24,28)	≥28	1,2,0	50	11
22	[28,32)	[0,12)	1,0	299	38
23	[28,32)	[12,16)	1,3,0	312	142
24	[28,32)	≥ 16	1,0	1359	216
25	≥ 32	[0,36)	1,0	1702	267

Table 5.14: *Branching the $NN\theta 550\mu 20\nu 20SPT$-policy by total capacity per resource. We call this policy Heu1NN550SPT*

to interpret than the RL-policies. When analyzing these approximate policies, we observe relations between resource capacity profiles and preference relations for the order types. Furthermore there are some infrequent states for which the learned decision seems to be illogical. Modifying these decisions we develop new heuristics that improve the RL-agent's results.

The case with stochastic processing times is more expensive computationally, due to the increase in the state space and complexity of the transitions. In our experiments we observe that the RL-agents trained with the expected value of the processing times (i.e., $NN\theta 450\mu 22\nu 22$), obtain better results that an equivalent (same parameter settings) RL-agent trained with the stochastic processing times (i.e., $NN\theta 450\mu 22\nu 22spt$).

Agent	Description			AR_{20000}	% Error
RL	$NN\theta550\mu20\nu20spt$			37.22	0.0
Heu1NN550SPT	Table 5.14			38.01	15.8
Heu2NN550SPT	same as Heu1NN550 but for $C_1 \geq 24$ use priority-rule=(1,0)			38.03	16.9
Heu3NN550SPT	same as Heu2NN550 but			38.24	17.8
	C_1	C_2	pref. order		
	[12,16)	[16,20)	1,2,0		
	[12,16)	≥20	1,0		
	[16,20)	[12,20)	1,3,0		
	[12,16)	≥20	1,0		
	[20,24)	[12,24)	1,3,0		
	[20,24)	[24,28)	1,0		

Table 5.15: *Results from the RL-agent and the heuristics obtained from an analysis of the policy learned by the RL-agent in the stochastic case of the job-shop model*

Agents	F	Pr
$CapacityLevel(1, 0.5) + NN\theta550\mu20\nu20sp$	23.63	0.0013
$CapacityLevel(1, 0.5)+$ Heu3NN550SPT	795.75	2.69x10^{-9}
$NN\theta550\mu20\nu20sp+$ Heu3NN550SPT	794.27	2.71x10^{-9}

Table 5.16: *Results of one-way ANOVA tests between the Caplev(1,0.5), the RL-agent $NN\theta450\mu22\nu22$, and the heuristic Heu3NN550SPT in the stochastic case of the job-shop model*

Chapter 6

OA involving additional decisions

In this chapter we consider Reinforcement Learning supporting Order Acceptance together with other kinds of decisions in an integrated planning approach. The idea is to show how RL may also be useful in more general and realistic decision making settings.

We consider three types of problems. The first type of problem considers choosing routes for each order in an open job shop, the second one considers outsourcing decisions that may outsource a specific job of an order, and the last one includes a due-time and price negotiation with the customer.

In Section 6.1 we present the description of the problems and in Section 6.2 the SMDP approach. In Section 6.3 we discuss the application of RL to these problems and the experimental results are shown in Section 6.4. Finally in Section 6.5 we draw some conclusions.

6.1 Description of the problems

The three problems have a similar description since we use the same integrated simulation environment as in Chapter 5. The main differences are on the description of the arriving orders, the type of decisions, and the consequence of the decision. First we briefly comment on the similarities, which can be found more extensively in Chapter 5. Then we explain the particularities of each specific problem.

There is a finite number n of order types that arrive continuously over time but are collected in batch every fixed period of time. Each batch arrival is processed for acceptance. Each order i consists of a set of n_i jobs and is denoted by $Oi = \{J_{i1}, , J_{in_i}\}$. For simplicity reasons we consider in this chapter that every order consists of at most two jobs ($n_i \leq 2$), deterministic processing times, and two types of resources in the shop floor.

Also here the decision is made sequentially, choosing orders one by one during a *time off.* Besides choosing an order for acceptance, another decision must be taken which is different for each type of problem: choosing a route in the first problem, deciding about outsourcing in the second, and choosing a due-time option in the third. Once a batch of orders is finally accepted (the word "finally" is used here to emphasize that it is after the negotiation with the customer in the third type of problem) we have a set of orders with the same characteristics as in Chapter 5. These accepted orders are then processed in the same way as we explained in that chapter. The EDDL procedure is also used here to support the OA decision and the resequencing problem in the shop floor.

6.1.1 Routing problem

Here we consider routing decisions in an open shop. The problem in this case is similar to the multiresource problem. But the description of the orders does not include a fixed routing and a decision has two parts: as before one is which orders to accept, and the other part is for each of the accepted orders which routing should be used.

Order Types

There is a finite number n of order types that arrive continuously over time. Each order consists of a set of jobs. The order i is characterized by the arrival rate (λ_i), due-time (t_i), processing time (β_i) and receives an immediate reward (r_i) upon acceptance. The requested capacity β_i is a vector (β_{ij}) where β_{ij} is the processing time of job j in order i . Job j in order i must be done in resource j, and the only constraint is that two jobs of the same order are not simultaneously processed.

Decision making

Every time period a decision must be taken on the collected orders. Orders are chosen one by one from the arrival list. Besides, we should choose a route for each order. Since we are only considering two resources, there are at most two routes for each type of order, depending on the capacity profile and the list of accepted jobs waiting for service. In different situations occurrences of the same type of order may be accepted with different routes, even during the same *time off* period. At the end of the decision period all accepted orders have been assigned a route, therefore the situation at this point is similar to the situation in Chapter 5.

6.1.2 Outsourcing problem

Here we consider together with the order acceptance decision, the possibility of deciding to outsource part of the order. Reasons for outsourcing might be that there is no available capacity for the whole order or -differently- you want to reserve capacity for more profitable jobs. For simplicity reasons here we consider that only the second job of an order could be outsourced, so there are two outsourcing options. Not outsourcing an order means that both jobs of the order will be processed in the shop.

Outsourcing an order means that the first job of the order will be processed in the shop and the other one will be outsourced. Furthermore there could be a limit lo on the total amount decided to outsource at each decision moment. The description of the orders includes two options for the immediate reward upon acceptance, depending on the outsourcing option.

A decision consists of two parts: as before one is which orders to accept, and the other part is whether to outsource the accepted orders or not.

Order Types

There is a finite number n of order types that arrive continuously over time. The orders are subdivided into at most two jobs. The order type i is characterized by the arrival rate (λ_i), a due-time (t_i), a routing (σ_i), and a set of options (φ_i) for the immediate reward upon acceptance. We put $\varphi_i = (r_{i1}, r_{i2})$ where r_{i1}, r_{i2} are the immediate rewards upon acceptance of only the first job or both jobs of the order, respectively. In accepting only the first job the reward r_{i1} is the total reward expected from the market minus outsourcing costs of the second job. The jobs in the order i define a fixed routing σ_i with linear precedence relations. A routing σ_i is a sequence of pairs (R_{ij}, β_{ij}) where R_{ij} is the resource and β_{ij} is the processing time of job j in the order type i.

Decision making

Every time period a decision must be taken on the collected orders. Orders are chosen one by one from the arrival list. Besides, we should choose whether to outsource or not. The possible decisions are depending on the capacity profile, the list of accepted jobs waiting for service, and the limit in the total amount to outsource every period.

Accepted orders are totally or partially accepted. Total acceptance of an order means that both jobs of the order will be processed in the shop. Partial acceptance means that the first job of the order will be processed in the shop and the other one will be outsourced.

At the end of the decision period, the situation is similar to the situation in Chapter 5.

6.1.3 Due-time and price negotiation

Here we consider together with the order acceptance decision, the selection of a due-time for the accepted orders. Associated to each due-time option there is a reward for acceptance. Furthermore there is a certain probability that the customer will agree on the selection of that option.

For simplicity reasons we consider each order with two due-time options. A decision has two parts, as before one is which orders to accept, and the other part is for each of the accepted orders which due-time to agree on. A substantial difference is that the customer could decide to quit the system.

Order Types

There is a finite number n of order types that arrive continuously over time. The orders are subdivided into at most two jobs. The order type i is characterized by the arrival rate (λ_i), a routing (σ_i), and a set of due-time options (τ_i). The jobs in the order i define a fixed routing σ_i with linear precedence relations. A routing σ_i is a sequence of pairs (R_{ij}, β_{ij}), where R_{ij} is the resource and β_{ij} is the processing time of job j in the order of type i. We consider here that each order has a set τ_i of two options for the due-time. We put $\tau_i = (\phi_{i1}, \phi_{i2})$, and the triplet $\phi_{il} = (t_{il}, p_{il}, r_{il})$ where t_{il}, p_{il}, r_{il} are the due-time, the probability that the customer will agree with the option, and the immediate reward upon acceptance and customer agreement, is the option l for the order type i. Here $l = 1, 2$.

Decision making

Every time period a decision must be taken on the collected orders. Orders are chosen one by one from the arrival list. Besides, we should choose a due-time option. The possible decisions are depending on the capacity profile and the list of accepted jobs waiting for service.

Accepted orders are only temporarily accepted. Once the decision period finishes, the customers may disagree with the chosen due-time options for their orders. In such a case the orders go out of the system as rejected orders. If a customer agrees with the selected option, the order is definitely accepted.

At the end of the decision period, after the customer's agreement or disagreement, the situation is similar to the situation in Chapter 5.

First, in the next section, we give general ideas about the SMDPs that we define to model the dynamics of the environment in which our RL-agents are situated. We define three similar but different models, one for each problem. Then we discuss some aspects of the implementations of the RL-agents.

6.2 The SMDP models

These problems can be modeled as Semi Markov Decision Problems. Since we use the same integrated simulation environment as in Chapter 5, the models are similar to the one in that chapter.

The state of the system is defined by $s = (k, C)$, where $k = (k_1, ..., k_N)$ is the order list and k_i represents the number of orders of type i requesting service. For each order type i we restrict the maximum number of orders in the order list to m_i. This number may be determined by the arrival probabilities or by the limited capacity. In the routing and the due-time negotiation problems we use the same definition of the capacity profile as in Chapter 5, $C = (EP, JL)$. $EP = (e_{jt}) \in Z^{MH}$ is the execution profile and by e_{jt} we refer to the occupied capacity in the resource j at stage t of the planning horizon. JL is a table containing information of all the jobs of the accepted orders which are still waiting to be served. The information in the job-list table includes the release-time, the due-time, the expected processing time, the resource, the predecessors and successors of each job and a field indicating if the job is active or not.

For the outsourcing problem we add another element to the capacity profile (out), to keep record of the total amount outsourced at each time period. This amount might be limited.

The action is a pair $a = (i, o)$ specifying the type of order $i = 1...n$ and the option $o = 1, 2$ for the chosen order. In the three problems there are two options for each chosen order, as is shown in Table 6.1. Furthermore we define the action $a = (0, 0)$ as rejection.

Problem	Options	
	Option 1	Option 2
Routing	route 1	route 2
Outsourcing	outsource	do not outsource
Due-time Neg.	due-time 1	due-time 2

Table 6.1: *Options for the accepted orders in the routing, outsourcing and due-time negotiation problems*

The set of allowed actions $A(s)$ for a state $s = (k, C)$ is defined as usual. Action $a = (i, o)$ is allowed if order type i is present in the order list k and capacity is available for at least one occurrence of that order type with option o. Rejection of the complete order list is always an option, i.e., $A(s) = \{(i, o) | i \in [1..N], o = 1, 2,\ k_i \neq 0, EDDL(C, i, o) \text{ is possible}\} \bigcup \{0, 0\}$.

The one step state transition from the current state s is described in Table 6.2, Table 6.3 and Table 6.4. The transitions are similar to the transitions in Chapter 5. Here the action has a new component (the option o), and the capacity profile evolution is always deterministic since we do not consider stochastic processing times. Hence, penalties due to unexpected capacity excess - as in Chapter 5 - will never occur. This simplifies the transition probabilities.

Table 6.2 describes the dynamics for the *routing* problem. The first column represents all the characteristics defining a state transition from the current state s. The other two columns represent these characteristics, given that the action i in the current state is different from rejection (second column), respectively is rejection (third column).

current state (s)	$(k, C) = (k, EP, JL)$	
action $a = (i, o)$	$i \neq 0, o \neq 0$	$(0, 0)$
next state (s')	$(k - e_i, EP, JL + O_{io})$	$(k',\ EP', JL')$
$rew(s, a)$	r_i	0
$d(s, a)$	0	1
$Pr(s, a, s')$	1	$Pr\{k'\}$

Table 6.2: *Dynamics of transition from the current state s in the routing problem*

In case an order of type i is chosen ($i \neq 0$), an immediate reward r_i is received independently of the route, and a deterministic transition to the next state s' occurs ($Pr(s, a, s') = 1$). Note that there is a *time off* at this moment, i.e., the elapsed time $d(s, a)$ is 0. The order is then removed from the arrival list so the new arrival list is given by $k - e_i$ where e_i is a unit vector with 1 in the position i. The new job-list is updated including the jobs of the accepted order with the chosen route o, and the capacity profile remains unchanged. By O_{io} we refer to an order of type i using the fixed route o.

In case the job list is rejected ($a = (0, 0)$) the decision process at the current time is finished; there is a *time on* until the next decision epoch. The new order list k' is determined by the order arrival process. The new capacity profile $C' = (EP', JL')$ is updated according to the loading plan. The immediate reward is zero. Note that the transition probability only depends on the arrival process since it is the only stochastic process going on.

