Average Reward Adjusted Deep Reinforcement Learning: Near-Blackwell-Optimal Policies for Order Release

Manuel Schneckenreither[1], Stefan Haeussler[1], and Juanjo Peiró[2]

[1] Department of Information Systems, Production and Logistics Management, University of Innsbruck, Austria
manuel.schneckenreither@uibk.ac.at,stefan.haeussler@uibk.ac.at
[2] Departament d'Estadística i Investigació Operativa, Universitat de València, Spain
juanjo.peiro@uv.es

Abstract. [MS: proof-of-concept work] An essential task in manufacturing planning and control is to determine when to release orders to the shop floor. One key parameter is the lead time which is the planned time that elapses between the release of an order and its completion. Lead times are normally determined based on the actual duration orders previously took to traverse through the production system (flow times). Traditional order release models assume static lead times, although it has been shown that they should be set dynamically to react the dynamic operational characteristics of the system. Therefore, we present an order release model which sets lead times dynamically by using an reinforcement learning approach. Therefore, we provide an in-depth analysis of reinforcement learning to show that average reward reinforcement learning is a better approach for operations research applications as discounted reinforcement learning. Additionally we present an average reward reinforcement learning algorithm which infers near-Blackwell-optimal policies. We use a simulation model of a [MS: todo] two-stage flow-shop to compare the algorithm to well-known order release mechanisms. We show that in the current version our proposed model using reinforcement learning outperforms some, but not all other tested approaches.

Keywords: operations research, production planning, order release, machine learning, reinforcement learning

1 Introduction

An important goal in Manufacturing Planning and Control (MPC) systems is to achieve short and predictable flow times, especially where high flexibility in meeting customer demand is required. Besides achieving short flow times, one should also maintain high output and due-date performance while keeping the work-in-process level low. One approach to address this problem is to collect all incoming orders in an order-pool and periodically decide which orders to release to the shop floor. Once orders are released, costs start to accumulate as planned

orders materialise as actual jobs in the production system. The main challenge is to find a good compromise of balancing the shop floor and timely completion of jobs. Although the performance of such systems can be measured manifold the most overarching objective is to maximise profits by adequately assigning holding and lateness costs.

One of the key modeling parameters for order release mechanisms is the *lead time*, which refers to the planned time that elapses between the release of an order and its arrival in the finished goods inventory. This planning parameter is often based on the observable time an order needs to traverse the production system, which in contrast is denoted as the *flow time*. Flow times consist of processing, setup, control, transport, and waiting times, whereas the latter is the governing factor (Zäpfel, 1982, p.223). Waiting times are a result from queuing (e.g. jobs queue before and after processing), depend heavily on the amount of jobs in the system (WIP) and thus are relatively difficult to estimate, which makes the setting of favorable lead times so difficult (e.g., Tatsiopoulos and Kingsman 1983; Wiendahl 1995). Most state-of-the-art order release mechanisms use static or fixed lead times to address the order release problem, and thus neglect the nonlinear relationship between resource utilisation and flow times, which is well known from practice and queuing theory (Pahl et al. 2007).

One way to address this nonlinear interaction effects is to set lead times dynamically. Intuitively the order release problem is solved by perfectly matching the lead times to the flow times, but the corresponding optimisation problem faces "sampling issues" meaning that the flow times depend on the lead times. An extreme scenario of this problem is the so called "lead time syndrome", which describes a vicious cycle where increasing flow times perpetually inflate the lead times which leads to worse performance (see e.g., Knollmann and Windt 2013; Mather and Plossl 1978; Selcuk et al. 2006. Thus, setting lead times dynamically harbors optimisation potential (Hoyt, 1978), but may also substantially degrade the system performance. Schneckenreither et al. (2020) have established following categorisation of dynamic lead time management approaches:

- Reactive lead time management approaches set lead times by reacting on earlier flow times (e.g., Enns and Suwanruji 2004; Selcuk et al. 2006). Note that the forecast is always based on *past* data as the most recent system changes cannot be reflected by flow times until the corresponding orders arrive in the FGI, which might take several periods.
- Proactive lead time management may incorporate *past* data in conjunction with the *current* system state to set lead times (e.g., Bertrand 1983; Chung and Huang 2002). Put differently, these methods aim to find a function that provides lead times based on the current state of the production system and possibly information from the past.
- Predictive lead time management may not only incorporate *past* data and the *current* system state to set lead times, but also utilises the anticipated *future* system state to detect arising issues of future periods and react accordingly (e.g. Paternina-Arboleda and Das 2001; Schneckenreither 2019; Schneckenreither et al. 2020). Thus, it extends proactive lead time management from

a flow time forecasting or simple lead time setting technique to a lead time management approach that integrates the future behaviour of the system when setting lead times. This allows reasoning of the system dynamics, as for instance triggering the lead time system, and thus such an algorithm can react accordingly to find a more farsighted optimal lead time update.

We hypothesis that dynamic lead time management approaches need to aim for a predictive design in order to be able to compete with state-of-the-art order release methods from literature. However, there only exist three papers that propose a predictive lead time management algorithm in literature.

The first approach by Paternina-Arboleda and Das (2001) introduces a predictive order release model by using reinforcement learning in a single product, four-station serial flow line and compare its performance (WIP costs) with conventional order release policies (e.g., Kanban and CONWIP). The algorithm of Paternina-Arboleda and Das (2001) decides on whether or not to release an order after each system change, the completion of an operation of any order at any stage or a new order arrival, and assumes that any unsatisfied demand is lost. Thus, they use a continuous order release method although in practice order release decisions often need to be made on a periodical basis, e.g. daily (see Enns and Suwanruji 2004; Gelders and Van Wassenhove 1982). They outperform existing control policies with their tabular based reinforcement learning agent. Then, Schneckenreither (2019) use several different reinforcement learning algorithms to make periodic order release decisions for a flow shop production system. The algorithm directly sets lead times for each product type, which are then used to release the orders in the order pool. They show that their approach outperforms static order release mechanisms by yielding lower costs, lateness and standard deviation of lateness, but conclude that research using average reward reinforcement learning methods harbor optimisation potential for the order release problem. And finally, Schneckenreither et al. (2020) present a flow time estimation procedure to set lead times dynamically using an artificial neural network, which is used to forecast flow times. By implementing a rolling horizon order release simulation of the proceeding periods they lift their approach to a predictive lead time management approach, which is able to detect backorders of future periods and reacts by releasing orders earlier. Nonetheless, their method is unable to foresee the triggering of the lead time syndrome and therefore they introduce an upper lead time bound to prevent the negative effects of the lead time syndrome. Their model outperforms static and reactive lead time management approaches, especially for scenarios with high utilisation and high variability in processing times.

As reinforcement learning stems from dynamic programming the future system state is by design considered as a main driver of decision making in the current period. Its goal is to find the best stationary policy for a given problem. The advantage of reinforcement learning over dynamic programming is that (i) the problem space is explored by an agent and thus only expectantly interesting parts of the problem space need to be assessed and (ii) the knowledge (acquisi-

tion) of transition probabilities becomes unnecessary as the states are evaluated by consecutively observed states solely.

