
Proceedings of the 9th International Conference on Neural Information Processing (ICONIP'02), Vol. 3
Lipo Wang, Jagath C. Rajapakse, Kunihiko Fukushima, Soo-Young Lee, and Xin Yao (Editors)

REINFORCEMENT LEARNING FOR ORDER ACCEPTANCE ON A SHARED RESOURCE

M. Mainegra Hing, A. van Harten, P. Schuur

Dept. of Technology and Management, University of Twente, The Netherlands

ABSTRACT

Order Acceptance (OA) is one of the main functions in business control. Basically, OA involves for each order a reject / accept decision. Always accepting an order when capacity is available could disable the system to accept more convenient orders in the future with opportunity losses as a consequence. Another important aspect is the availability of information to the decision-maker. We use a stochastic modeling approach, Markov decision theory and learning methods from Artificial Intelligence to find decision policies, even under uncertain information. Reinforcement Learning (RL) is a quite new approach in OA. It is capable of learning both the decision policy and incomplete information, simultaneously. It is shown here that RL works well compared with heuristics. Finding good heuristics in a complex situation is a delicate art. It is demonstrated that a RL trained agent can be used to support the detection of good heuristics.

1. INTRODUCTION

One of the main functions in a business control framework is Order Acceptance (OA). It is essential to manage the interface between customers and production. Traditionally this problem is solved by always accepting orders, greedily with respect to immediate profits, as long as sufficient capacity is available. However, always accepting an order when there is capacity available is myopic. Such a policy can disable the system to accept more profitable orders in the future (opportunity loss). Another important aspect 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 due date and the price are assumed to be known or estimated. Further, a model of the production process and the planning procedure for execution of the jobs is also considered to be known beforehand. But most of the time it is difficult to have such information exactly. For example, there might be uncertainty in job execution.

Here we analyze the trade-off between on one hand, capacity reservation to avoid opportunity losses and to have a safety margin for disturbances and on the other hand, immediate yield in case of order acceptance under uncertainty. Reinforcement Learning (RL) is a quite new approach that combines very well with the idea of Markov decision modeling and its solution methods based on value functions. Further, there are some other aspects that make RL appealing to 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, not only with respect to job execution, but also with respect to other parameters such as arrival rates of certain types of jobs. 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.

RL is a rather new field and successful applications are not always fully understood yet on a theoretical level. It means that convergence properties of the algorithms and procedures for tuning the parameters have to be explored. Hence, a lot of work still has to be done in order to understand how RL can best be applied to business problems in various areas and to get insight into the RL outcomes. Altogether this problem area constitutes a new and interesting field for several lines of research. In this paper we focus on the possible contributions to order acceptance. Our goal is to compare decision policies found by an RL trained agent with heuristics.

In Section 2 we introduce the OA situation under study and model it as a semi-markov decision (SMD) problem. In Section 3 we specify the Reinforcement Learning techniques used. In Section 4 we introduce a new general class of heuristics for the purpose of comparison with and interpretation of RL trained agents. In Section 5 we report on computational results comparing RL trained agents with (advanced) heuristics. Finally we give our conclusions, summarize the acquired new insight and formulate some ideas for further research.

2. OA CASE DESCRIPTION

Order definitions are based on a finite number N of job types, where type i has an arrival rate (λ_i), due-date (t_i), requested capacity (w_i) and pays an immediate reward (r_i) upon acceptance. The number of arrivals in a unit time interval follows a Poisson distribution. Each job asks for service on a shared resource, which can process different jobs at the same time (i.e. concurrency is allowed). A job is divisible in the sense that the requested capacity could be distributed over a planning time horizon of the capacity, just before the due-date of the job. The planning time horizon with H stages. Each stage spans one time unit. There is a maximum capacity $C_{\max}$ per stage that can only be excessed at the cost of a penalty $pen(c,p)$ where $c = (c_1,..,c_2)$ is the capacity profile and p is a stochastic perturbation term. By c_t we refer to the occupied capacity at stage t of the planning horizon. The stochastic perturbation may be due to workload realization which differs from the expectation, causing a possible occupied capacity perturbation. For simplicity we consider a random perturbation term $p \in \{-1,0,1\}$ which affects the capacity profile c_t only for $t = 1$. 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 $c_1 > 0$. The probability of a perturbation p is denoted by $Pr(p)$.