Table 6.3 describes the dynamics for the *outsourcing* problem. The transitions for this problem are very similar to the routing problem. The first column represents all the characteristics defining a state transition from the current state s. The other three columns represent these characteristics given that the action a in the current state is choosing an order with outsourcing (second column), choosing an order without outsourcing (third column), respectively is rejection (fourth column).

current state (s)	$(k, C) = (k, EP, JL, out)$		
action $a = (i, o)$	$i \neq 0, o = 1$	$i \neq 0, o = 2$	$(0, 0)$
next state (s')	(k-e_i,EP,JL+J_{i1},out+w_{i2})	(k-e_i,EP,JL+O_i,out)	(k$'$, EP$'$,JL$'$,0)
$rew(s, a)$	r_{i1}	r_{i2}	0
$d(s, a)$	0		1
$Pr(s, a, s')$	1		$Pr\{k'\}$

Table 6.3: *Dynamics of transition from current state s in the outsourcing problem*

In case an order of type i is chosen ($i \neq 0$), with outsourcing ($o = 1$) or without outsourcing ($o = 2$), an immediate reward r_{io} is received and a deterministic transition to the next state s' occurs ($Pr(s, a, s') = 1$), during a *time off*, ($d(s, a) = 0$). The order is then removed from the arrival list so the new arrival list is given by $k - e_i$ where e_i is a unit vector with 1 in the position i. The new job-list is updated including only the first job of the chosen order ($JL + J_{i1}$) in case of outsourcing, and the complete chosen order ($JL + O_i$) in the case without outsourcing. Note that in this problem the capacity profile has a new component (out) which indicates the amount of capacity outsourced in the decision moment. In case an order type i is chosen ($i \neq 0$), with outsourcing ($o = 1$) the processing time w_{i2} of the second job of the order is added to the *out* component of the capacity profile. In case $o = 2$ then the component *out* remains the same.

In case the job list is rejected ($a = (0, 0)$) the transition is similar to the transition in the same case of the routing problem. Only in this case the *out* component of the capacity profile is zero so new jobs may be outsourced.

Table 6.4 describes the dynamics for the *due-time* negotiation problem. This problem behaves quite different, compared to the others, because of the possibility that the customer does not agree with the chosen action, so all the transitions are stochastic.

current state (s)	$(k, C) = (k, EP, JL)$		
action $a = (i, o)$	$i \neq 0, o \neq 0$		$(0, 0)$
next state (s')	$(k - e_i, EP, JL + O_{io})$	$(k - e_i, EP, JL)$	$(k',\ EP', JL')$
$R(s, a)$	r_{io}	0	0
$d(s, a)$	0		1
$Pr(s, a, s')$	p_{io}	$1 - p_{io}$	$Pr\{k'\}$

Table 6.4: *Dynamics of transition from current state s in the due-time negotiation problem*

In case an order type i is chosen ($i \neq 0$) with due-time option o ($o = 1$, or $o = 2$), there is a stochastic transition to the next state s' depending on whether the customer agrees with the chosen option or not, during a *time off* ($d(s, a) = 0$). The order is then removed from the arrival list, so the new arrival list is given by $k - e_i$, where e_i is a unit vector with 1 in the position i, and the execution profile remains unchanged. The customer agrees with the chosen due-time option with probability p_{io} in which case there is an immediate reward r_{io} and the new job list is updated including the chosen order with the corresponding option ($JL + O_{io}$). The customer does not agree with the chosen due-time option with probability $1 - p_{io}$ in which case there is zero immediate reward and the new job list remains unchanged.

In case the job list is rejected ($a = (0, 0)$) the transition is identical to the transition in the same case of the routing problem

In these three models the objective is to find a deterministic policy π:

$$\pi(s) = \begin{cases} (i, o) & i \in \overline{1..N},\ \ o \in \overline{1..2} \text{ selecting order type } i \text{ with option } o \\ (0, 0) & \text{reject the job-list} \end{cases}$$

which maximizes the performance of the system. We call this an OA+option policy. The performance of the system is measured as the expected value of the total discounted reward. The corresponding Bellman equation for the state value function V^π is given by:

$$V^\pi(s) = rew(s, \pi(s)) + \gamma^{d(s,\pi(s))} \sum_{s'} Pr(s, \pi(s), s') V^\pi(s'),$$

where γ is the discount factor and $d(s, i)$ is the elapsed time as introduced before. The optimal action value function Q^* in this case satisfies:

$$Q^*(s, i) = rew(s, i) + \gamma^{d(s,i)} \sum_{s'} Pr(s, i, s') \max_{i'} Q^*(s', i')$$

and an optimal policy $\pi^*(s)$ can be determined by: $\pi^*(s) = \arg\max_i Q^*(s, i)$.

6.3 Reinforcement Learning approach

Here we discuss the application of Reinforcement Learning to the problems described above. We use QL methods that aim to approximate the optimal Q-value function as discussed in Chapter 2. The RL-agent should learn an OA+option policy (i.e., OA augmented with another decision) while interacting with the environment. We focus here on traditional Q-learning using ANN, as introduced in Chapter 2.

Since here we use a similar environment as in Chapter 5, there is also the problem with the complexity of the state representation. Therefore we use here the same idea of features of the state, see Section 5.3. That means we use the set of features (kr, L) as observations of the state, that are received by the RL-agent during the interaction with the environment. Here kr is a feature of k which also contains some information about the capacity profile: it has a zero in the components corresponding to orders that do not fit in the capacity profile. The loading profile L is a feature of the capacity profile which combines information from the execution profile EP and the job list JL.

The structure of the RL-agent is also similar to the one in Chapter 5, see Section 5.3. The only difference is in adding one integer input more corresponding to the option in the chosen action. For the tuning of the parameters of the RL-agents we use here the strategy discussed in Section 5.3.1 with the *(θ, μ, ν)-learning schedule*.

6.4 Experimental results

In this section we present the application of RL to three cases, one per each type of problem presented in this chapter. In all cases we compare the results of the RL-agents against the results obtained with some heuristics from the general heuristics as presented in Section 4.3. First we discuss how we use such a general class of heuristics in these problems, then we present the cases.

All these heuristics use partial information about the problem like rewards, capacity request, capacity profile; they do not take into account the arrival rates, and specific routings for example. Defining heuristics that consider all these factors is a difficult task. Unlike these heuristics, the RL-agent has a learning mechanism, which we expect to be useful for extracting the implicit information from the state transitions in order to learn a good OA-policy.

The RL-agents are trained using the *(θ, μ, ν)-learning schedule* as explained in Section 5.3.1. Furthermore, we apply the technique explained in Section 4.4.1 in order to interpret the policy learned by the RL-agents. As a performance measure we use AR_{20000}, the total average reward accumulated per unit of time up to the time corresponding to iteration 20000.

6.4.1 Heuristics

In these three problems the decisions have two parts: choosing orders (OA), and choosing the option for each chosen order (option).

In the last two problems (outsourcing and due-time negotiation), both parts of the decision define the immediate reward upon acceptance for each order. Therefore we may use the same general heuristics as used in the previous models in order to take decisions. Remember that these heuristics are profit oriented, defining preference relations according to the reward per requested unit of capacity for a job (i.e., $\frac{rj}{wj}$ for job type j)

That is not the case in the first problem (routing) where the immediate reward is defined by the type of order independently of the route. For the OA part we can use the same general heuristics as used in the previous models, and for the routing part we choose the route that gives the minimum makespan (MM) in the loading plan.

The heuristics are *OrderQuality(b)* and *CapacityLevel(ρ),* see Section 4.3. Furthermore we include two other heuristics: GREEDY and RANDOM. The GREEDY heuristic takes the decision with highest reward per requested unit of capacity for which there is enough capacity. This heuristic is similar to *CapacityLevel(1)* or *OrderQuality(0)* in case all rewards are positive. The RANDOM heuristic, chooses at random as long as there is enough capacity.

6.4.2 Routing problem

This is a case with 4 types of orders and 2 types of resources. The planning horizon is 5 days ($H = 5$) with 8 working hours a day ($C_{max} = 8$). All types of orders have 2 jobs, one at each resource. Table 6.5 shows the data of this case. The orders of type i follow a Poisson arrival with parameter λ_i in between decision moments. The processing time vector β_i indicates the processing time needed for each job in the order i. Remember that job j in order i must be done in resource j, and the only constraint is that two jobs of the same order are not simultaneously processed. The table also indicates the reward upon acceptance r_i, the due-time t_i and the maximum number of arrivals per day m_i for each type of order. Decisions should be taken daily, on which orders to accept from the pool of orders that have arrived in the last 24 hours. Orders that are not accepted in that day will go out of the system. The order type 1 is the least frequent but the more profitable one, so a good policy might always accept this type of order. The orders of type 2 and 3 have the same arrival rate and processing time, but an order of type 2 is a little more profitable. The order type 4 is the least profitable order, but since it has the smallest processing time, and it is the most frequent, it could be convenient to accept it sometimes.

$order_i$	λ_i	β_i	r_i	r_i/β_i	t_i	m_i	*Capacity Profile*
1	1	$(32, 8)$	200	5	120	1	$H = 5$
2	3	$(16, 8)$	116	4.8	72	1	
3	3	$(8, 16)$	108	4.5	72	1	$C_{max} = 8$
4	5	$(8, 8)$	32	2	120	4	

Table 6.5: *Description of the case in the routing problem*

Note that in this problem the components of the capacity profile would take only the values 0 and 8 because of the values of the processing times and $C_{\max}$. Therefore the cardinality of the observation space is $|S| = 5 * 2^{13} = 40\,960$.

Heuristics

Table 6.6 shows results of the heuristic policies for this case. We consider for simplicity $CapacityLevel(\rho)$ policies with the same safety level for both resources and all orders, and also $CapacityLevel(1, \rho)$, policies that consider level of safety 1 for order type 1 and ρ for the rest of the orders. The Greedy policy uses safety level 1, the orders are accepted as long as capacity is available. The $OrderQuality(4)$ policy only chooses order types 1, 2 and 3 in that order. These are the three more profitable orders.

scenario	AR_{20000}	scenario	AR_{20000}
Greedy	32.04	$CapacityLevel(1, 0.9)$	32.09
$CapacityLevel(0.9)$	32.09	$CapacityLevel(1, 0.7)$	33.15
$CapacityLevel(0.7)$	32.91	$CapacityLevel(1, 0.5)$	51.77
$CapacityLevel(0.5)$	32.39	$CapacityLevel(1, 0.3)$	48.40
$CapacityLevel(0.3)$	15.94	$CapacityLevel(1, 0.1)$	47.53
Random	32.03	$OrderQuality(4)$	**64.50**

Table 6.6: *Heuristics results for the case in the routing problem*

The best heuristic is $OrderQuality(4)$. Its performance is substantially better than the other heuristics (approximately 24.6% from the best of the other heuristics).

RL-agents

Table 6.7 shows results of some RL-agents trained for this case. RL-agents are trained sequentially according to the (θ, μ, ν)-methodology described in Section 5.3.1.

scenario	AR_{20000}
$NN\theta350\mu13\nu12$	33.95
$NN\theta400\mu14\nu12$	73.94
$NN\theta450\mu15\nu14$	**74.11**

Table 6.7: *Results of RL-agents in the routing problem*

The best RL-agent ($NN\theta450\mu15\nu14$) outperforms $OrderQuality(4)$ by 14.9% and the Greedy by 131.3%.

Learning the RL-policy

Here we analyze the policy learned by the RL-agent $NN\theta450\mu15\nu14$ using 19882 non-trivial iterations (where rejection was not a trivial decision). Note that in this problem there are 9 different actions: rejection and two routes for each of the four types of orders $\{(i,o) : i = 1...4; o = 1,2\} \bigcup \{0,0\}$. Using the simple data mining technique that we explained in Section 4.4.1, we obtain the general preference relation for the zero level of the decision tree ϕ_0 =((2,1)(2,2)(3,1)(3,2)(1,1)(1,2)). Recall that we do not write in the preference relation the actions with less preference than the rejection action $(0,0)$. The matrix of selected/possible actions A_{ϕ_0} is shown in Table 6.8.

Selected Actions	**Possible Actions**								
	(2,1)	(2,2)	(3,1)	(3,2)	(1,1)	(1,2)	(0,0)	(4,1)	(4,2)
(2,1)	175	3	165	1	116	2	175	175	175
(2,2)	0	3812	0	3461	0	2428	3812	3808	3808
(3,1)	22	0	207	20	18	17	207	203	203
(3,2)	0	0	0	3704	0	116	3704	3651	3651
(1,1)	4	0	6	0	11	0	11	11	11
(1,2)	0	17	0	0	0	28	28	27	27
(0,0)	0	0	0	0	0	0	11933	11895	11933
(4,1)	0	0	0	0	0	0	7	7	7
(4,2)	0	0	0	0	0	0	5	4	5

Table 6.8: *Selected action against possible actions in 17546 iterations of the RL-agent in the routing problem*

The policy obtained from the preference relation ϕ_0 = ((2,1)(2,2)(3,1)(3,2) (1,1)(1,2)) makes 61 errors, which makes 99.7% of quality, with respect to the policy learned by the RL-agent. So ϕ_0 defines an excellent approximation to the RL-policy. In this preference relation, orders of type 2 and 3 have higher priority than orders of type 1, and there is a clear tendency not to accept orders of type 4.

As for the routes the most frequent is route 2. But every time when route 2 was chosen there was not the possibility of choosing route 1. A similar situation exists for the cases where route 1 was chosen, except for a few exceptions. So we can not really say that there is a real preference for a route. Actually setting a preference for route 1 gives a smaller error with respect to the learned policy.

Note that this policy does not give higher priority to orders of type 1, which is the more profitable one. This order introduces a disequilibrium in the resources workload, demanding much more use of resource 1. Orders type 2 and 3 have smaller profitability than order type 1 but because of the distribution of their work request, they seem to make a better use of the capacity when combined. Furthermore they have higher arrival rates.

6.4.3 Outsourcing problem

This is a case with a two-resources shop where 4 types of orders arrive according to independent Poisson arrivals. Table 6.9 shows the data of this case. The planning horizon is 6 days ($H = 6$) with 8 working hours a day ($C_{max} = 8$). The order type i is characterized by the arrival rate (λ_i) in between decision moments, a due-time (t_i), a routing (σ_i), and a set of immediate rewards upon acceptance (φ_i) for the outsourcing options. The processing times β_{ij} and the due-times t_i are given in hours. Decisions should be taken daily, on which orders to accept from the pool of orders that have arrived in the last 24 hours. Orders that are not accepted in that day will go out of the system. The maximum amount that is possible to outsource per resource every day is 16 hours, i.e., $lo = 16$.

The order type 1 is the least frequent but the more profitable one; particularly with the outsourcing option 2 (not outsourcing). For all types of orders it holds that it is more profitable not to outsource the order (outsourcing option 2). Still there may be cases where choosing to outsource (the first option) could be a better alternative in the long run. The order type 4 is the least profitable order, but since it has the smallest processing time, and it is the most frequent, it could be convenient to accept it sometimes.