Over the past decades reinforcement learning has been applied to various problems, for which astonishing results have been reported. E.g. only recently Mnih et al. (2015) have presented a novel value-iteration reinforcement learning agent which exceeds human-level abilities in playing many classic Atari 2600 games (Bellemare et al., 2012). Further, Mnih et al. (2016) present improved results with asynchronous actor-critic reinforcement learning. Also games like Go (Silver et al., 2016) and Chess (Silver et al., 2017) have been mastered with superhuman performance by *tabula rasa* reinforcement learning agents. Furthermore, the method has also been applied in the setting of manufacturing system, e.g. to improve the ramp-up process (Doltsinis et al., 2012), in locally selecting appropriate dispatching rules (Wang and Usher, 2005; Zhang and Dietterich, 1995) or scheduling (Waschneck et al., 2018; Zhang and Dietterich, 1995). However, all these applications use discounted reinforcement learning and are either designed to investigate a rather simple MDP, e.g. by selecting heuristics instead of optimising the underlying problem itself, or by mapping the actual objective in a reward function that is approximately 0 on average over time. This is due to the fact that state value is largely composed of a term defined by the policys' average reward value (Blackwell, 1962; Miller and Veinott, 1969) which would otherwise dilute the state values and thus decrease the solution quality, as can for instance be observed in Schneckenreither and Haeussler (2019) and Gijsbrechts et al. (2018).

Therefore, most applications that incorporate and directly reflect costs or profit in the reward function use average reward reinforcement learning. Aydin and Öztemel (2000) use it in the setting of scheduling, while in a series of papers Mahadevan et al. investigated several problem domains starting with simple MDPs (Mahadevan, 1996b,c). After these foundational works for average reward reinforcement learning they introduced a continuous time average reward reinforcement learning algorithm named SMART (Mahadevan et al., 1997). Applications of SMART reach from the optimisation of queuing systems (Mahadevan, 1996a,d), maintenance of an inventory system (Das et al., 1999) to optimising transfer line in terms of maximizing demand, while keeping inventory levels of unfinished product as low as possible (Mahadevan and Theocharous, 1998). However, in practise usually decision have to be made on a daily basis (Enns and Suwanruji, 2004; Gelders and Van Wassenhove, 1982). Therefore, we refrain from this adaption and concentrate on standard MDPs only. Furthermore, often continuous-time semi-MDP problems can be converted through uniformisation into equivalent discrete time instances (see Bertsekas et al., 1995; Puterman, 1994).

Like discounted reinforcement learning also average reward reinforcement learning is based on an oracle function, in our case the accumulated costs of a period, to assess the decisions taken by the agent. By repeatedly choosing different actions the agent examines the problem space and rates possible actions for any observed state. The advantage of average reward reinforcement learning

over the widely applied discounted reinforcement learning framework is that the underlying optimisation technique is able to find better policies. This yields from the fact that in standard discounted reinforcement learning method the states are assessed independently and by a single value. To be more precise, average reward reinforcement learning splits up the evaluation of the average reward per step, a bias value that specifies the amount of collected rewards to reach the optimal path when starting in a suboptimal state and an error term which defines the number of steps to reach the optimal path (Howard, 1964; Mahadevan, 1996b; Puterman, 1994). In contrast to that in the standard discounted framework one single scalar value consisting of the addition of these subterms is estimated, where however the average reward is scaled by $1/(1-\gamma)$. The discount factor γ is usually set very close to 1, e.g. 0.99, which leads to the fact that this subterm dominates the other two. Further, in commonly applied unichain (and thus ergodic) MDPs the average reward per step is equal for all states. Thus, independently assessing it is not only computationally unwisely, but also problematic when iteratively annealed as done in reinforcement learning. Therefore, we adapt an algorithm that uses a scalar value for the estimation of the average reward over all states, and estimates for every state-action pair that incorporate the bias and error term. The later values are adjusted by the average reward.

Thus, as opposed to the commonly applied discounted reinforcement learning algorithm we use an average reward adjusted reinforcement learning algorithm to adaptively release orders based on the assessed state values of the production system. In contrast to the aforementioned works on average reward reinforcement learning our algorithm incorporates the optimisation of not only the average reward over time, but also the bias values, which is an important adaption in highly stochastic systems. Only the work by Mahadevan (1996a) integrates this second-level refinement optimisation. However, their algorithm requires the selection of a reference state to prevent an infinite increase of state values. This results from the lack of feedback in the iterative process of optimising the different refinement optimisation levels. Furthermore, they asses the average reward independently for each state.

In summary, we propose a novel average reward adjusted reinforcement learning algorithm to assess orders for their release to the shop floor. This is the first average reward adjusted reward learning algorithm that operates using a artificial neural network as function approximation. To ensure scalability we adapt a multi-agent approach, where each agent optimises the lead time management of a single product type. The agents do so by assessesing the expected costs for the possible releases imposed by adapting the lead time. According to the learned estimates each agent sets a planned lead time for the corresponding product type and with that releases orders into the production system. The highly optimised value-iteration algorithm uses improved n-step reinforcement learning with small action-based replay memories and worker agents to infer order release policies which outerperform [**MS:** todo] .

Structure. The rest of the paper is structured as follows. The next section introduced average reward reinforcement learning and presents the used algorithm

(see also Schneckenreither 2020). Section 3 describes the simulation model we use to evaluate the approach. [**MS:** todo]

2 Average Reward Adjusted Reinforcement Learning

This section briefly reintroduces the most important concepts of average reward adjusted reinforcement learning, elaborates on optimality criteria and provides insights of the underlying algorithm. For a more extensive introduction to average reward reinforcement learning we refer to Schneckenreither (2020).

Like Miller and Veinott (1969) we are considering problems that are observed in a sequence of points in time labeled $1, 2, \ldots$ and can be modelled using a finite set of states $\mathcal{S}$, labelled $1, 2, \ldots, |\mathcal{S}|$, where the size $|\mathcal{S}|$ is the number of elements in $\mathcal{S}$. At each point t in time the system is in a state $s_t \in \mathcal{S}$. Further, by choosing an action a_t of a finite set of possible actions A_s the system returns a reward $r_t = r(s_t, a_t)$ and transitions to another state $s_{t+1} \in \mathcal{S}$ at time $t+1$ with conditional probability $p(s_{t+1}, r_t \mid s_t, a_t)$. That is we assume that reaching state s_{t+1} from state s_t with reward r_t depends solely on the previous state s_t and chosen action a_t. In other words, we expect the system to possess the Markov property (Sutton et al., 1998, p.63). Reinforcement learning processes that possess the Markov property are referred to as Markov decision processes (MDPs) (Sutton et al., 1998, p.66).

Thus, the action space is defined as $F = \times_{s=1}^{|\mathcal{S}|} A_s$, where A_s is a finite set of possible actions. A *policy* is a sequence $\pi = (f_1, f_2, \ldots)$ of elements $f_t \in F$. Using the policy π means that if the system is in state s at time t the action $f_t(s)$, i.e.the s-th component of f_t, is chosen. A stationary policy $\pi = (f, f, \ldots)$ does not depend on time. In the sequel we are concerned with stationary policies only.

2.1 Discounted Reinforcement Learning

In the widely applied discounted framework the value of a state $V_\gamma^{\pi_\gamma}(s)$ is defined as the expected discounted sum of rewards under the stationary policy π_γ when starting in state s. Note that the policy π_γ depends on the selected discount factor. That is

$$V_\gamma^{\pi_\gamma}(s) = \lim_{N \to \infty} \mathbb{E}[\sum_{t=0}^{N-1} \gamma^t R_t^{\pi_\gamma}(s)] \,,$$

where $0 \leqslant \gamma < 1$ is the discount factor and $R_t^\pi(s) = \mathbb{E}_\pi[r(s_t, a_t) \mid s_t = s, a_t = a]$ the reward received at time t upon starting in state s by following policy π (Mahadevan, 1996b). The aim in the discounted framework is to find an optimal policy $\pi_\gamma^\star$, which when followed, maximises the state value for all states s as compared to any other policy π_γ: $V_\gamma^{\pi_\gamma^\star} - V_\gamma^{\pi_\gamma} \geqslant 0$. This criteria is usually referred to as γ-optimality as the discount factor γ is fixed. However, this also means that the actual value set for γ determines the best achievable