The state at each decision moment is defined by $s = (k,c)$. $k = (k_1,...,k_N)$ is the job list and k_i represents the amount of jobs 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. For each job type i we restrict the maximum amount of jobs in the job list to m_i. This number may be determined by the arrival probabilities or by the limited capacity. The state space $S = \{s\}$ is the set of all possible states and its cardinality is $(C_{max}+1)^H \prod_{i=1}^{N}(m_i+1)$.

The capacity profile is updated given the decision (I) ($i > 0$) of accepting a job of type i, or (II) ($i = 0$) rejecting the complete job list. In case (I) we choose for a "just in time" (JIT) principle, since allocating capacity as close to the due date as possible provides the best conditions for accepting more jobs from the job list. Case (II) is completely different. If there is still available capacity in the first stage ($t = 1$), then it is lost unless we adapt the allocation. We may create the best conditions for accepting jobs 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.

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 job type i is present in the job list and capacity is available for that job type, rejection of the complete job list is always an option (i.e., $A(s) = \{i \in [1..N] \,|k_i \neq 0, JIT(c,i)$ is possible$\} \bigcup \{0\}$. In case a job type i is chosen an immediate reward r_i is received and a transition to the next state s' occurs deterministically with the *time off*, i.e. the elapsed time $d(s,i) = 0$. The new job list is given by $k - e_i$ where e_i is a unit vector with 1 in position i and the capacity profile is updated with the procedure $JIT(c,i)$. In case the job list is rejected ($i = 0$) the immediate reward is stochastically dependent on the capacity perturbation (p) and is given by the penalization function. 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 jobs is considered. The new arrivals vector k' is determined by the jobs 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 state transition is described in Table 1.

s	(k,c)	
i	i∈A(s)	0
d(s,i)	0	1
s′	(k-e$_i$, JIT(c,i))	(k', LSBp(c,p))
Pr(s,i,s')	1	Pr{k'}Pr{LSBp(c,p)}
rew(s,i,s′)	r$_i$	pen(c,p)

Table 1: state transition

The objective is to find a policy π a mapping $S \rightarrow A$ which maximizes the performance of the system. The performance of the system is measured as the expected value of the total discounted reward with γ the discount factor and $d(s,i)$ is the elapsed time as introduced before. The optimal action value function Q^* in this case satisfies:

$$Q^*(s,i) = \sum_{s'} Pr(s,i,s') \left[rew(s,i,s') + \gamma^{d(s,i)} \max_{i'} Q^*(s',i') \right] \quad (1)$$

and the optimal policy $\pi^*(s)$ can be determined by:

$$\pi^*(s) = \arg\max_i Q^*(s,i) \quad (2)$$

In this problem even in the case that the parameters of the model are known solving the problem is still a difficult task due to what is usually referred to as the "curse of dimensionality". A simple policy structure "*accept if capacity is available and the reward per unit of requested capacity is sufficiently high*" as found in the prototype problem in [2] 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

jobs which generate low profit per unit of requested capacity than in case of high utilization. Further one should expect that even in case of high utilization certain small rush jobs, 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 jobs. 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.

3. REINFORCEMENT LEARNING

In the previous sections we have argued that in order to support order acceptance decisions we need to explore new alternatives able to cope with long term opportunity loss and uncertain environments with incomplete information. In this section we deal with one of such possible alternatives: Reinforcement Learning. An RL system basically consists of two components, an agent and its environment.

An RL-agent is characterized by a knowledge structure, a learning method to update its knowledge and a specific behavior (policy). The environment is assumed to be in general an SMD process where the actions are controlled by an agent. In this context we can situate our OA problem with incomplete information by considering the decision maker as an agent who has to act in a (partially) unknown environment where orders arrive and have to be accepted or rejected. Specifically in our problem we consider incomplete information in the sense that the agent does not know the frequency of order arrivals. Using the model described in Section 2 we simulate the dynamics of the environment. Then through interaction an agent who does not know about such dynamics, should learn a good policy through the learning method. Order characteristics become available upon arrival. Next we discuss the agent knowledge representation, learning method, behavioral structures and related parameters. For an introduction on RL see [1] and [3].