$order_i$	λ_i	$\sigma_i = [(R_{ij}, \beta_{ij})]$	t_i	$\varphi_i = (r_{i1}, r_{i2})$	m_i	*Capacity Profile*
1	1	$[(1,24),(2,8)]$	96	$(132,192)$	1	
2	2	$[(1,\ 8),(2,8)]$	72	$(\ 20,\ 48)$	2	$H = 6$
3	2.3	$[(2,16),(1,8)]$	120	$(\ 45,\ 70)$	2	$C_{max} = 8$
4	4	$[(2,\ 8),(1,8)]$	144	$(\ 12,\ 32)$	5	$lo = 16$

Table 6.9: *Description of the case in the outsourcing problem*

Note that in this problem the components of the capacity profile would take only the values 0 and 8 because of the values of the processing times and C_{max}. Therefore the cardinality of the observation space is $|S| = 3^3 * 2^{14} = 442\,368$.

Heuristics

Table 6.10 shows results of the heuristic policies for this case. We consider for simplicity $CapacityLevel(\rho)$ policies with the same safety level for both resources and all orders, and also $CapacityLevel(1, \rho)$, policies that consider level of safety 1 for order type 1 and ρ for the rest of the orders. The *Greedy* policy uses safety level 1, the orders are accepted as long as capacity is available. The $OrderQuality(6)$ policy only chooses order type 1 without outsourcing (option 2) which is the more profitable action. The policy that allows to choose both options for orders of type 1 is $OrderQuality(5.5)$, but for this case it gave exactly the same result as $OrderQuality(6)$. The $OrderQuality(3)$ policy only chooses order type 1 and 2 with both types of outsourcing options.

scenario	AR_{20000}	scenario	AR_{20000}
Greedy	32.05	$CapacityLevel(1, 0.9)$	31.98
$CapacityLevel(0.9)$	31.98	$CapacityLevel(1, 0.7)$	37.46
$CapacityLevel(0.7)$	37.46	$CapacityLevel(1, 0.5)$	43.6
$CapacityLevel(0.5)$	42.93	$CapacityLevel(1, 0.3)$	38.2
$CapacityLevel(0.4)$	40.40	$OrderQuality(6)$	**53.83**
$CapacityLevel(0.3)$	31.10	$OrderQuality(5.5)$	**53.83**
Random	33.88	$OrderQuality(3)$	51.61

Table 6.10: *Heuristics results for the case in the outsourcing problem*

The best of these heuristic policies is $OrderQuality(6)$.

RL-agents

Table 6.11 shows results of some RL-agents trained for this case. RL-agents are trained sequentially according to the (θ, μ, ν)-methodology described in Section 5.3.1. The table only shows the results of the best RL-agents for some values of θ. The $NN\theta400\mu15\nu13$ RL-agent outperforms by far all the previous heuristics, $OrderQuality(6)$ by 31.97% and Greedy by 121.6%.

scenario	AR_{20000}
$NN\theta100\mu7\nu7$	40.77
$NN\theta250\mu11\nu10$	50.72
$NN\theta300\mu13\nu13$	54.07
$NN\theta400\mu15\nu13$	**71.04**

Table 6.11: *Results of RL-agents in the outsourcing problem*

Learning the RL-policy

Here we analyze the policy learned by the RL-agent $NN\theta400\mu15\nu13$ using 19758 non-trivial iterations (where rejection was not a trivial decision). Using the simple data mining technique we explained in Section 4.4.1 we obtain the general preference relation for the zero level of the decision tree $\phi_0 = ((1, 2)\,(1, 1))$. The matrix of selected/possible actions A_{ϕ_0} is shown in Table 6.12.

Selected	**Possible Actions**								
Actions	(1,2)	(1,1)	(0,0)	(2,2)	(2,1)	(3,1)	(3,2)	(4,1)	(4,2)
(1,1)	3043	3043	3043	2641	2645	2725	2725	2891	2981
(1,2)	1	4	0	0	4	0	0	4	4
(0,0)	0	0	11212	3868	3868	4311	4311	11103	11103
(2,2)	0	0	1522	1522	1522	1370	1370	1500	1500
(2,1)	0	0	48	25	48	8	8	43	43
(3,1)	0	0	2559	34	34	2559	2559	2510	2510
(3,2)	0	0	357	0	0	357	357	351	351
(4,1)	0	0	886	43	43	39	39	886	886
(4,2)	0	0	127	19	19	45	45	127	127

Table 6.12: *Selected action against possible actions in 19758 iterations of the RL-agent* $NN\theta400\mu15\nu13$ *in the outsourcing problem*

The policy obtained from the preference relation $\phi_0 = ((1,2)\,(1,1))$ makes 5500 errors, which makes 71.4% of quality, with respect to the policy learned by the RL-agent. So ϕ_0 defines a policy that only chooses orders of type 1, and it is not a good approximation of the policy learned by the RL-agent. Such a policy resembles the heuristic $OrderQuality(5.5)$.

Table 6.13 summarizes a heuristic policy that better approximates the RL-policy. The heuristic mainly considers branching according to the total amount of loaded capacity TL_1 and TL_2 at each resource respectively, but it also considers distribution of capacity according to the loading plan L and amount of orders in the order-list k in some cases. Recall from Chapter 5 that $TL_i = \sum_{j=1}^{H} L_{ij}$ where L is the MxH matrix representing the loading profile. We use the set definition $\Theta = \{k\,|k_3 = 2, k_4 \in [1,4]\}$ to describe some cases with a particular order list. The columns in the table specify the number of the node, the description of the node, the preference relation, the number of transitions covered by the node, and the number of errors.

N_0	States	preference relation	cases	errors
1	All states	$(1,2)$,$(1,1)$	3047	1
2	TL$_1$=TL$_2$=0	$(3,1)$,$(2,2)$	122	63
3	TL$_1$=0,TL$_2$=8	$(2,2)$,$(3,2)$	1515	0
4	(TL$_1$=0,TL$_2$=16) or TL$_1$=TL$_2$=8,L$_{11}$=8,L$_{21}$=0,k=(0,0,2,5) or (TL$_1$=40,TL$_2$=24,L$_{23}$=0)	$(4,2)$	75	13
5	TL$_1$=8,TL$_2$=0	$(3,1)$,$(2,1)$	15	5
6	(TL$_1$=TL$_2$=8,L$_{11}$=0) or (TL$_1$=TL$_2$=16,L$_{11}$=8,L$_{12}$=0)	$(2,2)$	20	1
7	(TL$_1$=8,TL$_2$=16,L$_{11}$=0) or (TL$_1$=TL$_2$=16,L$_{11}$=0)	$(2,2)$,$(3,2)$	174	3
8	TL$_1$=8,TL$_2$=24	$(2,2)$,$(2,1)$	30	0
9	(TL$_1$=16,TL$_2$=0)or (TL$_1$=TL$_2$=24,L$_{11}$=8,L$_{23}$=0) or (TL$_1$=32,TL$_2$=8,L$_{21}$=0) or (TL$_1$=32,TL$_2$=24,L$_{23}$=8)	$(4,1)$	668	3
10	(TL$_1$=16,TL$_2$=8,L$_{21}$=0) or (TL$_1$=24,TL$_2$=16,L$_{24}$=8)	$(3,1)$,$(4,1)$	2670	1
11	TL$_1$=16,TL$_2$=8,L$_{21}$=8	$(2,1)$,$(4,1)$	5	1
12	(TL$_1$=16,TL$_2$=24,L$_{11}$=0) or (TL$_1$=TL$_2$=24,L$_{11}$=0) or (TL$_1$=24,TL$_2$=32,L$_{11}$=0)or (TL$_1$=24,TL$_2$=8,L$_{22}$=8)	$(2,1)$	22	0
13	TL$_1$=16,TL$_2$=32,L$_{11}$=0	$(2,1)\,,(4,2)$	20	4
14	(TL$_1$=24,TL$_2$=0,L$_{11}$=0) or (TL$_1$=24,TL$_2$=8,L$_{24}$=8,k $\in \Theta$) or (TL$_1$=32,TL$_2$=8,L$_{21}$=8)	$(3,1)$	197	1
15	(TL$_1$=24,TL$_2$=8,L$_{21}$=8)or (TL$_1$=24,TL$_2$=16,L$_{22}$=8)	$(3,2)\,,(4,1)$	27	0
16	TL$_1$=24,TL$_2$=8,L$_{24}$ $= 8, k \notin \Theta$	$(3,2)\,,(4,2)$	20	0
17	TL$_1$=32,TL$_2$=16,L$_{22}$=8	$(3,2)$	10	0
18	TL$_1$=32,TL$_2$=16,L$_{24}$=8	$(3,1)\,,(4,2)$	38	3
19	Otherwise	$(0,0)$	11212	8

Table 6.13: *Branching the RL-policy* $NN\theta400\mu15\nu13$ *in the outsourcing problem. We call this policy HeuNN400*

The table leads us to a heuristic policy (HeuNN400) represented by a hierarchical set of decision rules. This heuristic policy has quality 99.36% with respect to the policy learned by the RL-agent. However, a one way-Anova test shows significant difference between the performance of these two policies, see Table 6.14. For this reason we consider these two policies as non-robust policies. By non-robustness here we mean that small differences in the two policies leads to significant difference in their performances.

Source	SS	df	MS	F	P
Column	1.061	1	1.061	16.640	0.0035
Error	0.510	8	0.064		
Total	1.571	9			

Table 6.14: *Results of a one-way ANOVA test for the RL-agent and a heuristic elaborated on the RL-learned policy*

Still both policies achieve better performance than any of the previous heuristics. Table 6.15 shows the average results on 5 samples of the HeuRL and the RL-agent.

scenario	AR_{20000}
$NN\theta400\mu15\nu13$	70.99
HeuRL400	70.34

Table 6.15: *Average results on 5 samples for the RL-agent and a heuristic elaborated on the RL-learned policy*

Furthermore, the heuristic HeuRL400 offers some general ideas about the learned RL-policy $NN\theta400\mu15\nu13$.

1. Acceptance of orders of type 1 is the most preferred preference, and if there is enough capacity, it is preferred not to outsource it.

2. There is in general opportunity to choose orders of type 1 only when $L_1 \leq 16$.

3. In case $L_1 \leq 16$ and it is not possible to choose orders of type 1, there is a certain strategy to occupy capacity (in order not to lose it) in a way that does not limit a possible acceptance of orders of type 1 in the near future. This is achieved by a specific selection of orders of type 2 and 3 only when $L_{11} = 0$, and accepting orders of type 3 and 4 with outsourcing when $L_{21} = 0$.

4. The outsourcing option (option 1) is preferred in cases where there is the possibility of losing capacity in one resource and the other one has a very high utilization; or when the capacity is reserved for a more profitable order as orders of type 1.

This RL-agent is also characterized for the acceptance of orders even in cases with high utilization over the planning horizon. Note for example the cases in the table where orders of type 4 may be accepted. This high utilization could explain the non-robustness of the heuristic HeuRL400.

Learning another RL-policy Despite of the good performance of the RL-agent $NN\theta400\mu15\nu13$, we decided to further train other RL-agents on this case. The performance did not improve too much but we obtained a more robust heuristic policy. Specifically we analyze here the policy learned by the RL-agent $NN\theta700\mu21\nu18$. This agent obtains an Average Reward of 71.47 after training, outperforming all the previous heuristics, $OrderQuality(6)$ by 32.8% and Greedy by 122.9%.

Using the simple data mining technique that we explained in Section 4.4.1, we obtain the general preference relation for the zero level of the decision tree $\phi_0 = ((1,2)\,(1,1))$. The matrix of selected/possible actions A_{ϕ_o} is shown in Table 6.16 for a total of 19881 non-trivial iterations.

Selected actions	**Possible Actions**								
	(1,2)	(1,1)	(0,0)	(3,1)	(2,2)	(2,1)	(4,1)	(4,2)	(3,2)
$(1,2)$	3025	3025	3025	2396	2345	2631	2963	2963	2396
$(1,1)$	0	111	111	0	0	102	108	108	0
$(0,0)$	0	0	11393	3658	3150	3859	10992	11076	3658
$(3,1)$	6	6	3285	3285	647	647	3244	3244	3285
$(2,2)$	0	0	1362	645	1362	1362	1316	1316	658
$(2,1)$	0	0	247	93	115	247	245	245	93
$(4,1)$	49	49	278	78	73	73	278	278	78
$(4,2)$	0	0	180	1	0	179	166	180	1
$(3,2)$	0	0	0	0	0	0	0	0	0

Table 6.16: *Selected action against possible actions in 19758 iterations of the RL-agent* $NN\theta700\mu21\nu18$ *in the outsourcing problem*

The policy obtained from the preference relation $\phi_0 = ((1,2)\,(1,1))$ makes 5352 errors, which makes 73.1% of quality, with respect to the policy learned by the RL-agent.

Table 6.17 summarizes a heuristic policy that better approximates the RL-policy. The heuristic mainly considers branching according to the total amount of loaded capacity TL_1 and TL_2 at each resource respectively, but it also considers distribution of capacity according to the loading plan L and amount of orders in the order list k in some cases. We use the set definitions:

$$\Delta = \{k\,|k_2 = 1, k_4 \in [1,2]\,\}$$
$$\Lambda = \{k\,|k_3 \in [1,2]\,, k_3 + k_4 \geq 3\,\}\,.$$

to describe some cases with a particular order list. The columns in the table specify the number of the node, the description of the node, the preference relation, the number of transitions covered by the node, and the number of errors. The table leads us to a heuristic policy (HeuNN700) represented by a hierarchical set of decision rules which have 96.88% of quality, with respect to the policy learned by the RL-agent.