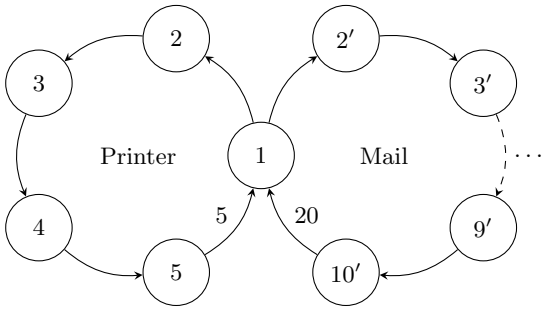

Fig. 1: A MDP with a single action choice in state 1, i.e. two different deterministic policies (Mahadevan, 1996c, adapted).

policy. For instance, as can be seen in Figure 1 setting $\gamma < 0.8027$ the printer-loop is preferred over the mail-loop, although the mail-loop accumulates more reward over time, i.e. selecting the mail-loop is a sub-optimal choice. Observe that the average reward received per step equals 1 for the printer-loop and 2 for the mail-loop. Thus, the (Blackwell-)optimal policy is to choose the mail-loop. However, if $\gamma < 3^{-\frac{1}{5}} \approx 0.8027$ an agent using discounted reinforcement learning chooses the printer loop as going the printer loop has a higher evaluation.

Therefore, in the seek of a more general optimality criteria for determining the best stationary policy π for a given problem, we introduce n-discount-optimality.

Definition 1 (n-Discount-Optimality). *Due to Veinott (1969) for a MDP a policy $\pi^\star$ is* n-discount-optimal *for $n = -1, 0, 1, \ldots$ for all states $s \in \mathcal{S}$ with discount factor γ if and only if*

$$\lim_{\gamma \to 1} (1-\gamma)^{-n} \ (V_\gamma^{\pi^\star}(s) - V_\gamma^{\pi}(s)) \geqslant 0 \ .$$

Only if a policy is n-discount-optimal for all $n < m$ it can be m-discount optimal (Puterman, 1994; Veinott, 1969). If a policy is -1-discount-optimal it is called gain-optimal, if it is 0-discount-optimal it is also called bias-optimal. Furthermore, if a policy is ∞-discount-optimal then it is said to be Blackwell-optimal (Blackwell, 1962). That is, for Blackwell-optimal policies $\pi^\star$ there exists a discount factor $\gamma^\star < 1$ such that $V_\gamma^{\pi^\star}(s) \geqslant V_\gamma^{\pi}(s)$ for all $\gamma \geqslant \gamma^\star$ and under all policies π (Blackwell, 1962; Mahadevan, 1996d). Informally that means there exists a discount factor $\gamma < 1$ which finds Blackwell-optimal policies. However, (i) in more complex, i.e. real world, MDPs this value can be arbitrary close to 1 and (ii) the difference in state values may be very small, which (iii) due to the need of state value function approximation likely causes errors when choosing among different actions in the discounted framework. Schneckenreither (2020) establishes a practically relevant definition for near-Blackwell-optimality.

Definition 2. *If an algorithm infers for any MDP bias-optimal policies, and for a given MDP can in theory be configured to infer Blackwell-optimal policies, but*

in practise this ability is naturally limited due to the finite accuracy of floating-point representation of modern computer systems, it is said to be near-Blackwell-optimal*. An according to a near-Blackwell-optimal algorithm inferred Blackwell-optimal policy is called near-Blackwell-optimal.*

Note that standard discounted RL does not meet the requirements of near-Blackwell-optimality, as it generally does not infer bias-optimal policies. Only by chance the resulting policy could be bias-optimal. The corresponding Bellman equation defined as $V_\gamma^\pi(s) = \mathbb{E}_\pi[r + \gamma V_\gamma^\pi(s')]$ (see e.g. Sutton et al., 1998, p.70) provides a way to assess the state values using two consecutively observed states s, s' and the observed reward r returned by the system. When this formula is used as an update rule it provides an algorithm that converges the state values by iteratively adapting the state value estimates. There are four major issues with discounted RL, which results in the fact that it is inapplicable for many problems of operations research.

1. In general standard discounted RL can only infer suboptimal policies as the discount factor is strictly less than one, i.e. $\gamma < 1$.
2. It is very difficult to specify a desired balance between the short-term and long-term (average) rewards. This results from the fact that the long-term rewards are scaled exponentially, where the factor depends on the chosen γ value.
3. Episodic MDPs with an average reward per step that is non-zero cannot be solved correctly, as the average reward is ignored in the terminal states.
4. The average reward is independently assessed for each state, even though it is the dominating part for all state values and thus is shared between more than one or, for unichain MDPs, even all states. This usually leads to an exponentially increased number of learning steps required for policies to converge.

Especially due to the fourth issue standard discounted RL is not well applicable in the operations research domain. This mainly results from the fact that a continuous assessing of the state values using costs or profit is often required, which leads to intractable long learning phases with high exploration rate even for small sized problems. But as high exploration rates produce errors in the state value estimations (Miller and Veinott, 1969) finding a well working parameterisation is difficult. Average reward reinforcement learning overcomes these issues by separately assessing the subterms of the state values directly.

2.2 Average Reward Reinforcement Learning

Due to Howard (1964) the average reward $\rho^\pi(s)$ of a policy π and a starting state s is defined as

$$\rho^\pi(s) = \lim_{N\to\infty} \frac{\mathbb{E}[\sum_{t=0}^{N-1} R_t^\pi(s)]}{N} \ .$$

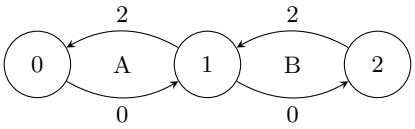

Fig. 2: A MDP with two gain-optimal deterministic policies, π_A going left in 1 is also bias-optimal, while π_B is not (Adapted from Mahadevan, 1996b).

In the common case of unichain MDPs, in which only a single set of recurrent states exists, the average reward $\rho^\pi(s)$ is equal for all states s (Mahadevan, 1996b; Puterman, 1994). In the sequel we focus on unichain MDPs in this work and thus may simply refer to it as ρ^π. A policy $\pi^\star$ that maximises the average reward $\rho^{\pi^\star}(s) - \rho^\pi(s) \geqslant 0$ in every state s as compared to any other policy π is called gain-optimal. Mahadevan (1996a) shows that n-discount-optimality for $n = -1$ describes gain-optimality.

Further, for an unichain aperiodic[3] MDP problem, such as ergodic MDPs are, the average adjusted sum of rewards or bias value is defined as

$$V^\pi(s) = \lim_{N\to\infty} \mathbb{E}[\sum_{t=0}^{N-1} (R_t^\pi(s) - \rho^\pi)] \,,$$

where again $R_t^\pi(s)$ is the reward received at time t, starting in state s and following policy π. Note that the bias values are bounded due to the subtraction of the average reward. Thus the bias value can be seen as the rewards that additionally sum up in case the process starts in state s. A policy $\pi^\star$ that is gain-optimal is also bias-optimal, if it maximises the bias values $V^{\pi^\star}(s) - V^\pi(s) \geqslant 0$ in every state s and compared to every other policy π. Bias-optimality is given by setting $n = 0$ in the definition of n-discount-optimality (Mahadevan, 1996a).

Especially for highly probabilistic systems bias-optimality is important. To clarify this consider Figure 2 which again consists of two possible deterministic policies with the only choice in state 1. Both policies have the same average reward of 1. However, only taking the A loop is bias-optimal, as its policy π_A leads to bias values $V^{\pi_A}(1) = 0.5$, $V^{\pi_A}(0) = -0.5$ and $V^{\pi_A}(2) = 1.5$, while policy π_B which selects the B loop generates bias values $V^{\pi_B}(1) = -0.5$, $V^{\pi_B}(0) = -1.5$ and $V^{\pi_B}(2) = 0.5$. This yields from the fact that the actions with non-zero rewards are selected earlier in policy π_A. Consider starting in state 1. Under the policy π_A the reward sequence is $(2, 0, 2, 0, \ldots)$, while for the other policy π_B it is $(0, 2, 0, 2, \ldots)$.