Our focus here is on Q-Learning (QL) methods that aim to learn the optimal action value function (1) as a way to obtain an optimal policy. The method is based on estimations of the Q-values that are updated after each agent-environment interaction. At each decision moment t in which the environment is in state s_t the agent chooses an action a_t based on its current estimation $Q_t(s,a)$ and some specific policy. Examples of policies are a greedy policy[1] and an exploratory policy[2], but we will use an in between policy. Exploration increases experience by for example choosing actions at random. Exploitation deals with the use of the available knowledge by for example choosing the greedy actions using (2).

The update rule for this method given the experience tuple $< s_t, a_t, r_{t+1}, s_{t+1} >$ is as follows:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha_t \delta_t$$

$$\delta_t = r_{t+1} + \gamma^{d(s_t,a_t)} \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t) \quad (3)$$

Agents are trained for a period of L iterations, considering one iteration as an agent-environment interaction as explained before. Here we use a policy which at iteration k chooses with probability ε_k for the exploratory policy and with probability $1-\varepsilon_k$ for the greedy policy. We use decreasing functions as in [2] for the exploration and learning parameter. At iteration k the learning and exploration parameters are

$$\varepsilon_k = \frac{\varepsilon_0}{1+\frac{k}{T_e}}, \; \alpha_k = \frac{\alpha_0}{1+\frac{k}{T}} \quad (4)$$

where ε_0 and α_0 are initial values and T_e, T define the decreasing speed respectively.

The knowledge representation in this method concerns the Q-values representation. Here we use parametrized function approximations that can generalize and interpolate for states and actions never seen before. An Artificial Neural Network (ANN) is an example of such a parametrized function approximation with a massively parallel distributed structure. Such a structure and the capability to generalize make it possible for ANN to solve complex problems. Here we use a 2-layer perceptron with backpropagation training, m hidden neurons and one output. The number of parameters in such an ANN is kept moderately low, thus avoiding the curse of dimensionality of the full state space. The inputs of the ANN are the state of the system s and an action a for that state. For the state we use N integer inputs showing the amount of jobs of each type and H integer inputs showing the allocated capacity at each stage of the capacity profile. For the actions we use an integer input that takes values from 0 to N indicating rejection of the current job list or the type of job chosen. The output is then the corresponding Q-value $Q(s,a)$.

4. A GENERAL CLASS OF HEURISTICS

A general class of heuristics for the OA problem can be defined as follows. Given the state space S and the action space A we introduce for each state $s \in S$ the set of allowed decisions $A(s) \subset A$. Next we introduce a linear ordering $\succ$ on the action space that defines preference relations. We denote with i_k the action in position k with respect to this ordering: $\theta = i_1 \succ i_2 \succ ..... \succ i_N \succ i_{N+1}$ so job type i_1 is preferred over job type i_2. A heuristic policy π can be

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

[2] 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.

defined that assigns to each state $s \in S$ an allowed action from $A(s)$ considering the defined ordering as follows:

$$\pi(s) = \max_{\theta} A(s) \tag{5}$$

The exact optimal policy π^* can be represented in this form by defining unitary subsets $A(s) = \pi^*(s)$ and the ordering does not matter. Also, the RL-trained agent can be represented in this way by putting $A(s) = \arg\max Q(s,a)$ and again the ordering is not important. But of course, finding the subsets $A(s)$ and the actions ordering is just as difficult as solving the Markov decision problem with its curse of dimensionality exactly.