N_0	States	preference order	cases	errors
1	TL_1=TL_2=0	$(4,1),(1,2),(3,1)$	164	83
2	All states	$(1,2),(1,1)$	3099	0
3	TL_1=0,TL_2=8	$(3,1),(2,2),$	1352	494
4	TL_1=0,TL_2=16	$(2,1),(3,1)$	160	36
5	TL_1=0,TL_2=24	$(2,2),(2,1)$	693	0
6	TL_1=8,TL_2=0 or TL_1=16,TL_2=8,L_{23}=8 or or TL_1=24,TL_2=8, $k_4 \neq$5	$(3,1),(4,1)$	224	2
7	TL_1=TL_2=8,L_{11}=0 or TL_1=8,TL_2=16,L_{12}=8	$(2,2)$	114	5
8	TL_1=8,TL_2=16,L_{13}=8	$(2,2),(3,1),(4,1)$	27	0
9	TL_1=8,TL_2=24,L_{11}=0	$(2,1)$	24	0
10	TL_1=8,TL_2=32,$L_{11} = 8, k \in \Delta$	$(4,2)$	179	0
11	TL_1=16=TL_2,L_{11}=0,L_{22}=8 or TL_1=24,TL_2=16,$L_{24} = 8, k \in \Lambda$	$(3,1)$	2108	0
12	TL_1=16=TL_2,L_{23}=8,$k_4 \neq$5 or , TL_1=24TL_2=8,k_4=5 or TL_1=24,TL_2=16,L_{23}=8	$(4,1)$	169	2
13	Otherwise	(0.0)	11393	0

Table 6.17: *Branching the RL-policy $NN\theta700\mu21\nu18$ in the outsourcing problem. We call this policy HeuNN700*

The average results on 5 samples of HeuNN700 and the RL-agent $NN\theta700\mu21\nu18$ are shown in Table 6.18.

scenario	AR_{20000}
$NN\theta700\mu21\nu18$	71.382
HeuRL	71.255

Table 6.18: *Average results on 5 samples for the RL-agent and a heuristic elaborated on the RL-learned policy*

A one way-Anova test did not show significant difference between the performances of these policies. See Table 6.19.

We can infer the following analogies and differences with respect to the

Source	SS	df	MS	F	P
Column	0.0408	1	0.0408	1.227	0.300
Error	0.266	8	0.033		
Total	0.307	9			

Table 6.19: *Results of a one-way ANOVA test for the the RL-agent and a heuristic elaborated on the RL-learned policy*

policy learned by the previous RL-agent $NN\theta400\mu15\nu13$.

1. As before, action (1,2) is preferred over (1,1). Both actions are preferred over all the others in most of the cases. From Table 6.16 there are 55 cases in which other actions are chosen ((4,1),(3,1)) instead of choosing orders of type 1. This may look like an error on the learned policy, but a more detailed analysis reveals that these 55 cases occur when all the capacity is free ($L_1 = L_2 = 0$), and right after choosing one of the actions ((4,1),(3,1)) an order of type 1 was chosen.

2. As before, action (2,2) is preferred over (2,1) but there is a decrement in the number of cases where (2,2) is chosen and an increment in the cases where (2,1) is chosen.

3. There is an increment in the number of cases where (3,1) is chosen, and (3,2) is never chosen.

4. As before, action (4,1) is preferred over (4,2) but there is a noticeable decrement on the selection of (4,1) and though there is also a slight increase in choosing (4,2), the final outcome is a decrement in the selection of orders of type 4.

The general strategy of this RL-agent is very similar to the one used by the RL-agent $NN\theta400\mu15\nu13$, which is to prioritize orders of type 1 and obtain a high utilization of the capacities. However some differences in strategy allow this RL-agent to increase the possibilities of choosing orders of type 1. Some of the noticed differences are the following:

- increase in the cases of outsourcing,
- reduce the cases of complete acceptance of orders of type 2 and 3,
- reduce the cases of choosing orders of type 4.

A one way-Anova test (Table 6.20) shows significant difference between the performances of the two learned RL-policies ($NN\theta400\mu15\nu13$ and $NN\theta700\mu21\nu18$).

However, both RL-policies are better than the best general heuristic we could define for this problem ($OrderQuality(6)$). The RL-agent $NN\theta700\mu21\nu18$

Source	SS	df	MS	F	P
Column	0.383	1	0.383	8.684	0.018
Error	0.352	8	0.044		
Total	0.735	9			

Table 6.20: *Results of a one-way ANOVA test for the two RL-agents* $NN\theta400\mu15\nu13$ *and* $NN\theta700\mu21\nu18$

performs slightly better than the RL-agent $NN\theta400\mu15\nu13$ and it leads also to a more robust policy.

6.4.4 Due-time and price negotiation

This is a case with a two-resources shop where 4 types of orders arrive according to independent Poisson arrivals. Table 6.21 shows the data of this case. The planning horizon is 6 days ($H = 6$) with 8 working hours a day ($C_{max} = 8$). The order type i is characterized by the arrival rate (λ_i), a routing (σ_i), and a set of due-time and reward (ϕ_i) options. Recall that the triplet $\phi_{il} = (t_{il}, p_{il}, r_{il})$ is the option l for the order type i, and t_{il}, p_{il}, r_{il} are the due-time, the probability that the customer will agree with the option, and the immediate reward upon acceptance and customer agreement. Here we consider two options ($l = 1, 2$) .

The processing times β_{ij} and the due-times t_{il} are given in hours. Decisions should be taken daily, on which orders to accept from the pool of orders that have arrived in the last 24 hours. Orders that are not accepted in that day will go out of the system.

The order of type 1 is the least frequent but the more profitable one; particularly with the due-time option 1. For all types of orders it holds that the first due-time option is more profitable than the second one. Still there may be cases where choosing the second option could be a better alternative in the long run. The order type 4 is the least profitable one, but since it has the smallest processing time, and it is the most frequent, it could be convenient to accept it sometimes. The customers will always agree with the selection of the first due-time option (that may be the one they come with) and there is a high probability they will accept the second due-time option.

$order_i$	λ_i	σ_i	ϕ_i	n_i	*Capacity Profile*
1	1	$[(1,24),(2,8)]$	$[(96,192,1),(144,112,0.9)]$	1	
2	2	$[(1,\ 8),(2,8)]$	$[(72,\ 48,1),(144,\ 38,0.8)]$	5	$H = 6$
3	2.3	$[(2,16),(1,8)]$	$[(120,70,1),(144,\ 68,0.85)]$	2	$C_{max} = 8$
4	4	$[(2,\ 8),(1,8)]$	$[(96,\ 32,1),(144,\ 24,0.95)]$	5	

Table 6.21: *Description of the case in the due-time problem*

Note that in this problem the components of the capacity profile would take only the values 0 and 8 because of the values of the processing times and C_{max}. Therefore the cardinality of the observation space is $|S| = 6^3 * 2^{12} = 884\,736$.

Heuristics

Table 6.22 shows results of the heuristic policies for this case. We consider for simplicity *CapacityLevel*(ρ) policies with the same safety level for both resources and all orders, and also *CapacityLevel*($1, \rho$), policies that consider level of safety 1 for order type 1 and ρ for the rest of the orders. The Greedy policy uses safety level 1, the orders are accepted as long as capacity is available. The *OrderQuality*(6) policy only chooses order type 1 with the first due-time option which is the more profitable action. The policy that allows to choose both options for orders of type 1, is *OrderQuality*(3.5).

scenario	AR_{20000}	scenario	AR_{20000}
Greedy	33.62	$CapacityLevel(1, 0.9)$	37.83
$CapacityLevel(0.9)$	37.83	$CapacityLevel(1, 0.7)$	41.39
$CapacityLevel(0.7)$	41.58	$CapacityLevel(1, 0.5)$	41.34
$CapacityLevel(0.5)$	44.49	$CapacityLevel(1, 0.3)$	41.08
$CapacityLevel(0.4)$	41.53	$CapacityLevel(1, 0.1)$	40.88
$CapacityLevel(0.3)$	22.86	$OrderQuality(6)$	**53.83**
Random	33.21	$OrderQuality(3.5)$	40.88

Table 6.22: *Heuristics results for the due-time problem*

The best of these heuristic policies is *OrderQuality*(6).

RL-agent

Table 6.23 shows results of some RL-agents trained for this case. RL-agents are trained sequentially according to the (θ, μ, ν)-methodology described in Section 5.3.1. The table only shows the results of the best RL-agents for some values of θ. The $NN\theta700\mu21\nu18$ RL-agent outperforms all the previous heuristics, *OrderQuality*(6) in 10.2% and the Greedy in 76.4%.

scenario	AR_{20000}
$NN\theta500\mu13\nu13$	36.25
$NN\theta600\mu19\nu18$	37.18
$NN\theta700\mu21\nu18$	**59.31**

Table 6.23: *Results of RL-agents in the due-time problem*

Learning the RL-policy

Here we analyze the policy learned by the RL-agent $NN\theta700\mu21\nu18$ using 19501 non-trivial iterations (where rejection was not a trivial decision). Using the simple data mining technique we explained in Section 4.4.1 we obtain the general preference relation for the zero level of the decision tree $\phi_0 = ((1,1))$. The matrix of selected/possible actions A_{ϕ_0} is shown in Table 6.24.

Selected Actions	**Possible Actions**								
	(1,1)	(0,0)	(1,2)	(2,1)	(3,1)	(2,2)	(3,2)	(4,1)	(4,2)
(1,1)	2297	2297	2297	1998	2035	1998	2063	2260	2260
(0,0)	17	11773	3042	3902	3423	9274	4571	9044	11546
(1,2)	2	9	9	4	7	8	2	7	7
(2,1)	0	1205	0	1205	997	1205	1009	1187	1187
(3,1)	0	2321	1	356	2321	1660	2284	2295	2308
(2,2)	0	1019	0	34	784	1019	782	864	971
(3,2)	0	333	2	50	147	76	333	145	328
(4,1)	0	346	11	16	18	271	18	346	342
(4,2)	0	198	7	8	98	86	93	115	198

Table 6.24: *Selected action against possible actions in 19501 iterations of the $NN\theta700\mu21\nu18$ RL-agent in the due-time problem*

The policy obtained from the preference relation $\phi_0 = ((1,1))$ makes 5450 errors, which makes 72.1% of quality, with respect to the policy learned by the RL-agent. So ϕ_0 defines a policy that only chooses orders of type 1, with the due-time option 1 (the earliest), and it is not a good approximation of the policy learned by the RL-agent. Such a policy resembles the heuristic $OrderQuality(6)$.

Table 6.25 summarizes a heuristic policy that better approximates the RL-policy. The heuristic mainly considers branching according to the total amount of loaded capacity TL_1 and TL_2 at each resource respectively, but it also considers distribution of capacity according to the loading plan L and amount of orders in the order list k in some cases. We use the following definitions:

$$\begin{aligned}
Tk_{24} &= \sum_{i=2}^{4} k_i, \\
\bar{k}_{24} &= (k_2, k_3, k_4)\,, \\
\Omega &= \{(1,2,3)\,,(3,2,5)\,,(4,2,*)\}\,, \\
\Phi &= \left\{\bar{k}_{24}\,|k_4 = 4, k_3 = 2, k_2 \neq 4\right\}, \\
\Psi &= \left\{\bar{k}_{24}\,|(k_4 \geq 3, k_3 = 2) \vee (k_3 = 1, k_4 = 5)\right\}
\end{aligned}$$

to describe some cases with a particular order-list. The symbol $*$ in the above definition means that there could be any value in the position of the symbol. The columns in the table specify the number of the node, the description of the node, the preference relation, the number of transitions covered by the node, and the number of errors.

N_0	*States*	preference relation	# of cases	errors
1	All states	(1,1)	2316	19
2	$TL_1=TL_2=0$	(3,1),(2,1)	846	327
3	$TL_1=0,TL_2=8$ $TL_1=8,L_{11}=0,(L_{12}=8 \vee Tk_{24} \leq 4)$ $TL_1=TL_2=16,L_{11}=0,(L_{12}=8 \vee Tk_{24} \leq 4)$ $TL_1=32,TL_2=24,L_{21}=L_{22}=L_{25}=8,\bar{k}\notin \Omega$	(2,1),(3,1)	665	12
4	$TL_1=8,L_{11}=0,(L_{12}=8 \vee Tk_{24}>4)$ $TL_1=TL_2=16,L_{11}=0,(L_{12}=8 \vee Tk_{24}>4)$	(2,1),(3,2)	494	0
5	$TL_1=16,TL_2=24,L_{11}=0$ $TL_1=32,TL_2=24,L_{21}=L_{22}=L_{25}=8,\bar{k}\in \Omega$	(2,1),(2,2)	58	0
6	$TL_1=24,TL_2=8,L_{23}=8$; $TL_1=32,TL_2=16,L_{21}=L_{22}=8$ $TL_1=40,TL_2=32,L_{24}=L_{26}=0$	(2,2),(3,1)	271	23
7	$TL_1=24,TL_2=8,L_{24}=8$	(3,1),(4,1)	1463	21
8	$TL_1=24,TL_2=8,L_{21}=8$; $TL_1=40,TL_2=24,L_{21}=L_{23}=L_{25}=8,\bar{k}\in \Phi$	(3,1)	38	0
9	$TL_1=24,TL_2=16$, $L_{23}=L_{22}=0$ $L_{23}=L_{24}=0$ $TL_1=32,TL_2=24,L_{21}=L_{24}=L_{25}=8$	(2,2),(3,1),(4,1)	1167	133
10	$TL_1=24,TL_2=24,L_{22}=0$; $TL_1=40,TL_2=24,L_{21}=L_{23}=L_{25}=8, \bar{k}\notin \Phi$	(2,2)	55	0
11	$TL_1=24,TL_2=24,L_{23}=0$	(4,1)(2,2)	155	18
12	$TL_1=32,TL_2=16,L_{21}=0, Tk_{24}\in \Psi$; $TL_1=40,TL_2=32,L_{22}=L_{25}=0$	(4,2)	86	0
13	$TL_1=32,L_2=16,L_{21}=0, Tk_{24}\notin \Psi$	(3,2),(2,2)	19	0
14	$TL_1=32,TL_2=24,L_{21}=L_{23}=L_{24}=8$	(2,2),(4,2)	45	0
15	$TL_1=40,TL_2=24$, $L_{21}=L_{22}=L_{25}+L_{26}=8$ $L_{21}=L_{24}=L_{26}=8$ $TL_1=40,TL_2=32$, $L_{23}=L_{24}=0$ $L_{22}=L_{23}=0,k_4=5$ $L_{24}=L_{25}=0$	(3,2)	61	0
16	$TL_1=40,TL_2=32,L_{22}=L_{23}=0,k_4\neq 5$	(3,2),(4,2)	154	0
17	Otherwise	Reject	11773	69

Table 6.25: *Branching the RL-policy* $NN\theta700\mu21\nu18$ *in the due-time problem. We call this policy HeuNN700*

The table leads us to a heuristic policy (HeuNN700) represented by a hierarchical set of decision rules which have 96.36% of quality, with respect to the policy learned by the RL-agent. A complete branching according to the possible values of the arrival list is very expensive computationally, since the number of combinations could be up to $216*2^{12} = 884\,736$ (216 types of arrivals and 6 stages per each of the 2 resources, each taking values 0 and 8).