[3] In the periodic case the Cesaro limit of degree 1 is required to ensure stationary state transition probabilities and thus stationary bias values (Puterman, 1994). Therefore to ease readability we concentrate on unichain and aperiodic MDPs.

2.3 The Laurent Series Expansion of Discounted State Values

Miller and Veinott (1969) established the link between discounted RL state values $V_\gamma^\pi(s)$ and average reward RL values $\rho^\pi(s)$ and $V^\pi(s)$ using the Laurent series expansion as

$$V_\gamma^\pi(s) = \frac{\rho^\pi(s)}{1-\gamma} + V^\pi(s) + e_\gamma^\pi(s) \ ,$$

where the error term $e_\gamma^\pi(s)$ exists of infinitely many subterms and Puterman (1994) shows that $\lim_{\gamma \to 1} e_\gamma^\pi(s) = 0$. Note how the first term depending on the average reward $\rho^\pi(s)$ converges to infinity as γ increases.

However, an important insight is the connection between n-discount-optimality and the Laurent series expansion of $V_\gamma^\pi(s)$. Each addend corresponds to one step in n-discount-optimality. That is for (-1)-discount-optimality the addend describes (a scaled version of) the term to maximise for gain-optimality, for 0-discount-optimality the term to maximise for bias-optimality, and finally n-discount-optimality for $n \geqslant 1$ requires to maximise the error term $e_\gamma^\pi(s)$. The later only exists as γ is strictly less than 1. Thus this term incorporates the number of expected steps and their corresponding reward on the path to the Blackwell-optimal policy. Put differently, it optimises the expected reward collected to reach the Blackwell optimal policy according to the occurrence on the paths, where shorter paths and those which collect the rewards sooner are preferred.

These addends, where $n = -1, 0, \ldots$ denote the coefficients of the Laurent series expansion and thus correspond to the n of the definition of n-discount-optimality, can be reformulated to following constraint problem (Miller and Veinott, 1969; Puterman, 1994, p.346).

$$\rho^\pi(s) - \mathbb{E}[\rho^\pi(s)] = 0 \qquad \text{for } n = -1 \tag{1}$$

$$\rho^\pi(s) + V^\pi(s) - \mathbb{E}[V^\pi(s)] = R^\pi(s) \quad \text{for } n = 0 \tag{2}$$

$$W_{n-1}^\pi(s) + W_n^\pi(s) - \mathbb{E}[W_n^\pi(s)] = 0 \qquad \text{for } n \geqslant 1, \text{where } W_0^\pi(s) = V^\pi(s) \tag{3}$$

The error term $e_\gamma^\pi(s)$ is given by $e_\gamma^\pi(s) = \sum_{n=1}^\infty W_n^\pi(s)$ and the expected reward $R^\pi(s) = \mathbb{E}_\pi[r(s,a)]$. Puterman (1994, p.343ff) shows that due to the given degree of freedom if $n = -1, 0, \ldots, M$ constraints are satisfying the above conditions for all states s, then only $\rho^\pi(s), V^\pi(s), W_1^\pi, \ldots, W_{M-1}^\pi$ are unique, whereas W_M^π is offset by the vector u where for the transition probability matrix P the vector u is characterised by $(I - P)u = 0$. Note that u is determined by the number of closed irreducible classes of P, that is for ergodic MDPs u is determined by a single constant. Average reward learning is based on the above formulation.

A major problem occurring at average reward RL is that the bias values are not uniquely defined without solving the first set of constraints defined by the error term addends (see Mahadevan, 1996d; Puterman, 1994, p.346). Our average reward adjusted learning algorithm (ARAL), based on the tabular version of Schneckenreither (2020), does not require the exact solution for $V^\pi(s)$, but a solution which is offset suffices. Clearly this observation reduces the required

iteration steps tremendously as finding the exact solution, especially for large discount factors, is tedious. Therefore, we allow to set $\gamma = 1$, which induces $X_\gamma^\pi(s) = V^\pi(s) + u$, where u is for unichain MDPs a scalar value independent of s, i.e. equivalent for all states of the MDP (Puterman, 1994, p.346). If we are interested in correct bias values, i.e. γ is sufficiently close but strictly less than 1, our approach is a tremendous advantage over average reward RL as it reduces the number of iterative learning steps by requiring only a single constraint per state plus one for the scalar average reward value. That is, for an MDP with N states only one more constraint $(N+1)$ has to be solved in ARAL as compared to (at least) $2N+1$ nested constraints for average reward RL. Therefore, it is cheap to compute $X_\gamma^\pi(s)$, while it is rather expensive to find the correct values of $V^\pi(s)$ directly, especially in an iterative manner as RL is.

2.4 Average Reward Adjusted Reinforcement Learning

Thus, average reward adjusted RL can be seen as a specialised version of average reward RL, where it targets unichain MDPs only. In the sequel we briefly present the needed formalism of average reward adjusted RL. For more comprehensive version we refer to Schneckenreither (2020).

Definition 3. *The average reward adjusted (discounted) state value $X_\gamma^\pi(s)$ of a state s under policy π and with discount factor $0 \leqslant \gamma \leqslant 1$ is given as $X_\gamma^\pi(s) \coloneqq V^\pi(s) + e_\gamma^\pi(s)$.*

With the Laurent Series expansion this reformulates to $X_\gamma^\pi(s) = V_\gamma^\pi(s) - \frac{\rho^\pi}{1-\gamma}$. Therefore, $X_\gamma^\pi(s)$ describes the discounted state value adjusted by the average reward term. The Bellman Equation is given by $X_\gamma^\pi(s) = \mathbb{E}_\pi[r_t + \gamma X_\gamma^\pi(s_{t+1}) - \rho^\pi \mid s_t = s]$. The corresponding Bellman optimality equation then is given by $X_\gamma^{\pi^\star}(s) = \max_a \mathbb{E}_{\pi^\star}[r_t + \gamma X_\gamma^{\pi^\star}(s_{t+1}) - \rho^\pi \mid s_t = s]$. By turning this formula into an update rule it builds the foundation of our algorithm.

Average Reward Adjusted Deep RL Algorithm. The model-free average reward adjusted RL algorithm is based on the tabular version of Schneckenreither (2020) and depicted in Algorithm 1. The algorithm operates on a target network θ_X^T and a worker network θ_X^W as in Mnih et al. (2015). In the action selection process a ϵ-sensitive lexicographic order $a = (a_1, \ldots, a_n) \prec_\epsilon (b_1, \ldots, b_n) = b$ is used. It is defined as $a \prec_\epsilon b$ if and only if $|a_j - b_j| \leqslant \epsilon$ for all $j < i$ and $|a_i - b_i| > \epsilon$. Note that the resulting sets of actions may not be disjoint, but taking the maximum as required in our algorithm is straight-forward and thus cheap to compute. The first action selection criteria selects the actions which maximise the bias values, the second the error term. Equation 2 of the average reward RL constraints is used to estimate the average reward based on the currently inferred state values of two consecutive states and the returned reward. This infers a solid average reward prediction of the current policy in few steps only and thus quickly leads to better policies, as ones with higher gain are preferred. Schneckenreither (2020)