To illustrate the flexibility of this approach to heuristics we consider next some more 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 (ρ) of the full capacity. An example of this sort is the optimal policy for a simple OA problem in [2]. Here we consider the following family of policies:

- $A(s) = \left\{ i \in A | \frac{r_i}{w_i} \geq b,\ cap(s,i) \leq HC_{\max} \right\} \bigcup \{0\}$
- ordering: $i_{N+1} = 0,\ i \succ j$ if $\left(\frac{r_i}{w_i} \geq \frac{rj}{wj} \right)$

Here $cap(s,i)$ denotes the occupied capacity after accepting job i when in state s. $HC_{\max}$ is the total capacity in the planning horizon. The threshold b is related to the avoidance of opportunity losses. In the next section this sort of policies will be referred to as $Jobquality(b)$.

5. COMPUTATIONAL EXPERIMENTS

We present the application of RL methods as discussed in Section3 to some cases of the OA problem described in Section 2 with Poisson arrivals, and discount factor $\gamma = 0.9$. We compare the RL policies with some heuristic policies from Section 4. As a performance measure we use the Total Average reward. The graphics show the total Average Reward of agents following the different policies. For the RL agents we show only the results after a period of training (Q-values are not updated anymore). Table 2 lists the parameters for the RL-agents, L is the number of iterations for training the RL agent, $\alpha_0, T, \varepsilon_0, T_\varepsilon$ are the parameters of the learning and exploration rate as in (4) and m is the number of hidden neurons. These parameters are chosen trough experimental experience and are by no means optimized. For more details on the effects of changing the number of hidden neurons and/or the exploration and exploitation functions in the training algorithm of our ANN we refer to [2]. In the graphics the initial behavior, before the lines become almost constant, are due to the samples needed for convergence of the long-run average reward.

	L	α_0	T	ε_0	T_ε	m
case 1	40000	0.001	400	1	1000	500
case 2	500000	0.001	5000	1	5000	100

Table 2: RL parameters

To test the RL approach we use two cases with different complexity. The first case does not consider capacity perturbation. In the last case we consider capacity perturbation, penalization for using non-regular capacity and different $\frac{r_i}{w_i}$. In this last case the capacity profile is simplified in order to reduce the state space and be able to make a better interpretation of the results. Both cases have 5 job types with the same arrival rate $\lambda_i = 0.4$, the same due-dates $t = t_i$, and $m_i = 2$ is the maximum number of arrivals to be considered.

case (1): Job type i request $i + 5$ units of capacity and the reward for acceptance is such that that $\frac{r_i}{w_i} = 1 + \frac{10}{i}$ is decreasing $\frac{r}{w} = [11,\ 6,\ 4,\ 3.5,\ 3]$. $(t = 5, C_{max} = 4, H = 5. |S| = 5103)$. The best of the family of heuristics is $Jobquality(6)$ which only accepts jobs type 1 and 2 in that order. This threshold illustrates how opportunity costs are avoided by always rejecting jobs with $\frac{r_i}{w_i} < 6$. The RL-agent approximates this heuristic with an average relative error of 0.43% in the average reward and outperforms the greedy policy ($Jobquality(3)$) in 9.25%. (see figure 1)

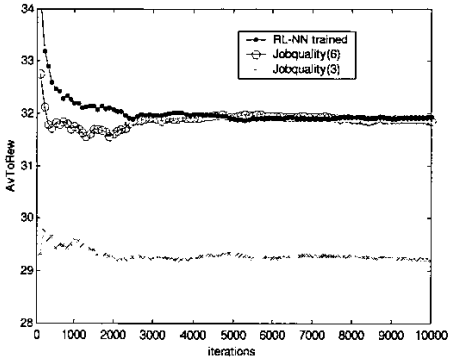

Figure 1: Case 1

case (2): Job type i request i units of capacity and the reward for acceptance is $r = (1, 4, 15, 16, 15)$ yielding the vector $r/w = (1, 2, 5, 4, 3)$. The perturbation values are [-1,0,1] uniformly distributed and there is a penalization ($\eta = 5$) for capacity disturbances $(t = 3, C_{max} = 2, H = 3, |S| = 1701)$

Here the situation is less clear. The best of the heuristics is $Jobquality(3)$. This heuristic only accepts jobs of type $3, 4$, and 5 whose $\frac{r_i}{w_i} \geq 3$ in that order. The RL-policy