The average results on 5 samples of the HeuRL and the RL-agent are shown in Table 6.26.

scenario	AR_{20000}
$NN\theta700\mu21\nu18$	59.44671239
HeuRL	59.58850616

Table 6.26: *Average results on 5 samples of the RL-agent* $NN\theta700\mu21\nu18$ *and a heuristic elaborated on the RL-learned policy in the due-time problem*

A one way-Anova test did not show significant difference between the performances of these policies. See Table 6.27.

Source	SS	df	MS	F	p
Column	0.050263686	1	0.050263686	2.261824902	0.171007814
Error	0.177780997	8	0.022222625		
Total	0.228044683	9			

Table 6.27: *Results of a one-way ANOVA test for the* $NN\theta700\mu21\nu18$ *RL-agent and a heuristic elaborated on the RL-learned policy in the due-time problem*

From the analysis on the policy learned by the RL-agent, we can draw some conclusions:

1. There is a prioritization of choosing order type 1 with the earliest due-time option (option 1). This is the action with highest reward per capacity request. However the latest due-time does not seem to be a good choice for this order type, even though it is the second more profitable action.

2. There is opportunity to choose order type 1 with option 1 only when $C_1 \leq 16 \wedge L_2 \leq 24$.

3. In case $L_1 \leq 16 \wedge L_2 \leq 24$ and it is not possible to choose order type 1 with option 1, there is a certain strategy to occupy capacity (in order not to lose it) in a way not to limit a possible acceptance of order type 1 in the near future. This is achieved with a specific selection of orders type 2 and 3, only when $L_{11} = 0$.

4. In case $L_1 > 16 \vee L_2 > 24$, the taken actions guarantee the use of the free capacity that could be lost in resource 2.

5. The second option of due-time is preferred when there is a high utilization of capacity such that there is more flexibility to reschedule the accepted orders.

The learned policy seems to allocate capacity in a more look-ahead-like way, being concerned not only with the immediate next step but also taking into account more steps ahead, and the frequency of arrivals of each order type. We believe that this is the explanation of the way orders of type 2, 3, and 4 are chosen. It is not possible to determine this policy using only the information of the capacity profile and the order list. Thus we can conclude that the RL-agent was able to extract implicit information of the problem in order to obtain a reasonably good policy for the Order Acceptance and due-time decisions.

6.5 Conclusions

In this chapter, we have shown that RL can be a good option to find efficient decision policies in complex situations, where the acceptance decision is combined with other decisions like routing, outsourcing or due-time negotiation. In all cases RL-agents improve simple heuristics.

In the OA+routing case the RL-agent outperforms the Greedy with 131.3% and with 14.9% the best general heuristic agent that we could define a priori with complete information. With the simple data mining technique explained in Section 4.4.1, we could define heuristic rules with 99.7% quality.

In the OA+outsourcing case the RL-agent outperforms the Greedy with 122.95% and with 32.77% the best general heuristic agent that we could define a priori with complete information. With the simple data mining technique we could define heuristic rules with 96.88% quality.

In the OA+due-time case the RL-agent outperforms the Greedy with 76.4% and with 10.2% the best general heuristic agent that we could define a priori with complete information. With the simple data mining technique we could define heuristic rules with 96.4% quality.

The interpretation of the learned policies shows that better heuristics for these cases may be policies that prioritize other orders than the order with higher reward per unit of capacity request. The learned policies seem to allocate capacity in a more look-ahead-like way, being concerned not only with the immediate next step but also taking into account some more steps ahead, and the frequency of arrivals of each order type. It is not possible to define this kind of policies using only the information on the capacity profile and the order list. Thus we can conclude that RL-agents are able to extract the implicit information of the problems in order to obtain reasonably good policies for the Order Acceptance and other decisions.

Chapter 7

Conclusions and further research

This chapter summarizes the results from the thesis taking into account the research questions introduced in Chapter 1. Furthermore we indicate directions for further research.

The general motivation of the research laid down in this thesis is to exploit to what extent intelligent computational techniques can be used to solve optimization problems with implicit information. Since OA is an essential business problem that has not been extensively studied in this respect, we decided to focus on OA under uncertainty taking into account opportunity costs. This leads us to the first research question:

1. How can we model OA under uncertainty taking into account opportunity costs?

The use of RL methods is motivated by the fact that Reinforcement Learning is a promising approach that combines the ideas of modeling uncertainty and solving the problem of incomplete information. The idea of learning without the necessity of complete model information and the possibility of learning even from delayed rewards allows us to consider different degrees of uncertainty and to take into account the opportunity cost problem in a natural way. Since RL is still in his childhood, successful applications are not always fully understood at a theoretical level. Especially the problem of tuning the training parameters for the RL-agents has not been addressed in the literature in a general theoretical setting. The latter is both a challenge and a threat. Therefore it can be considered as one of the true concerns of this thesis. This leads us to our second and third research questions:

2. How to tune the parameters for RL?
3. How does RL perform compared to other heuristics for OA?

Another concern is the question how to open the black box formed by the agent's brain, i.e., how to interpret the results found during the learning process and how to translate them into useful heuristic rules for the original OA problem.

4. How can we interpret the knowledge learned by using the RL approach in OA under uncertainty?

In Section 7.1 we give a summary of the contributions of this thesis and outline the answers to our research questions. We conclude this thesis in Section 7.2 with suggestions for further research.

7.1 Contributions

The main contributions of this thesis are:

- new semi-Markov decision models for OA
- a methodology for tuning the RL training parameters
- a general class of heuristics for OA
- a general framework for generating new heuristic rules for OA problems under uncertainty.

All these contributions seen as a whole, define a general *RL-metaheuristic* for the application of RL to OA. We believe this is a general approach that may also be applied to other logistic problems. In conclusion, we may say that RL is a good option to obtain advanced heuristic policies especially in problems with a large state space, and when the decision making has to be done under uncertainty.

Next we describe the main contributions in more detail and outline the answers to our research questions.

7.1.1 Modeling Order Acceptance under uncertainty

The *first research question* is about modeling OA under uncertainty taking into account opportunity costs. In this thesis, we present six new semi-Markov decision models for OA problems under uncertainty with different levels of complexity (Chapters 3-6). We consider in all models a finite number of types of orders. Types of orders are characterized by their immediate reward, due-time (except for Model 1), arrival rate and processing time. Capacity is modelled in different ways: as a single server that can handle only one job at a time, as a single resource consisting of multiple identical servers, and as multiple different resources. In the first three models the goal of the decision maker is to maximize the expected value of the total discounted reward by accepting orders without violating possible due-times. In the last three models besides the OA decision other decisions are considered: routing, outsourcing and due-time negotiation.

From an SMDP point of view, uncertainty in one or more of the considered OA problems is given by the following stochastic processes: arrivals of orders, processing times, capacity perturbation and negotiation of due-time. Optimizing long term criteria in each semi-Markov decision problem allows us to consider the opportunity costs problem in a natural way. Although not fully exploited in this thesis, the RL approach to solve these models allows us to consider incomplete information as a different source of uncertainty, namely not knowing the parameters of the stochastic processes mentioned above. The RL approach also allows us to consider simulation models instead of explicit models. This is very useful for complex problems where it may be difficult to obtain the transition probabilities.

This modeling approach is very flexible and may be easily extended to other OA situations. The RL-OA approach does take into account opportunity costs and uncertainty. Hence we have addressed our first research question about modeling OA under uncertainty taking into account opportunity costs.

For the simulation models we use two different types of software.

1. Matlab (11.1, MathWorks 2000) for the possibility of using the ANN toolbox. We use it for the first two models which are quite simple.
2. eM-Plant (Tecnomatix, 2002). This is a more sophisticated simulation software which allows us to incorporate our OA models into an integrated planning approach previously developed by other researchers (Ebben et al., 2005).

7.1.2 Tuning the training parameters

In order to address our *second research question* we present in Chapter 5 a methodology for the automatic tuning of the training parameters for the RL-agents we use in this thesis. Tuning parameters when stochasticity is involved is generally more an art than a science. In this thesis we use a specific structure for the RL-agents which leads to a learning schedule with six parameters $(T, \alpha_0, \epsilon_0, T_\alpha, T_\epsilon, \theta)$, see Section 2.3.2. This schedule defines the number of iterations of the training process (T), the initial value of the learning and exploration rate (α_0, ϵ_0), the parameters for the learning and exploration rates decreasing functions (T_α, T_ϵ), and the number of hidden neurons (θ). From the numerical experiments with the different problems discussed in Chapters 3 and 4, we observed some regularities that help us to set guidelines for the tuning of these parameters. In Chapter 5 we present a methodology that reduces the set of six parameters to a *(θ, μ, ν)-learning schedule* with three parameters using $\alpha_0 = 10^{-3}$, $\epsilon_0 = 1$, $T = \mu 10^4$, $T_\alpha = \mu 10^3$, $T_\epsilon = \nu 10^3$ and $\nu \in [1, ..., \mu]$. We use this learning schedule in Chapters 5 and 6 aiming to improve the best known heuristic. In the deterministic case in Chapter 5 we could only approximate the best known heuristic. We believe the heuristic at hand in this case gives already a good policy. Obtaining an RL-agent that improves such a heuristic would have been computationally very expensive. Still the RL-agent trained using the methodology for tuning parameters, allowed us to obtain an advanced heuristic which outperformed the best known heuristic so far. In the three cases in Chapter 6 we obtain RL-agents using the methodology for tuning parameters that outperform the best known heuristic.

7.1.3 The Reinforcement Learning approach outperforms simple heuristics for OA

To answer the *third research question* we need to consider heuristic methods to be compared with our RL approach. In Chapter 4 we present a general class of heuristics for OA. This class of heuristics may be viewed as a characterization of the SMDP policies by means of a set of linear orderings on the action space combined with a prescription of the allowed actions for each state. Using this characterization we describe a one-parameter family of heuristics (*OrderQuality* (b)) and a multi-parameter family of heuristics (*CapacityLevel* (ρ)). We use these heuristics to compare the quality of the policies learned by the RL-agents.

According to the *OrderQuality*(b) heuristic only orders with a reward per processing time above a threshold b may be accepted. This threshold is related to the avoidance of opportunity losses. This family of heuristics includes the Directed Costing rules and Absorption Costing rules from the literature.

CapacityLevel(ρ) heuristics consider a capacity level threshold (ρ_i) for each type of order i. This family of heuristics includes the policies that reserve capacity for the most valuable orders. Note that, in case of penalization for excess capacity, it might be better to fill capacity only up to a certain level of utilization. So there is a safety margin for dealing with perturbations due to non-anticipated extra capacity demand during job execution.

In this thesis we present the analysis of 13 cases. In the first three cases, where the state space is quite small, the results of the RL-agents are compared with optimal policies. In all the three cases RL-agents outperform the greedy policies and approximate the results of the optimal policies. For the other cases we use parametric subclasses of the general class of heuristics to compare with. In one case a trained RL-agent approximated the best known heuristic (0.66% of relative error in the average reward). In nine cases, trained RL-agents outperformed the best known heuristic. In all these ten cases, the final results of the *RL-metaheuristic* outperform the results of the parametric heuristics.

Hence we have answered our third research question. The RL-approach is able to extract implicit information from OA problems under uncertainty in order to find good decision policies that outperform simple heuristics rules.

7.1.4 Generating new heuristic rules for OA under uncertainty.

An essential contribution of Chapter 4 is the development of a framework for interpreting the learned RL policies and to elaborate new advanced heuristic policies. Though developed in a specific context of the OA problem at hand, the framework is very general. In fact, it applies to any decision problem under uncertainty. The framework is built up out of three steps. In the first step we train the RL-agent and after having done so perform a large series of iterations with it. In the second step we employ a simple data mining technique to extract explicit knowledge about action preferences from the performed series of iterations. In the third step we identify heuristic rules from this action preference knowledge. The method laid down in our framework is powerful and universal. We use it throughout the Chapters 4, 5 and 6.

Using this framework we were able to find advanced heuristics for OA under uncertainty and interpret the learned RL-policies. Herewith we have answered our *fourth research question.*

7.2 Further research

7.2.1 Problem dimensions

We have modelled six OA problems. We have gone from the simplest model with a single resource and simple arrivals to integrated planning models with multiresources capacity and batch arrivals. However, other OA problems may be studied using the approach presented in this thesis.

As an interesting possibility we mention the case that rejected orders are not immediately lost, but can be put in "inventory" for a certain amount of time. In this way they may be accepted at a later moment in time than their arrival. This might happen if at a future decision moment new arrivals of more interesting orders than those in inventory do not occur. Actually this is a simple variant of the OA problems that we studied in the thesis. Orders in inventory can simply be considered as "arrivals from inventory".

All the cases in the job-shop environment only considered two different resources. These cases were complex enough to illustrate the applicability of our approach, however in future -with faster implementations- larger and more realistic cases could be considered.

As a last generalization we mention, that the rejection of an order may have repercussions for the future customer relations, which would required a different modelling approach.

7.2.2 Solution dimensions

In this thesis we have applied some classical RL algorithms. However, RL is an area of much recent research which offers important theoretical and practical challenges. At the conclusion of this thesis, the number of applications and theoretical studies of RL has grown. There are several directions that will be worthwhile to take into account for extending the research line set out in this thesis. In general we propose:

1. Different knowledge representation and training when using function approximation.
2. Incorporating prior information into the learning mechanism.
3. Parallel implementation of the parameter tuning process.
4. Improving data mining algorithms to interpret the learned RL policies.
5. Multi-agent systems for the loading procedures to support OA.
6. Average reward instead of expected discounted reward algorithms.

Leaving the field of order acceptance it would be worthwile to investigate to what extent the RL-metaheuristic developed in this study may be helpful to generate advanced heuristic rules for other intricate logistic problems with their own specific uncertainties.

Bibliography

Andrews, R., Diederich, J., & Tickle, A. (1995). Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8(6), 373–389.