Algorithm 1 Basic near-Blackwell-optimal deep RL for unichain MDPs

Initialise state s_0 and network parameters θ_X^T, θ_X^W randomly, set an exploration rate $0 \leqslant p_{\mathsf{learn}} < p_{\mathsf{expl}} \leqslant 1$, exponential smoothing learning rates $0 < \alpha, \gamma < 1$, and discount factors $0 < \gamma_0 < \gamma_1 \leqslant 1$, where $\gamma_1 = 1$ is usually a good choice.
while the stopping criterion is not fulfilled **do**
    With probability p_{expl} choose a random action and probability $1 - p_{\mathsf{expl}}$ one that fulfills $\max_a \preccurlyeq_\epsilon (X^\pi_{\gamma_1}(s_t, a; \theta_X^W), X^\pi_{\gamma_0}(s_t, a; \theta_X^W))$. Let a_t^{rd} indicate if the action was chosen randomly.
    Carry out action a_t, observe reward r_t and resulting state s_{t+1}. Store the experience $(s_t, a_t, a_t^{\mathsf{rd}}, r_t, s_{t+1})$ in the experience replay memory M.
    Sample random mini-batch of m experiences from M
    Reset gradients: $d\theta_X^W \leftarrow 0$
    for each experience i with $(s_t, a_t^{\mathsf{rd}}, a_t, r_t, s_{t+1})$ **do**
        if a non-random action was chosen or $p_{\mathsf{expl}} > p_{\mathsf{learn}}$ **then**
            $\rho^\pi \leftarrow (1-\alpha)\rho^\pi + \alpha[r_t + \max_a X^\pi_{\gamma_1}(s_{t+1}, a; \theta_X^W) - X^\pi_{\gamma_1}(s_t, a_t; \theta_X^W)]$
        $y_{i,\gamma_0} \leftarrow r_t + \gamma_0 \max_a X^\pi_{\gamma_0}(s_{t+1}, a; \theta_X^T) - \rho^\pi$
        $y_{i,\gamma_1} \leftarrow r_t + \gamma_1 \max_a X^\pi_{\gamma_1}(s_{t+1}, a; \theta_X^T) - \rho^\pi$
        Sum gradients on $\left(y_{i,\cdot} - X^\pi(s_{t+1}, a; \theta_X^W)\right)^2$ wrt. parameters θ_X^W in $d\theta_X^W$
    Perform update with average of the gradients $d\theta_X^W / m$ on θ_X^W
    Every C steps exponential smoothly set target network: $\theta_X^T \leftarrow (1-\gamma)\theta_X^T + \gamma\theta_X^W$
    Set $s \leftarrow s'$, $t \leftarrow t+1$ and decay parameters

and Tadepalli and Ok (1998) find that updating the average reward by Equation 2 is superior as compared to exponentially smoothing the actual observed rewards. This makes sense as the later rather evaluates past policies.

The average reward adjusted state value, which is approximated based on the Bellman optimality equation, is estimated twice. Once to infer the bias values with a very high discount factor γ_1, e.g. $\gamma_1 = 1.0$, and a second time with a discount factor $\gamma_0 < \gamma_1$. The later is used to more selectively choose actions in case more than one action leads to bias-optimal policies. The predetermined discount value γ_0 is a measure for farsightedness. Lower values prefer to partially collect reward in fewer steps and higher values are used to prefer actions for which the full bias is collected earlier. The corresponding update values are saved to calculate the gradients based on the quadratic errors. Every C steps the target network smoothly adapts the parameters of the worker network.

3 Experimental Evaluation

This section introduces the simulation model and the parameterisation of the algorithm. To provide a proof-of-concept for the proposed algorithm we adapt the flow shop presented in Schneckenreither et al. (2020). However, our algorithms operates on discrete lead times on a product type basis and thus is not directly comparable to their order based lead time management approach. Nonetheless, this section is largely based on the corresponding section of Schneckenreither et al. (2020).

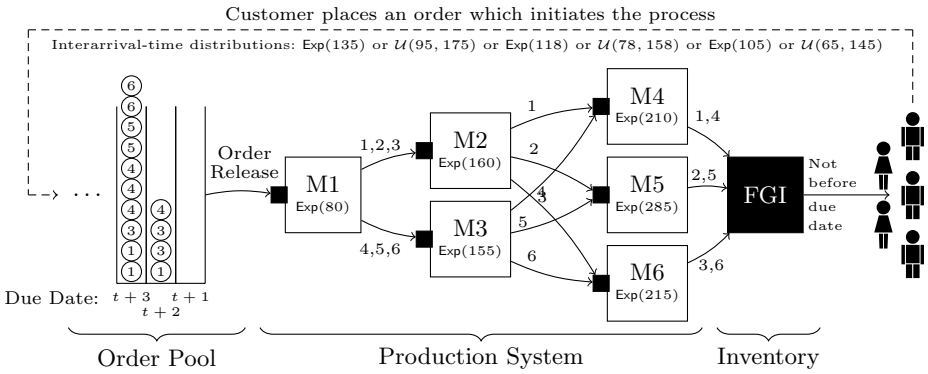

Fig. 3: Production System of the Simulation Model with routing, processing time distributions and demand interarrival time distributions.

3.1 Simulation Model

We adapt the simulation model of a make-to-order (MTO) flow shop with six different products and six work-centers of Schneckenreither et al. (2020) and examine the system under moderate and high utilisation in combination with moderate or high variability in demand and processing times. This section provides the details of the simulation model and the experimental setup.

The simulation model is depicted in Figure 3, where the customers initiate the process by placing orders as indicated by the dashed line. The manufacturer produces six different product types, each of which is identified by a natural number $1, 2, \ldots, 6$ and has a different routing through the production system. The routing setup is provided by corresponding labels of the edges. For instance, product types $1-3$ are routed from machine $M1$ to machine $M2$, while the other products are transferred to machine $M3$. The production system consists of three production stages with diverging material flow and no return visits. The incoming orders are uniformly distributed among the product types. A period lasts 960 minutes (= 16 hours) which represent two 8-hour shifts. At the beginning of each period orders can be released into the production system assuming material availability for all orders at their release date. Upon release, the orders are placed in the buffer at machine $M1$. Buffers and inventories are plotted as filled squares. Thus, each workstation is equipped with a buffer which queues the orders until processing is started. All buffers use a first-come-first-serve dispatching rule. Furthermore, each workstation processes only one order at a time and early deliveries are prohibited. Orders completed ahead of their due date remain in the finished goods inventory (FGI) until they are due and sent to the customer.

Processing Times. The machine processing time distributions are specified under the corresponding node labels of the machines. To simulate high variance the processing times for all machines are drawn from an exponential distribution.

There is one bottleneck machine ($M5$) and therefore we refer to product types 2 and 5, those routed through machine $M5$, as bottleneck products whereas products $1, 3, 4, 6$ are non-bottleneck products.

Demand. The incoming orders are placed in the order pool with a due date slack of 7 periods, that is each order arriving within period t will be due at the end of period $t + 7$. The due date slack of 7 was chosen to (i) ensure sufficient time between the first occurrence of orders in the order pool and the latest possible release of orders and (ii) to have a clear cause and effect relationship between the order release decision and the cost performance in the analysis. To simulate medium and high variability of the demand process the interarrival time between consecutive incoming orders is either drawn from an exponential (Exp) or an uniform ($\mathcal{U}$) distribution, which are adjusted to yield the desired bottleneck utilisation level of either 70% with $\mathsf{Exp}(135)$ and $\mathcal{U}(95, 175)$, 80% with $\mathsf{Exp}(118)$ and $\mathcal{U}(78, 158)$, or 90% with $\mathsf{Exp}(105)$ and $\mathcal{U}(65, 145)$. These values were chosen as they comprise the non-linear relationship between flow times and high utilisation levels.

Order Release. Before the order release decision is made, all orders in the order pool are sorted by due date. This portrays the implementation of a sequencing rule. According to this sequence, orders are considered for release in the beginning of each period starting with the highest priority order.