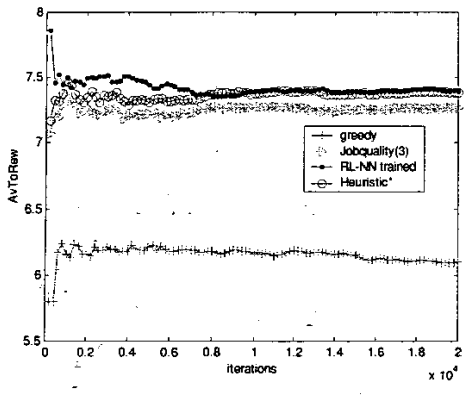

Figure 2: Case 2

outperforms this policy with an improvement of 2.32% and the greedy one in a 23.51%. (see figure 2)

Here it is necessary to introduce more flexibility in the heuristics. A better heuristic should deal with the issue of accepting jobs with a less higher $\frac{r_i}{w_i}$ but not risking too much for penalties. To find such heuristic we will use the RL-trained agent as our inspiration. It is interesting that the RL-policy can be described using the general structure as in (b) in the previous section by defining:

- $A(s) = \{i \in A \mid cap(s,i) \leq \rho_i HC_{\max}\} \bigcup \{0\}$
- ordering: $i_{N+1} = 0$, $i \succ j$ if $\left(\frac{r_i}{w_i} \geq \frac{rj}{wj}\right)$

Let us try to characterize the learned RL-policy in terms of this heuristic. The learned ordering is $3 \succ 4 \succ 5 \succ 2 \succ 1 \succ 0$ and $\rho_3 = \rho_4 = \rho_5 = 1, \rho_2 = 0.5, \rho_1 = 0$. This policy never accepts jobs of type 1 which is the least profitable one, but it is also the smaller one. Obviously, if no jobs of type $2,3,4,5$ are requesting service, the system is empty and a job of type 1 is requesting service, it is better to accept this job, otherwise the capacity at $t = 1$ would be lost without any purpose (there is no risk of penalization). The conclusion is that the RL-rule can still be improved by putting in the previous heuristic $\rho_1 = 0.5$. However considering this case (*Heuristic**) does hardly improve the RL policy performance as can be seen in the figure. Both policies behave quite similar, the relative error is 0.4%. The extra rule included in the *Heuristic** policy seldom occurs (0.022 frequency).

Recall that we use the heuristics as a mean of measuring the RL performance but as the opposite for the RL, the heuristics need to have complete information in order to be used. In more complex cases even higher flexibility may be needed in heuristics when jobs differ also in due dates and arrival rates. We have shown that training an RL agent is a good alternative.

6. CONCLUSIONS

We describe a RL-NN based decision support approach for order acceptance problems under uncertainty that take into account opportunity losses. This approach had shown good performance. In cases where good heuristic policies could be defined, the RL-NN approach approximates those heuristics. Recall that in order to define the used heuristics we need complete information about the problem case. Particularly finding the ρ parameters is not an easy task even with complete information.

In more complex situations our RL approach outperforms simple heuristic policies, building a mixture of simple heuristics. The RL-NN method has also a built in adaptability and generic design. The same RL-NN structure may be used to learn different policies for different cases. For online use the long training period may be improved by using model based simulation events. Here for a relative small size problem we use statistic information in order to interpret the RL-NN policy in terms of a general framework for heuristics. The interpretation of the RL-NN policy in general still requires development of adequate methods. We expect to find intelligent rules in the policies obtained with these methods. We believe the RL approach is a very flexible method that could help in the search for good order acceptance rules. Further studies could analyze extended models and the possibility of including OA-RL strategies in an integrated capacity planning approach. Furthermore other neural networks architectures may be explored that make better use of some information about the problem structure or some general a priory information about the model.

7. REFERENCES

[1] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[2] M. Mainegra Hing, A. van Harten, and P. Schuur. Order acceptance with reinforcement learning. WP 66, BETA, Technical University of Eindhoven, The Nederlands, dec 2001.

[3] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, London, England, 1998.