Bakker, B. (2002). Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems (NIPS'13)*, volume 13. On J. Schmidhuber's CSEM grant 2002.

Bakker, B., Linaker, F., & Schmidhuber, J. (2002). Reinforcement learning in partially observable mobile robot domains using unsupervised event extraction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)* Switzerland and Sweden. Lausanne.

Bakker, B., Zhumatiy, V., & G. Gruener, a. J. S. (2003). A robot that reinforcement-learns to identify and memorize important previous observations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2003.*

Bertsekas, D. P. & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Belmont, MA: Athena Scientific, 1st edition.

Boyan, J. A. & Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. In J. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in Neural Information Processing Systems*, volume 6 (pp. 671–678).: Morgan Kaufmann, San Mateo, California.

Brouns, G. A. & van der Wal, J. (2000). *Optimal threshold policies in a workload model with a variable number of service phases per Job.* SPOR-Report 66, Technical University of Eindhoven, The Netherlands.

Carlstrom, J. (2000). *Reinforcement Learning for Admision Control and Routing.* PhD thesis, Departmente of Computer Systems. Uppsala university.

Charnsirisakskul, K., Griffin, P. M., & Keskinocak, P. (2004). Order selection and scheduling with leadtime flexibility. kasarin, pgriffin, pinar at isye.gatech.edu.

Crites, R. H. & Barto, A. G. (1996). Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), *Advances in Neural Information Processing Systems*, volume 8 (pp. 1017–1023).: Cambridge, MA. The MIT Press.

Ebben, M., Hans, E., & Olde Weghuis, F. (2005). Workload based order acceptance in job shop environments. *OR Spectrum*, 27(1), 107–122.

Garbe, R. (1996). *Algorithmic aspects of interval orders*. PhD thesis, University of Twente, The Netherlands.

Gietzmann, M. B. & Monahan, G. E. (1996). Absorption versus direct costing: the relevance of opportunity costs in the management of congested stochastic production systems. *Management Accounting Research*, 7, 409–429.

Gietzmann, M. B. & Ostaszewski, A. (1996). Optimal disbursement of a sunk resource and decentralized cost allocation. *Accounting and Business Research*, 27(1), 17–40.

Han, J. & Kamber, M. (2000). *Data Mining: Concepts and Techniques: Management Systems*. Morgan Kaufmann Series in Data Management Systems, 2ndt edition.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition.

Hofstadter, D. R. (1999). *Gödel, Escher, Bach: an Eternal Golden Braid*. Penguin Books, 20th-aniversary edition edition.

Ivanescu, V. C. (2004). *Order acceptance under uncertainty in batch process industries*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands.

Kaelbling, L., Littman, M., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 99–134.

Kaelbling, L., Littman, M., & Moore., A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 237–285.

Kearns, M., Mansour, Y., & Ng., A. (2000). Approximate planning in large pomdps via reusable trajectories. In *Advances in Neural Information Processing Systems 12*, volume 12: The MIT Press.

Kearns, M. & Singh, S. (1999). Finite-sample convergence rates for q-learning and indirect algorithms. In *Advances in Neural Information Processing Systems 11*, volume 11 (pp. 996–1002).: The MIT Press.

Mainegra-Hing, M., van Harten, A., & Schuur, P. Reinforcement learning versus heuristics for order acceptance on a shared resource. *to appear in Journal of Heuristics*.

Mainegra-Hing, M., van Harten, A., & Schuur, P. (2001). *Order acceptance with reinforcement learning.* Technical Report 66, BETA, University of Twente, The Netherlands.

Mainegra-Hing, M., van Harten, A., & Schuur, P. (2002). Reinforcement learning for order acceptance on a shared resource. In *Proceedings of the 9th International Conference on Neural Information Processing* (pp. 815–819). Orchid Country Club, Singapore.

Miller, B. L. & Buckman, A. (1987). Cost allocation and oportunity cost. *Management Science*, 33(5), 59–62.

Nawijn, W. M. (1985). The optimal look-ahead policy for admission to a single server system. *Operations Research*, 33(3), 625–643.

OldeWeghuis, F. (2002). *Integrated Order Acceptance and Job Shop Loading in a Rolling Horizon: a simulation approach.* Master thesis, University of Twente, The Netherlands.

Pinedo, M. (2002). *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall; 2 edition.

Pinedo, M. & Chao, X. (1999). *Operations Scheduling with applications in manufacturing and services.* McGraw Hill.

Puterman, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* J. Wiley and Sons.

Raaymakers, W. (1999). *Order acceptance and capacity loading in batch process industries.* PhD thesis, Technische Universiteit Eindhoven, The Netherlands.

Ratitch, B. (February 2005). *On characteristics of markov decision processes and reinforcement learning in large domains.* PhD thesis, McGill University, Montreal.

Ratitch, B. & Precup, D. (2003). Using mdp characteristics to guide exploration in reinforcement learning. In D. Thomas (Ed.), *Proceedings of the European Conference on Machine Learning* (pp. 313–324).

Riedmiller, S. C. & Riedmiller, M. A. (1999). A neural reinforcement learning approach to learn local dispatching policies in production scheduling. In D. Thomas (Ed.), *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, volume 12 (pp. 764–771). S.F.: Morgan Kaufmann Publishers.

Samuel, A. (1959). Some studies in machine learning using the game of chekers. *IBM Journal on Research and Development*, 3, 211–229. Reprinted in E.A. Feigenbaum and J. Feldman (eds.) Computers and Thought, pp.71-105,1963.

Samuel, A. (1967). Some studies in machine learning using the game of chekers. ii-recent progress. *IBM Journal on Research and Development*, 11, 601–617.

Singh, S. & Bertsekas, D. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In M. C. Mozer, M. I. Jordan, & T. Petsche (Eds.), *Advances in Neural Information Processing Systems*, volume 9 (pp. 974).: The MIT Press.

Snoek, M. (2000). Neuro-genetic order acceptance in a job shop setting. In *Proceedings of the 7th International Conference on Neural Information Processing* (pp. 815–819). Taejon, Korea.

Stidham, S. & Weber, R. (1993). A survey of markov decision models for control of networks of queues. *Queueing Systems*, 13, 291–314.

Sutton, R. & Barto, A. (1998). *Reinforcement Learning; An Introduction.* MIT Press, London, England.

Tadepalli, P. & Ok, D. (1998). Model-based average reward reinforcement learning. *Artificial Intelligence*, 100(1-2), 177–223.

ten Kate, H. (1995). *Order acceptance and production control.* PhD thesis, Rijksuniversity Groningen, The Netherlands.

Tesauro, G. J. (1994). Td–gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2), 215–219.

Thrun, S. (1992). The role of exploration in learning control. In D. A. S. David A. White (Ed.), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches.* Van Nostrand Reinhold, New York.

Tickle, A., Andrews, R., Golea, M., & Diederich, J. (1998). The truth will come to light: directions and challenges in extracting rules from trained artificial neural networks. *IEEE Trans Neural Networks*, 9(6), 1057–1068.

Towell, G. (2000). *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction.* PhD thesis, Computer Science Department, University of Wisconsin.

Towell, G. G. & Shavlik, J. W. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13, 71–101.

Van Roy, B. (1998). *Learning and value function approximation in complex decision processes.* PhD thesis, Massachusetts Institute of Technology.

Wang, J. (1994). Multicriteria order acceptance decision support in overdemanded job shops: A neural network approach. *Mathematical and computer modelling*, 33.

Watkins, C. (1989). *Learning from delayed rewards.* PhD thesis, Cambridge University.

Watkins, C. J. & Dayan, P. (1992). Q-learning. *Machine Learning*, 8, 279:292.

Wester, F., Wijngaard, J., & Zijm, W. (1992). Order acceptance strategies in a production to order environment with setup times and due dates. *International journal of production research*, 30, 1313–1326.

Wijngaard, J. & Miltenburg, G. (1997). On the cost of using capacity flexibility; a dynamic programming approach. *International Journal of Production Economics*, 53(1), 13–19.

Winston, W. L. (1994). *Operations research: Applications and algorithms.* Duxbury Press.

Wouters, M. (1997). Relevant cost information for order acceptance decisions. *Production Planning and Control*, 8(1), 2–9.

Zhang, W. & Dietterich, T. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1114–1120).

Appendix A

Glossary of symbols

A.1 Order characteristics

n	number of different type of orders
p_i	processing time of order of type i
q_i	probability of arrival of order of type i
λ_i	frequency of arrival of order of type i
t_i	due-time of order of type i
r_i	reward for acceptance of order of type i
w_i	expected processing time of order of type i
k	order-list
k_i	number of orders of type i requesting service
m_i	maximum amount of orders of type i in the order list
σ_i	routing of order of type i
R_{ij}	resource of job j in order of type i
β_{ij}	expected processing time of job j in order of type i
B_{ij}	processing time distribution for job j in order of type i
w_{ij}	processing time of job j in order of type i
JL	job list of accepted orders

A.2 Capacity profile characteristics

c	capacity profile
H	number of stages in the planning horizon of the capacity profile
$C_{\max}$	maximum capacity per stage of the capacity profile

c_t	occupied capacity at stage t of the planning horizon
L	loading profile
TL_i	Total loaded capacity in resource i
p	perturbation term
$pen(c,p)$	penalty for using non-regular capacity
η	penalty per unit of non-regular capacity

A.3 SMDP characteristics

S	state space
A	action space
$A(s)$	set of possible actions for state s
$rew(s,a,s')$	immediate reward when in state s action a is taken leading to the next state s′
$R(s,a)$	expected immediate reward when in state s action a is taken
$d(s,a,s')$	time between decision moments when in state s action a is taken leading to the next state s′
$\Pr(s,a,s')$	transition probability when in state s action a is taken leading to the next state s′
π	policy
$V(s)$	state-value function
$Q(s,a)$	action-value function
γ	discount factor

A.4 RL-agent characteristics

ANN	Artificial Neural Network
θ	number of hidden neurons in the ANN
I	inputs in the ANN
w	set of weights in the ANN
W	matrix of weights from the input to the hidden layer in the ANN
b	vector of biases in the hidden layer
W_o	vector of weights from the hidden layer to the output neuron
b_o	bias in the output neuron
w_t	vector of weights at iteration t
$Q_t(s,a,w_t)$	estimated action-value function at iteration t
λ	eligibility trace

continued on next page

continued from previous page

T	number of iterations of the learning process
α_t	learning rate at iteration t
ε_t	exploration rate at iteration t
T_α	parameter defining decreasing speed of learning rate
T_ε	parameter defining decreasing speed of exploration rate
$NN\theta A\mu B\nu C$	RL-agent trained using the (θ, μ, ν)-learning schedule for $(\theta, \mu, \nu) = (A, B, C)$

Appendix B

About experimental results

B.1 More experiments to support the learning schedule

In Section 2.3.2 we defined a general *learning schedule* with six parameters. This schedule defines the number of iterations of the training process (T), the initial value of the learning and exploration rates (α_0, ε_0), and the parameters for the learning and exploration rates decreasing functions ($T\alpha, T\epsilon$) and the number of hidden neurons (θ). Here we present some extra experimental results to support the new *learning schedule* with three parameters and the tuning procedure presented in Chapter 5.

B.1.1 Case 3 from Chapter 4

Here we show how we obtained the parameters for the experiments presented in 4.5.3. First we study the influence of the number of iterations in the training process (T) and the number of hidden neurons (θ) while keeping the other parameters constant. Taking into account that 10^4 is a small value compare to the size of the state space we start with $T = 10^4$ and increase this value. For the hidden neurons we just tried using 10, 20, 30, etc.

We consider $\alpha_0 = \varepsilon_0 = 1$ and the idea that the final value of the learning and exploration parameters should be less than 10% of their initial values. Hence $T_\alpha = T_\epsilon = 10^3$. Table B.1 shows the values of the performance measure AR_{20000}. The results were very poor.

Next the idea was to make $T_\alpha = T_\epsilon = \frac{T}{10}$ in order keep some level of explo-

	θ				
T	10	20	30	40	50
1x10^4	0	0.412	3.119	3.120	3.121
2x10^4	0	0.296	3.128	3.126	3.128
3x10^4	0	0.502	3.127	3.129	3.129

Table B.1: *Bad performance of RL-agents for* $\alpha_0 = \epsilon_0 = 1, T_\alpha = T_\epsilon = 10^3$

ration and learning when increasing the number T of iterations. Furthermore we focus on using $\theta = 50$, see Table B.2. The results were similar, still bad results.

	θ		
T	30	40	50
1×10^4	3.119	3.120	3.121
2×10^4	3.128	3.129	3.130
3×10^4	3.123	3.123	3.123

Table B.2: *Bad performance of RL-agents for* $\alpha_0 = \epsilon_0 = 1, T_\alpha = T_\epsilon = \frac{T}{10}$

Next we explore different initial values for the learning rate α_0. Furthermore we focus on using $\theta = 50$, since the results in Table B.2 does not show significant differences for the different values of θ. See Table B.3.

	$\alpha_0 = 0.1$	$\alpha_0 = 0.01$	$\alpha_0 = 0.001$
$T = 10^4, T_\alpha = T_\epsilon = 10^3$	4.753	4.875	3.379
$T = 2 \times 10^4, T_\alpha = T_\epsilon = 10^3$	4.033	4.977	3.075
$T = 3 \times 10^4, T_\alpha = T_\epsilon = 10^3$	4.234	4.816	3.050
$T = 2 \times 10^4, T_\alpha = T_\epsilon = 2 \times 10^3$	3.010	4.435	3.278
$T = 3 \times 10^4, T_\alpha = T_\epsilon = 3 \times 10^3$	2.509	5.037	4.627

Table B.3: *Improvement in performance of RL-agents for smaller values of* α_0. $\epsilon_0 = 1, \theta = 50$

Using a smaller initial learning rate diminishes the rank of the learning values during learning, so learning occurs slower. The last two columns show a significant improvement on performance when increasing the number of training iterations. The best results for these two values of α, are in the case $T =$ 3x$10^4, T_\alpha = T_\epsilon =$ 3x10^3 so we decided to explore more values for T and θ keeping the relation $T_\alpha = T_\epsilon = \frac{T}{10}$. To generate the values T and θ we use an enumerative recursive procedure with two levels, type of *push*, *pop* and *stack* (Hofstadter, 1999). A step in the higher level corresponds to a new value of θ, *to push* means to keep that value of θ while going to the lowest level where the values of T are increased in 10^4. The values of T are increased while the performance AR_{20000} is improved. If AR_{20000} does not improve with a new value of T we close operations at that level and resume operations where we left off on the highest level, this constitutes *to pop.* The value of θ is increased to 50. The *stack* is to keep record of where we were in each level, see Tables B.4 and B.5.