In our model, orders are released by specifying lead times $\mathsf{LT}_1, \mathsf{LT}_2, \ldots, \mathsf{LT}_6 \geqslant 1$, where the index corresponds to the product type and $\mathsf{LT}_i \in \mathbb{N}$. By setting a lead time LT_i a *planned release date* is computed for each order j of product type i in the order pool given by

$$\mathsf{PRD}_j = \mathsf{DD}_j - \mathsf{LT}_i \ , \tag{4}$$

where DD_j denotes the due date of the order. For dynamic order releases the lead times LT_i may vary from period to period, whereas in static order release methods the lead times are predetermined and fixed (Kim and Bobrowski 1995; Ragatz and Mabert 1988). For both approaches a job of product type i with lead time LT_i in period t and due date DD_j is released at the end of period t if and only if $t \geqslant \mathsf{PRD}_j$. Thus, in the dynamic setting the planned release date PRD_j of order j can be updated several times before it is actually released to the production system. However, once an order is released its release cannot be revoked. Hence, if the set lead time corresponds with the actual flow time the product is finished in the same period as it is shipped. If the lead times are set either too long or short, the order has to wait in the FGI until it is due or the order is late and backorder (BO) costs occur.

Costs. The cost parameters are set by assuming an increase in value from raw material (1 Dollar per order and period) to the final product (4 Dollar per order and period) and the backorder costs are set very high (16 Dollar per order and period) due to the MTO environment. All costs are assessed based on the production system state at the end of each period.

3.2 Conventional Order Release Rule

As external benchmark for comparison we use different parameterized backward infinite loading (BIL) techniques (Ackerman, 1963). BIL uses Equation 4 with a predetermined and fixed lead times LT_i.

3.3 Algorithm Setup

Markov Decision Process. The underlying MDP is unichain and looks as follows. Both, state space $\mathcal{S}$ and action space $\mathcal{A}$ are discrete. For the *state space* we adapt the result of Knollmann and Windt (2013), which state that the lead times ought to control the WIP level, instead of the due date reliability as changes are directly seen. Thus, any state $s \in \mathcal{S}$ of the state space is composed of the following information for each product type pt:

- The currently set lead time $LT_g \in \{1, 2, \ldots, \mathsf{dds}\}$ (Recall: Due Date Period − Lead Time = Release Period). Note that we bound the maximum lead time with due date slack dds, which is 7 in our setup.
- Counters $OP_{g,d} \in \mathbb{N}$ for the number of orders in the order pool divided in time buckets with $d \in \{1, 2, \ldots, \mathsf{dds}\}$, which stands for the number of periods until the due date.
- Counters $Q_{g,i,d} \in \mathbb{N}$ for the number of orders in the machine and queue $i \in \{M_1, M_2, \ldots, M_6\}$ divided in time buckets with $d \in \{1, 2, \ldots, \mathsf{dds}\}$, which stands for the number of periods until the due date.
- Counters $FGI_{g,d} \in \mathbb{N}$ of orders in the finished goods inventory divided in time buckets with $d \in \{1, 2, \ldots, \mathsf{dds}\}$, which stands for the number of periods until the due date. Orders with a due date with more than 3 periods ago are listed in the counter FGI_{-3}.
- Counters $S_{g,d} \in \mathbb{N}$ of shipped orders from the last period divided in time buckets with $d \in \{-3, -2, \ldots, \mathsf{dds}\}$, which stands for the number of periods until the due date. Orders with a due date with more than 3 periods ago are listed in the counter S_{-3}.

The algorithm implicitly learns a function which maps the current state of the production system to a lead time update and thus a release decision. In an optimal situation it does so by multiply exploring every action for each state and assessing its economic viability. Orders are released once the due date is within the interval $[t, t + \text{lead time}]$, whereas t is the current period. Clearly, this yields bulk releases, meaning that either all or no orders with the same due date and of the same product type are released.

The *action space* is composed of two decisions, the relative change of the lead times to the currently set lead times $LT_g \in \{1, 2, \ldots, \mathsf{dds}\}$ for each product type pt. We restrict the lead time update for state s_{t+1} according to the lead time LT_g from state s_t by a maximum change of 1. Thus, if LT_g is the current lead time of product type pt the agent can choose a lead time in $\{1, 2, \ldots, \mathsf{dds}\} \cap \{LT_g - 1, LT_g, LT_g + 1\}$. A naive approaches is to define one action for each combination of actions over all product type. This however, due to the exponential growth,

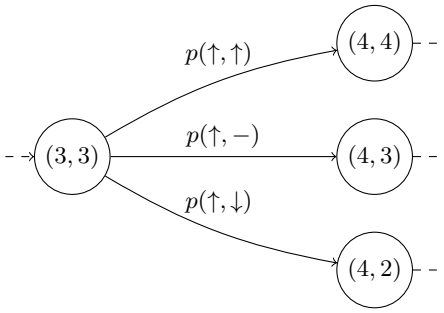

Fig. 4: Illustration of independent agents as agent one increases the lead time

results in only being able to solve very small production systems with only few product types. A common solution in production planning is to aggregate the products into product groups, cf. hierarchical production planning (Schneeweiβ, 1995). Although applicable in our setup, if we group the products into bottleneck and non-bottleneck products as done by Schneckenreither et al. (2020), also this approach simply does not scale.

Therefore, in the seek of a broader solution we establish the idea of *independent agents*. Each independent agent operates on the same system state, but chooses only a part of the actual executed action. In our case each independent agent is responsible for the lead time update of a specific product type. To the best of the author's knowledge this is the first work to reduce the action space in this way. Figure 4 illustrates the idea, where wlog. we use two agents and for simplicity omit the reward and truncate the state to the lead times. Assume that the first independent agent chooses to increase the lead time from 3 to 4. Thus, the future state, seen from this agent, depends on the probability of the other independent agent choosing to increase ($\uparrow$), decrease ($\downarrow$) or not change ($-$) the lead time. The Markov property, that is $p(s_{t+1}, r_t \mid s_t, a_t) = p((4, \cdot), r_t \mid (3, 3), (\uparrow, \cdot))$, is not touched by this adaption. While for the first agent the reward and future state only depends on the current state and action taken, the corresponding probabilities for future state and reward change during the learning process as the the agent starts to behave differently. This allows us to investigate the algorithms behaviour for even larger solution spaces. In particular, this adaption enables us to present the algorithm the above defined state space not only on the basis of product groups, but directly for each product type pt. Note that without this extension the problem is computationally infeasible due to $3^6 = 729$ possible actions. Removing the dependency between the lead times of different product types reduces the set of possible actions, and thus the number of ANN output layer nodes, to $3 \cdot 6 = 18$. In preliminary experiments on small examples we only observed a very slight increase of required steps during training. Therefore, this is a vital adaption to ensure scalability.

Reward. In each period the agents choose actions, which when executed generate a reward while traversing to the next period by simulating the production system.

The rewards are the accumulated costs at the end of the period consisting of the backorders, the WIP orders and the number of orders in the FGI. The objective is to minimise the returned reward.

Optimised Algorithm. Our implementation includes several optimisations, that help the algorithm in better exploring the solution space and stabilise the learning process. These include the above discussed independent agents that operate on the same state, n-step learning (Mnih et al., 2016), and ensuring to constantly reduce the average reward as searching for policies with higher average costs is unnecessary. Algorithm 2 depicts a detailed view on the algorithm including these extensions and with the objective to minimise the returned rewards. The idea is the same as in Algorithm 1, except that the action selection procedure is a loop over all independent agents and the implementation of the n-step logic. The algorithm is based around The agent first choose an action (steps $4-5$), executes the selected action and stores the experience in the replay memory (step 6) and then samples a mini-batch of experiences each of length *nstep* to improve the policy (steps $7-21$). Step 17 uses an exponentially smoothed overestimation (by 2.5%) of the average reward and rate 2^{-5}. All average reward updates above this overestimation are rejected.