θ	$\frac{T}{10^4}$	AR_{20000}	θ	$\frac{T}{10^4}$	AR_{2000}
50	3	**5.037**	250	7	5.106
	4	4.953		8	5.039
100	4	4.897	300	8	4.882
	5	4.539		9	4.941
150	5	**5.115**		10	5.071
	6	5.081		11	**5.125**
200	6	5.042		12	4.956
	7	4.860			

Table B.4: *push, pop and stack for two levels of θ and T.* $\alpha_0 = 0.01, \epsilon_0 = 1, T_\alpha = T_\epsilon = \frac{T}{10}$

θ	$\frac{T}{10^4}$	AR_{20000}	θ	$\frac{T}{10^4}$	AR_{20000}
50	3	**4.627**	150	11	5.097
	4	4.404	200	11	**5.119**
100	4	**4.699**		12	**5.161**
	5	4.881		13	5.124
	6	**5.039**	250	13	5.104
	7	**5.103**		14	5.152
	8	5.039		15	5.107
150	8	5.029	300	15	5.063
	9	5.090		16	**5.174**
	10	**5.107**		17	5.323

Table B.5: *push, pop and stack for two levels of θ and T.* $\alpha_0 = 0.001, \epsilon_0 = 1, T_\alpha = T_\epsilon = \frac{T}{10}$

In the case $\alpha_0 = 0.001$ although the learning is slower the final result with $\theta = 300$ is better than in the case $\alpha_0 = 0.01$. With all the cases in Chapter 4 the results were similar.

B.1.2 Routing case from Chapter 6

Here we show how we obtained the parameters for the experiments presented in 6.4.2. In the previous experiments using $T_\alpha = T_\epsilon$ was sufficient to achieve good results. Some more experiments show that decreasing the initial value of exploration rate does not help to improve performance but instead one must use a faster decrease of the exploration, e.g., use $T_\alpha = \frac{T}{10}$ and $10^3 \leq T_\epsilon \leq \frac{T}{10}$.Table

B.6 shows a *push*, *pop* and *stack* procedure with three levels for the parameters θ, T, and T_ϵ.

θ	$\frac{T}{10^4}$	$\frac{T_\epsilon}{10^3}$	AR_{20000}
300	11	11	32.020
	12	11	32,020
		12	**34.027**
	13	12	32.038
		13	32.056
350	13	12	33,953
		13	32.023
	14	12	32.175
		13	32.531
		14	32.281

θ	$\frac{T}{10^4}$	$\frac{T_\epsilon}{10^3}$	AR_{20000}
400	14	12	**73.939**
		13	32.029
		14	32.038
	15	12	63.532
		13	32,057
		14	32.220
		15	32.033
450	15	12	73.324
		13	32.023
		14	**74.107**
		15	73.963

Table B.6: *push, pop and stack for three levels of θ, T and* $10^3 \leq T_\epsilon \leq \frac{T}{10}$. $\alpha_0 = 0.001, \epsilon_0 = 1, T_\alpha = \frac{T}{10}$

These results show the sensitivity of the performance to the exploration rate. All the experiments in Chapters 5 and 6 use this scheme with three parameters corresponding to $(\theta, \frac{T}{10^4}, \frac{T_\epsilon}{10^3})$

B.2 Some thoughts on scalability issues

It is an intuitive idea that the more complex the OA problem, the more difficult the learning. We have the believe that it is in the complex problems where we can really appreciate the benefits of using the RL approach. However measuring complexity of the models is not a trivial issue. As an initial attempt we consider the size of the state and action space $|S \times A|$ as an indicator of the complexity of an SMDP. Consequently, we use the size of the observation and action space for the POMDP, in the sequel this indicator appears as $|S \times A|$ as well. Then we present the relation between this indicator and the results of our experiments. Let us first introduce some notation.

NNP: number of parameters of the ANN used in the first RL-agent that outperforms the best known heuristic.

%INC.AR: Percentage of the increase of the performance in the best RL-agent found with respect to the best known heuristic.

Rank HN: The rank of the number of hidden neurons used for the first and last RL-agents that outperform the best known heuristic

Rank It.: The rank of the number of iterations for the first and last RL-agents to outperform the best known heuristic.

We want to emphasize that the best RL-agent is usually the last RL-agent we obtained for each case, this could indicate that further improvement can be achieved by continuing training other RL-agents. For the purpose of this research we only aim to outperform the best known heuristics. Only in some small cases we extend the training of other RL-agents. Table B.7 shows the results.

Case	$\lvert S \times A \rvert$	$\frac{NNP}{\lvert S\times A\rvert}$	%Inc AR	Rank HN	Rank It.
example in Chapter 1	4.09×10^{11}	1.625×10^{-8}	6.4	350-350	$6\times10^{4}-1.1\times10^{5}$
1-Chapter 4	3.54×10^{5}	3.67×10^{-3}	2.12	100-300	$3\times10^{4}-10^{5}$
2-Chapter 4	3.54×10^{5}	3.67×10^{-3}	3.67	100-250	$4\times10^{4}-1.2\times10^{5}$
3-Chapter 4	2.41×10^{6}	2.7×10^{-4}	21.07	50-300	$3\times10^{4}-1.6\times10^{5}$
4-Chapter 4	2.41×10^{6}	2.7×10^{-4}	28.35	50-300	$2\times10^{4}-1.6\times10^{5}$
1-Chapter 5	7.97×10^{7}	9.6×10^{-5}	-0.66	450	2.2×10^{5}
2-Chapter 5	8.47×10^{12}	1.11×10^{-8}	0.27	550	2×10^{5}
1-Chapter 6	3.96×10^{5}	2.28×10^{-2}	14.9	400-450	$1.2\times10^{5}-1.4\times10^{5}$
2-Chapter 6	3.98×10^{6}	1.73×10^{-3}	31.9	300-400	$1.3\times10^{5}-1.5\times10^{5}$
3-Chapter 6	7.96×10^{6}	2.97×10^{-3}	10.2	700	1.8×10^{5}

Table B.7: *Relation between the size of the OA models and the experimental results*

The results show that in general the relation $\frac{NNP}{|S\times A|}$ improves (gets smaller) when the size of the problem increase. This suggests that the number of parameters for the ANN representations does not need to grow proportional to the size of the problem in order to outperform simple heuristics. We may say that for complex problems where heuristics do not offer good solutions, the RL-approach may be a good alternative that with no much effort can easily outperform the heuristics. %INC.AR also supports this. In the case that the RL-agent can not outperform the heuristic it may be that the heuristic is already a good policy for that problem.

With respect to the results from Rank HN and Rank It it seems that starting with 50 hidden neurons and 10^4 iterations is a good idea because for the smaller cases it helps to outperform the heuristic almost since the first agent. When using larger numbers of hidden neurons the learning becomes slower, as it seems the RL-agent is in a process to polish its policy.

Summary

In this thesis, we study Order Acceptance (OA) problems under uncertainty and their solutions using Reinforcement Learning (RL). OA is an essential business problem that has not been extensively studied in this respect. Orders with different characteristics arrive stochastically at a processing facility; based on expected total profit a decision has to be made whether or not to accept an incoming order. Unaccepted orders are lost forever. RL is a promising approach that combines the ideas of modeling uncertainty and solving the problem of incomplete information (using learning methods). The idea of modeling uncertainty is based on modeling semi-Markov decision problems (SMDP). Our first concern is to model OA problems under uncertainty in such a way that RL can be applied. Although RL has successful applications in various areas, these applications are not always fully understood at a theoretical level. Especially the problem of tuning the training parameters for the RL-agents has not been addressed in the literature in a general theoretical setting. The latter is both a challenge and a threat. Therefore it can be considered as one of the true concerns of this thesis. Another concern is the question how to open the black box formed by the agent's brain, i.e., how to interpret the results found during the learning process and how to translate them into useful heuristic rules for the original OA problem.

In Chapter 1 we outline relevant OA issues. We define our research questions, review the relevant literature and present an example to illustrate our line of research. Chapter 2 summarizes the basic background for RL and specifies our approach for using RL in OA.

In Chapter 3 we start with a simple OA problem and define our first OA model. In Model 1 we consider systems with a single server without a queuing facility, and a finite number of types of orders with single stochastic arrivals and deterministic processing times. The model of this problem as an SMDP is very simple. Methods from SMDP theory can easily cope with this problem when the complete model is known, yielding us the optimality equations and the threshold structure for an optimal policy. So we can use an optimal solution to compare it with the results from the RL-algorithms. This allows us to use this model as prototype to understand the techniques of the different methods of RL including the use of function approximation.

In Chapter 4 (Model 2) we consider systems with a shared unique resource,

which may consist of several parallel-servers. Arrivals of orders take place continuously, however, arrivals are only evaluated at discrete equidistant time moments, so we generally consider several arrivals at each decision moment (batch arrivals). The probability of arrival is order type dependent. Accepted orders are to be loaded over a planning horizon consisting of a fixed number of stages. Each stage spans the time between two consecutive decision moments. Each order may be split arbitrarily over different stages (preemption) as long as its due-time is not violated. Per stage a maximum regular capacity is available at the resource to process the orders. Of every incoming order the required capacity is known. However, with a certain probability perturbations in the required capacity may occur. It is possible to use non-regular capacity at a certain cost, in order to avoid possible violations of due-times for the accepted orders that may be caused by perturbations in the required capacity.

In order to reduce the number of possible decisions we impose some restrictions on the structure of the decision rule. Instead of focusing on all possible subsets of orders at once, as possible decisions, we impose that the decision is created sequentially, while we call a time off. Each single decision in the sequence is either the selection of one of the orders from the arrival list or the rejection of all of them. For the capacity planning we use fixed prescription rules: Backward Loading and Least Shift Back.

In Chapter 4 we also present a characterization of the SMDP policies by means of a general class of heuristics. Two subclasses of this class of heuristics play an important role in this thesis: the one-parameter family of heuristics OrderQuality and the multi-parameter family of heuristics CapacityLevel. According to the OrderQuality heuristics only orders with a reward per unit of processing time above a threshold may be accepted. This threshold is related to the avoidance of opportunity losses. CapacityLevel heuristics employ a capacity level threshold for each type of order. This threshold stands for the maximally admissible fraction of the total capacity in the planning horizon used by all the accepted orders after accepting an order of the type under consideration. This family of heuristics includes the policies that reserve capacities for the most valuable orders. Motivation for these heuristics is the consideration that, in case of penalization for excess capacity, it might be better to fill capacity only up to a certain level of utilization. Thus there is a safety margin for dealing with perturbations due to non-anticipated extra capacity demand during job execution. We use results of both families of heuristics to compare them with the results from the policies learned by the RL-agents.

An essential contribution of Chapter 4 is the development of a framework for interpreting the learned RL-policies and to elaborate new advanced heuristic policies. Though developed in the specific context of the OA problem at hand, the framework is very general. In fact, it applies to any decision problem under uncertainty. The framework is built up out of three steps. In the first step we train the RL-agent and after having done so perform a large series of iterations with it. In the second step we employ a simple data mining technique to extract explicit knowledge about action preferences from the performed series of iterations. In the third step we identify heuristic rules from this action preference knowledge. The method laid down in our framework is powerful and universal. We use it throughout the Chapters 4, 5 and 6.

In Chapter 5 (Model 3) we consider an extension of Model 2 from Chapter 4. Here the server system is a multiresource job-shop where the arriving orders are built up out of a number of jobs with fixed routes along the resources. The processing times of the jobs are stochastic. The system state at an arbitrary decision moment consists on the one hand of the list of arrived orders, on the other hand of the capacity profile, which in turn contains information about the jobs being executed, but also about the accepted jobs that are waiting for execution. A complex structure, since each of these accepted jobs should have an indication of its possible starting time (or its predecessor job) in order not to violate the precedence relations. To avoid blinding the RL-agent with an overwhelming amount of system state information we provide the agent – so to speak – with sunglasses: as RL-input we use features, simplifications of the state representation. Of course, the actions rendered by the agent to the environment in each iteration are affecting the real system states. As feature for the list of arrived orders we give to the RL-agent only the order types that fit within the available capacity on the basis of a tentative EDD loading plan. As feature for the capacity profile we give to the RL-agent the loading plan that emerges by adding - according to EDD - the jobs that are waiting for execution to the jobs that are being executed. As in Chapter 4 we use a time off. We evaluate an incoming order on the basis of a tentative EDD loading plan in which we reschedule the new and the waiting jobs, leaving in peace the jobs that are being executed.

Also in Chapter 5 we present a methodology for the automatic tuning of the training parameters for the RL-agents. This methodology is based on experimental results and reduction of the number of free parameters.

In Chapter 6 (Models 4, 5 and 6) we consider OA problems with an underlying decision problem. To simplify the discussion we assume that the orders are subdivided into at most two jobs. In Model 4 we consider routing decisions in an open job-shop. The problem in this case is similar to the multiresource problem. However, the description of the orders does not include a fixed routing and a decision has two parts: as before one is which orders to accept, and the other part is for each of the accepted orders which routing should be used.

In Model 5 we return to the multiresource job-shop of Model 3. However, we add the possibility of deciding to outsource part of an incoming order. For simplicity reasons we only consider the situation that the second job of an order may be outsourced, so there are two outsourcing options. Not outsourcing an order means that both jobs of the order will be processed in the shop. Outsourcing an order means that the first job of the order will be processed in

the shop and the other one will be outsourced. Furthermore there could be a limit in the total amount to outsource in each time period.

In Model 6 we consider - in the multiresource job-shop of Model 3 - some freedom in the selection of a due-time for an accepted order. Associated to each due-time option there is a reward for acceptance. Furthermore there is a certain probability that the customer will agree on the selection of that option. For simplicity reasons we consider orders with two due-time options. A decision has two parts, as before one is which orders to accept, and the other part is for each of the accepted orders which due-time to agree on. A substantial difference with the other problem settings is that the customer could decide to quit the system.