As in (Mnih et al., 2015, 2016) the algorithm uses a *Replay Memory* to save experiences and then randomly selects experiences from there for learning. This ensures that the independent and identically distributed (i.i.d.) assumption of the samples for the ANN backpropagation algorithms are satisfied. We extend this idea by splitting the available size of the replay memory into subsets, where each subset represents one action and sample uniformly over these subsets first. This prevents overfitting when one action is overrepresented in the policy, e.g. no lead time change occurs much more often than the other actions.

N-step learning collects a set of consecutive experiences, in our case retrieved from the replay memory, and connects the state values starting from the latest observed experience, such that the newly computed state value of the corresponding experience propagates to the experience of the previous period. This allows a faster convergence, as values bootstrap over multiple periods in contrast to just one step. The experiences are stored according to the first taken action in the n-step series. We uniformly sample over the actions to ensure each action is constantly trained. This adaption reduces catastrophic forgetting of already accumulated knowledge. Furthermore, we use 8 additional worker threads (parallel RL) that operate on the same environment to collect a broader range of experiences (as e.g. in Mnih et al., 2016). The average reward value is shared and computed as average over the main agent and all workers.

The algorithm initiation strategy is to first fill the replay memory buffer by skipping any learning (steps $7-21$ in Algorithm 2) until the period, that corresponds to the replay memory size, is reached. Then for 500 steps the average reward is set to the last computed exponentially smoothed value to prevent divergence due to the untrained network weight initialisation values.

Algorithm 2 Optimised near-Blackwell-optimal deep RL for unichain MDPs

Initialise state s_0 and network parameters θ_X^T, θ_X^W randomly, set an exploration rate $0 \leqslant p_{\mathsf{learn}} \leqslant p_{\mathsf{expl}} \leqslant 1$, exponential smoothing learning rates $0 < \alpha_{\max}, \alpha, \gamma < 1$, a sufficiently large default value for $\rho^\pi_{\max}$ and discount factors $0 < \gamma_0 < \gamma_1 \leqslant 1$
while the stopping criterion is not fulfilled **do**
  $a_t^{\mathsf{rd}} \leftarrow \mathsf{False}$
  for each independent agent $i \in \mathsf{ag}_1, \mathsf{ag}_2, \ldots, \mathsf{ag}_n$ **do**
    With probability $\frac{p_{\mathsf{expl}}}{n}$ choose action $a_{t,i}$ randomly or otherwise one of the set defined by $\min_{a \in \mathcal{A}(i)} \preccurlyeq_\epsilon (X^\pi_{\gamma_1}(s_t, a; \theta_X^W), X^\pi_{\gamma_0}(s_t, a; \theta_X^W))$. Set $a_t^{\mathsf{rd}} \leftarrow \mathsf{True}$ if the action was chosen randomly.
  Carry out action $a_t = (a_{t,\mathsf{ag}_1}, \ldots, a_{t,\mathsf{ag}_n})$, observe reward r_t and resulting state s_{t+1}. Store experience $(s_t, a_t, a_t^{\mathsf{rd}}, r_t, s_{t+1})$ in the experience replay memory M.
  if $t \mod n_{\mathsf{step}} \equiv 0$ **then**
    Reset gradient: $d\theta_X^W \leftarrow 0$
    Sample m random mini-batches of n_{step} consecutive experiences from M numbered by $1 \leqslant i \leqslant n_{\mathsf{step}}$, each with $(s_i, a_i^{\mathsf{rd}}, a_i, r_i, s_{i+1})$.
    for each mini-batch **do**
      for each experience, starting with the latest seen experience n_{step} **do**
        if experience n_{step} or $\neg a_i^{\mathsf{rd}}$ or $p_{\mathsf{expl}} > p_{\mathsf{learn}}$ **then**
          $\mathsf{target}_{\gamma 0} \leftarrow \min_a X^\pi_{\gamma_0}(s_{i+1}, a; \theta_X^T)$
          $\mathsf{target}_{\gamma 1} \leftarrow \min_a X^\pi_{\gamma_1}(s_{i+1}, a; \theta_X^T)$
        if $\neg a_i^{\mathsf{rd}}$ or $p_{\mathsf{expl}} > p_{\mathsf{learn}}$ **then**
          $\rho^\pi \leftarrow (1-\alpha)\rho^\pi + \alpha[r_i + \min_a X^\pi_{\gamma_1}(s_{i+1}, a; \theta_X^W) - X^\pi_{\gamma_1}(s_i, a_i; \theta_X^W)]$
          Ensure improvement of ρ^π, otherwise reset to old value
        $\mathsf{target}_{\gamma_0} \leftarrow r_i + \gamma_0 \mathsf{target}_{\gamma_0} - \rho^\pi$
        $\mathsf{target}_{\gamma_1} \leftarrow r_i + \gamma_1 \mathsf{target}_{\gamma_1} - \rho^\pi$
        $d\theta_X^W \leftarrow d\theta_X^W + \partial(\mathsf{target}_{\gamma_0} - X^\pi_{\gamma_0}(s_{i+1}, a; \theta_X^W))^2 / \partial \theta_X^W$
        $d\theta_X^W \leftarrow d\theta_X^W + \partial(\mathsf{target}_{\gamma_1} - X^\pi_{\gamma_1}(s_{i+1}, a; \theta_X^W))^2 / \partial \theta_X^W$
    Perform update of θ_X^W using $d\theta_X^W / (m \cdot n_{\mathsf{step}})$
  Every C steps exponential smoothly set target network: $\theta_X^T \leftarrow (1-\gamma)\theta_X^T + \gamma \theta_X^W$
  Set $s \leftarrow s'$, $t \leftarrow t+1$ and decay parameters

Neural Network Setup. Due to the vast possibilities the neural network architecture was manually optimised, starting with less stochastic systems and aiming for a coffin shape that can be used in all setups. During this process we constantly adapted and reevaluated the best network on various production system configurations. The resulting architecture depends on the number of features $\mathsf{n}_{\mathsf{inp}}$ and the number of output nodes $\mathsf{n}_{\mathsf{out}}$, which itself depends on the number of actions $\mathsf{n}_{\mathsf{act}} = |\mathcal{S}|$, the independent agent configuration, and the number of functions that are approximated. For the later, in case of ARAL this is 2, as we represent both functions $X^\pi_{\gamma_0}$ and $X^\pi_{\gamma_1}$ with the same neural network, while deep Q-Learning (cf. Mnih et al., 2015) only approximates the function Q^π_γ. The resulting feedforward deep ANN architecture is given as $\mathsf{n}_{\mathsf{inp}} \to 1.75\mathsf{n}_{\mathsf{inp}} \to 1.5\mathsf{n}_{\mathsf{inp}} \to \mathsf{n}_{\mathsf{out}}$, where each arrow symbolises a fully-connected layer with a leaky rectified linear unit (ReLU) as activation function, which leaks values smaller than 0 with rate 0.02. Note

that we explicitly do not use an output activation function. The idea is that we do not want to restrict the approximations to values inside this range. This is especially important for Q-Learning, as the range of the state values are difficult to forecast. The actual values $\mathsf{n}_{\mathsf{inp}}$ and $\mathsf{n}_{\mathsf{out}}$ depend on the used algorithm. In our case using 6 independent agents we gain $\mathsf{n}_{\mathsf{inp}} = 324$ and $\mathsf{n}_{\mathsf{out}} = 36$.