Summarizing, the main contributions of this thesis are: new semi-Markov decision models for OA under uncertainty, a general class of heuristics for OA, a methodology for tuning the RL training parameters and a general framework for generating new heuristic rules for decision problems under uncertainty by using RL. All these contributions seen as a whole, define a general RL-metaheuristic for the application of RL to OA. We believe this general approach is applicable to a multitude of logistic decision problems.

In this thesis we present the analysis of 13 cases. In the first three cases, where the state space is quite small, the results of the RL-agents are compared with optimal policies. In all the three cases RL-agents outperform the greedy policies and approximate the results of optimal policies. For the other cases we use parametric subclasses of a general class of heuristics to compare with. In all these 10 cases, the final results of the RL-metaheuristic outperform the results of the parametric heuristics.

In conclusion, we may say that our RL-metaheuristic is a valuable option for obtaining advanced heuristic policies especially for problems with a large state space, and such that the decision making has to be done under uncertainty.

Samenvatting

Centraal in dit proefschrift staat de aanpak van problemen rondom orderacceptatie (OA) onder onzekerheid door middel van Reinforcement Learning (RL). OA is een fundamenteel bedrijfskundig vraagstuk dat - voor zover bekend - nog niet eerder in deze context is bestudeerd. Orders met uiteenlopende karakteristieken komen stochastisch aan bij een bewerkingsstation; op grond van de verwachte totale opbrengst dient men te beslissen of een binnenkomende order al dan niet wordt geaccepteerd. Niet geaccepteerde orders gaan verloren. RL is een probleemaanpak die bij uitstek geschikt is om onzekerheid te modelleren en bovendien in staat is om via leermethoden om te gaan met onvolledige informatie. Bij het modelleren van onzekerheid bouwen we voort op de theorie van de semi-Markov beslissingsproblemen (SMDP). Belangrijk hierbij is om OA-problemen onder onzekerheid zodanig te modelleren dat RL kan worden toegepast. RL kent op uiteenlopende gebieden een aantal succesvolle toepassingen, die niet altijd voldoende theoretisch zijn onderbouwd. Met name het probleem van het kalibreren van de trainingsparameters voor RL-agenten is in de literatuur niet in een algemeen theoretisch kader geanalyseerd. Dat laatste vormt zowel een uitdaging als een bedreiging. Het kan worden opgevat als een belangrijk punt van zorg voor deze studie. Een niet minder belangrijk aandachtspunt is de vraag hoe toegang te krijgen tot de black box gevormd door het brein van de RL-agent. Hoe dienen we de resultaten die in de loop van het leerproces zijn verkregen te interpreteren? Hoe vertalen we die resultaten in nuttige heuristieken voor het oorspronkelijke OA-probleem?

In Hoofdstuk 1 kenschetsen we de OA-problematiek. We definiëren onze onderzoeksvragen en geven een overzicht van relevante literatuur. Verder geven we een voorbeeld dat illustratief is voor de te volgen onderzoekslijn. Hoofdstuk 2 resumeert de basisprincipes van RL en specificeert onze aanpak van RL voor OA. In Hoofdstuk 3 beginnen we met een eenvoudig OA-probleem en definiëren ons eerste OA-model. In Model 1 beschouwen we systemen met één bedieningsstation zonder wachtruimte en een eindig aantal ordertypes met deterministische bewerkingstijden. Orders komen individueel en stochastisch binnen. Het bijbehorende SMDP laat zich eenvoudig oplossen mits de probleemdata expliciet bekend zijn. Als optimale oplossing vinden we een policy met een drempelwaarde-structuur. Interessant is nu, dat we de gevonden optimale oplossing kunnen vergelijken met resultaten verkregen via RL-algoritmes. Aldus dient dit model als prototype om inzicht te verkrijgen in verschillende

RL-technieken, zoals het gebruik van functie-approximaties.

In Hoofdstuk 4 (Model 2) beschouwen we systemen met één resource. Orders komen continu binnen, maar worden slechts geëvalueerd op discrete tijdstippen met vaste tussenpozen. Per beslissingstijdstip dienen we doorgaans meerdere orderaankomsten te beschouwen (batch arrivals). De aankomstkansen zijn orderafhankelijk. Geaccepteerde orders dienen te worden ingepland over een planningshorizon ter grootte van een vast aantal fases, waarin een fase de tijdspanne is tussen twee opeenvolgende beslissingsmomenten. Orders mogen willekeurig worden opgesplitst over verschillende fases (pre-emptie) mits de order-due-time niet overschreden wordt. Per fase is er bij de resource een vaste hoeveelheid reguliere capaciteit beschikbaar om aan de orders te werken. Van iedere binnenkomende order is de vereiste capaciteit bekend; we bouwen echter een waarschijnlijkheid in van fluctuaties in de benodigde capaciteit. De mogelijkheid bestaat om tegen zekere kosten niet-reguliere capaciteit in te huren teneinde voor de reeds geaccepteerde orders overschrijding van due-times te voorkomen, indien er onverhoopt meer capaciteit nodig is dan aanvankelijk begroot. Om het aantal mogelijke beslissingen terug te brengen modificeren we het beslissingsproces. In plaats van - op ieder beslissingsmoment - alle mogelijke deelverzamelingen van binnengekomen orders te beschouwen, gaan we als volgt te werk. We zetten de tijd even stil (time off) en evalueren vervolgens één voor één de binnengekomen orders. Iedere beslissing tijdens de time off is ofwel de selectie van één van de orders van de aankomstlijst ofwel de verwerping van alle binnengekomen orders. Een verdere vereenvoudiging is dat we voor het inplannen van de orders gebruik maken van twee simpele, maar plausibele capaciteitsplanningsregels: Backward Loading en Least Shift Back.

Eveneens in Hoofdstuk 4 presenteren we een karakterisering van de SMDP-policies door middel van een algemene klasse heuristieken. Twee subklassen van deze klasse heuristieken spelen in dit proefschrift een belangrijke rol: de één-parameter familie heuristieken OrderQuality en de multi-parameter familie heuristieken CapacityLevel. Volgens OrderQuality mogen alleen orders worden geaccepteerd met een opbrengst per eenheid van bewerkingstijd die boven een drempelwaarde ligt. De drempelwaarde wordt zo gekozen dat het verlies van veelbelovende orders - doordat capaciteit wordt toegewezen aan onbeduidende orders - zoveel mogelijk wordt vermeden. Bij de familie heuristieken CapacityLevel wordt voor ieder ordertype een drempelwaarde gehanteerd voor de maximaal toelaatbare fractie van de totale capaciteit in de planningshorizon die na acceptatie van een order van het desbetreffende type door de gezamenlijke geaccepteerde orders wordt gebruikt. Onder deze familie heuristieken bevinden zich policies die capaciteit reserveren voor de meest waardevolle orders. Met het oog op de extra kosten die capaciteitsoverschrijding met zich meebrengt is het raadzaam om - via heuristieken als deze - de capaciteit van de resource slechts tot een bepaald niveau te benutten. Op deze wijze wordt een veiligheidsmarge gecreëerd tegen onverwachte verstoringen in de orderverwerking die leiden tot extra capaciteitsbehoefte. We gebruiken de resultaten van beide families heuristieken om ze te vergelijken met de resultaten van de policies die de RL-agenten hebben geleerd.

Een essentiële bijdrage van Hoofdstuk 4 is de ontwikkeling van een raamwerk om de geleerde RL-policies te interpreteren en deze om te zetten in nieuwe geavanceerde heuristieken. Hoewel dit raamwerk is ontwikkeld in de specifieke context van het voorliggende OA-probleem, is het zeer algemeen van aard. In feite kan het worden toegepast op ieder beslissingsprobleem onder onzekerheid. Het raamwerk is opgebouwd uit drie stappen. In de eerste stap trainen we de RL-agent en voeren vervolgens met de getrainde agent een lange reeks iteraties uit. In de tweede stap gebruiken we een eenvoudige data mining techniek om uit de uitgevoerde serie iteraties expliciete kennis af te leiden omtrent preferentierelaties in specifieke deelruimtes van de ruimte van mogelijke acties (= beslissingen). In de derde stap vertalen we de opgedane kennis omtrent actiepreferenties naar heuristieke regels. De in ons raamwerk vastgelegde methode is krachtig en universeel. We gebruiken hem voortdurend in de Hoofdstukken 4, 5 en 6.

In Hoofdstuk 5 (Model 3) beschouwen we een uitbreiding van Model 2 uit Hoofdstuk 4. Hier is het bedieningssysteem een multiresource job-shop waarbij de binnenkomende orders opgebouwd zijn uit een aantal jobs met voorgeschreven routes langs de resources. De bewerkingstijden van de jobs zijn stochastisch. De systeemtoestand op een willekeurig beslissingsmoment bestaat enerzijds uit de lijst binnengekomen orders, anderzijds uit het capaciteitsprofiel, hetgeen informatie bevat over de jobs die in bewerking zijn, maar ook over de jobs die geaccepteerd zijn en op bewerking wachten. Een complexe structuur daar elk van deze geaccepteerde jobs een indicatie dient te hebben van zijn mogelijke starttijd (of van zijn voorganger) teneinde de precedentierelaties niet te schenden. Om de RL-agent niet te verblinden met een overdaad aan toestandsinformatie zetten we de agent als het ware een zonnebril op door als RL-input gebruik te maken van features, simplificaties van de toestandsrepresentatie. Uiteraard grijpen de acties die de agent in iedere iteratie aan de omgeving teruggeeft in op de echte toestanden. Als feature voor de lijst binnengekomen orders kiezen we alleen die ordertypes die op grond van een voorlopig EDD toewijzingsplan passen binnen de beschikbare capaciteit. Als feature voor het capaciteitsprofiel kiezen we het toewijzingsplan dat ontstaat door de jobs die op bewerking wachten via EDD toe te voegen aan de jobs die reeds in bewerking zijn. Ook nu gebruiken we een time off als in Hoofdstuk 4. We evalueren een binnengekomen order op grond van een voorlopig EDD toewijzingsplan waarbij we de nieuwe en de wachtende jobs reschedulen, terwijl we de jobs in bewerking met rust laten.

Verder introduceren we in Hoofdstuk 5 een methodologie voor het automatisch kalibreren van de trainingsparameters voor de RL-agenten. Deze methodologie is gebaseerd op experimentele resultaten en reductie van het aantal vrije parameters.

In Hoofdstuk 6 (Modellen 4, 5 en 6) beschouwen we OA-problemen voorzien van een onderliggend beslissingsprobleem. Eenvoudigheidshalve nemen we aan dat binnenkomende orders bestaan uit ten hoogste twee jobs. In Model 4 beschouwen we routeringsbeslissingen in een open job-shop. Het probleem is identiek aan het multiresource probleem met dit verschil dat er nu niet per order een routering langs de resources is voorgeschreven. Een beslissing bestaat uit twee delen: welke orders accepteren we en welke routering kiezen we per geaccepteerde order.

In Model 5 keren we terug naar de multiresource job-shop van Model 3, maar voegen de mogelijkheid toe om een deel van een binnenkomende order uit te besteden. Voor de eenvoud nemen we aan dat alleen de tweede job van een order mag worden uitbesteed. Per geaccepteerde twee-jobs-order zijn er twee opties: (1) niet uitbesteden, hetgeen betekent dat beide jobs in de shop worden uitgevoerd; (2) uitbesteden, waarbij de eerste job in de shop wordt uitgevoerd en de tweede elders. We bouwen de mogelijkheid in om een grens stellen aan de totale uitbesteding in een bepaald tijdsinterval.

In Model 6 laten we - in de multiresource job-shop van Model 3 - enige speling toe in de due-time van een geaccepteerde order. Bij iedere due-time optie hoort een specifieke opbrengst. Verder is er per optie een bepaalde waarschijnlijkheid dat de klant ermee akkoord zal gaan. Om de complexiteit te beperken beschouwen we per order slechts twee due-time opties. Een beslissing bestaat uit twee delen: welke orders accepteren we en welke due-time stellen we aan de klant voor per geaccepteerde order. Een essentieel verschil met de voorgaande probleemsituaties is dat de klant kan besluiten het systeem te verlaten.

De belangrijkste bijdragen van dit proefschrift laten zich als volgt samenvatten: nieuwe semi-Markov beslissingsmodellen voor OA onder onzekerheid, een algemene klasse heuristieken voor OA, een methodologie om de RL-trainingsparameters in te stellen en een algemeen raamwerk om nieuwe heuristische regels te genereren voor beslissingsproblemen onder onzekerheid door gebruik te maken van RL. Het geheel van deze bijdragen kan worden opgevat als een algemene RL-metaheuristiek voor de toepassing van RL op OA. Naar onze overtuiging is deze algemene aanpak toepasbaar op een veelheid van andere logistieke beslissingsproblemen.

In dit proefschrift analyseren we 13 probleeminstanties. In de eerste drie daarvan is de toestandsruimte klein, hetgeen ons in staat stelt de resultaten van de RL-agenten te vergelijken met optimale policies. In alledrie de gevallen presteren de RL-agenten beter dan de greedy policies en benaderen ze de resultaten van optimale policies. Voor de andere 10 instanties vergelijken we onze RL-resultaten met die van geparametriseerde deelklassen van een algemene klasse heuristieken. In alle 10 gevallen presteert de RL-metaheuristiek beter dan de geparametriseerde heuristieken.

We sluiten af door te stellen dat onze RL-metaheuristiek een waardevolle optie is om geavanceerde heuristieke regels te genereren, in het bijzonder voor problemen met een grote toestandsruimte, waarbij het beslissingsproces plaatsvindt onder onzekerheid.

About the author

Marisela Mainegra Hing was born on April 15, 1972 in Santiago de Cuba, Cuba. In 1990 she began to study Computer Science at Central University of Las Villas, Santa Clara, Cuba, obtaining her B.Sc. diploma in 1995. Two years later she obtained her M.Sc. in Mathematics with a specialization in Combinatorial Optimization, at the same university. In 1997 she attended the master class program in Operations Research organized by the Mathematical Research Institute in The Netherlands. In 1998 she started as a Ph.D. student at the Technology and Management Faculty of the University of Twente on a research project "Intelligent Computational Techniques supporting Learning Organizations" which resulted in this thesis. Since January 2003 she is assistant professor at the University of Santa Clara, Cuba.