The features representing order counts are scaled using min-max scaling with minimum 0 and maximum 5, the lead times with minimum 0 and maximum 7 respectively. To further ensure smaller ANN parameter values the output is transformed using min-max scaling to the interval $(-400, 800)$, s.t. an ANN output of 1 represents a value of 800. On each training step, i.e. every n-step steps, randomly m sets of experiences the training mini-batch size each consisting of n-step experiences are selected, there gradients computed and the sum of gradients backpropagated though the worker network using Adam (Kingma and Ba, 2014), where the weight decay with parameter λ is implemented as presented by Loshchilov and Hutter (2017). Every C steps the target network is updated by exponentially smoothing the parameters with rate γ. As neural network initialisation method we use the commonly known Xavier initialisation (Glorot and Bengio, 2010), i.e. the initial value for each weight i is sampled from a uniform distribution $W_{l,i} \sim \mathcal{U}(-1/\sqrt{n_l}, 1/\sqrt{n_l})$, where n_l is the number of nodes in layer l.

3.4 Deep Q-Learning Algorithm

Furthermore, we adapt the deep Q-Learning algorithm of Mnih et al. (2015) with integrated extensions as described above, i.e. the adaptions to n-step learning, multiple workers, independent agents, scaling of the output, smoothly adapting the target network. The resulting Algorithm 3 is provided in the Appendix.

To ensure comparability we use the same neural network setup as for ARAL, except that we halve the number of output nodes, as the Q-Learning algorithm only approximates one state-action value function. Therefore the output nodes $\mathsf{n}_{\mathsf{out}}$ are 18, while the input nodes $\mathsf{n}_{\mathsf{inp}} = 324$ stay the same.

4 Computational Experiments

This section presents the experimental results and gives an overview of the performance of the established algorithm compared to the conventional static lead time setting algorithms.

Table 1 presents an overview of the parameter setup. The parameters α, γ and p_{exp} are exponentially decayed as indicated. Due to the vast possibilities we manually optimised the values. The methodology was to investigate the values on very small production system with just one machine and product due to the computational complexity of bigger sized problems. The resulting values were copied to the problem at hand and further manually optimised in pre-experimental runs. The actual used discount factors γ_1 differ for each method and thus are further specified in the result tables. In a similar approach we

Parameter	α	γ_0	γ_1	γ	ϵ	p_{exp}	p_{learn}
Start value	0.01	0.8	0.99/1.0	0.01	0.25	1.0	0.5
Exp. Smth. Rate	0.25			0.25		0.25	
Exp. Smth. Steps in 10^3	50			100		100	
Minimum value	5^{-5}			0.001		0.005	

Configuration	Value
Workers Min. Exploration	$0.01, 0.02, 0.03, 0.04, 0.05, 0.10, 0.15, 0.20$
Target Network Update	Exp. smooth. with rate γ every $C = 100$ steps
N-Step	5
Training Mini-Batch Size	4
Adam ANN Backpropagation	$\alpha = 0.005, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda = 0.001$
Replay Memory Size	10800

Table 1: Overview of Parameter Setup.

reached the configuration values, where a further increase of the values was hindered by the available computational resources.

For the experimental runs we used personal computers with Intel CPUs i7-7700 (16GB RAM) and i9-7900X (32GB RAM). For each method we performed 500.000 learning steps, which took about 2-3 days. For the same experimental setup the inputs/random variables are the same for each period. After learning each order release method was evaluated on 20 demand streams, each with 1000 periods of warm-up and 7000 periods of evaluation phase. Welch's procedure was applied to approximate the length of the warm-up period (see Law and Kelton (2000)).

Medium Demand Variability. The results for medium variability in the system are presented in Table 2, where a uniform demand interarrival time is used. We use the tuples 70-$\mathcal{U}$, 80-$\mathcal{U}$, and 90-$\mathcal{U}$ to indicate the experimental setup of using uniform demand interarrival times with a aimed bottleneck utilisation of 70%, 80%, or 90% respectively. The results state the accumulated costs, averaged over all replications, for the work in process (WIPC), the finished goods inventory (FGIC), the backorder costs (BOC), as well as the sum of the finished good inventory and backorder costs (FGI+BO) and the sum over all three cost centers (SUM). To ease readability all costs are given in 1000 Dollars. For a better insight in the timing performance we also provide the average shopfloor throughput time (SFTT), i.e. the average time an order spends in the system after being released and until reaching the finished goods inventory, the finished goods inventory time (FGIT), as well as the service level (SL). The later is given in percentage and states the percentage of orders that were produced on time. Finally, the actual measured costs per period (Cost.p.P) and the by the algorithm learned average reward ρ are stated. The tested methods are the deep Q-Learning algorithm with discount factor 0.99 ($\mathrm{QL}_{0.99}$), the ARAL algorithm with discount factors 0.99 ($\mathsf{ARAL}_{0.99}$) and 1.00 ($\mathsf{ARAL}_{1.00}$), as well as the backward infinite loading

70-$\mathcal{U}$	WIPC	FGIC	BOC	FGI+BO	SUM	SFTT	FGIT	SL(%)	Cost.p.P	ρ^π
$\mathsf{QL}_{0.99}$	38.56	134.70	81.70	216.41	254.97 [1]	1.34	1.22	90.6	36.42	
$\mathsf{ARAL}_{0.99}$	37.20	113.10	100.40	213.50	250.70	1.31	1.11	88.9	35.81	35.04
$\mathsf{ARAL}_{1.00}$	38.66	156.66	59.74	216.40	255.06 [1]	1.34	1.34	93.2	36.44	36.50
BIL1	32.06	0.00	512.90	512.90	544.96	1.20	0.49	44.9	77.85	
BIL2	32.06	83.46	105.01	188.47	**220.53**	1.20	0.94	88.6	**31.50**	
BIL3	32.16	247.73	21.68	269.41	301.56	1.20	1.83	97.8	43.08	
BIL4	32.14	429.10	5.08	434.17	466.32	1.20	2.81	99.5	66.62	

80-$\mathcal{U}$	WIPC	FGIC	BOC	FGI+BO	SUM	SFTT	FGIT	SL(%)	Cost.p.P	ρ^π
$\mathsf{QL}_{0.99}$	63.10	180.40	121.53	301.93	**365.03**	1.68	1.34	89.7	**52.15**	
$\mathsf{ARAL}_{0.99}$	65.31	168.49	148.59	317.08	382.39	1.72	1.28	86.4	54.63	52.92
$\mathsf{ARAL}_{1.00}$	60.78	234.05	76.17	310.21	370.99 [1]	1.64	1.59	93.4	53.00	52.29
BIL1	51.89	0.00	830.22	830.22	882.11	1.47	0.49	32.8	126.02	
BIL2	52.00	70.11	255.17	325.28	377.27	1.48	0.82	79.4	53.90	
BIL3	51.98	240.18	77.82	318.00	369.97 [1]	1.48	1.62	94.0	52.85	
BIL4	51.83	442.31	26.71	469.02	520.85	1.47	2.56	98.1	74.41	

90-$\mathcal{U}$	WIPC	FGIC	BOC	FGI+BO	SUM	SFTT	FGIT	SL(%)	Cost.p.P	ρ^π
$\mathsf{QL}_{0.99}$	98.91	190.68	297.33	488.01	586.92 [1]	2.13	1.28	82.7	83.85	
$\mathsf{ARAL}_{0.99}$	105.35	263.95	207.18	471.13	**576.48** [1]	2.24	1.59	87.9	**82.35**	79.83
$\mathsf{ARAL}_{1.00}$	107.87	240.80	253.87	494.66	602.54 [2]	2.28	1.49	84.7	86.08	80.43
BIL1	88.66	0.00	1,418.53	1,418.53	1,507.19	1.96	0.50	21.0	215.31	
BIL2	88.32	50.85	646.00	696.86	785.17	1.96	0.71	64.5	112.17	
BIL3	88.25	207.21	299.89	507.10	595.35 [2]	1.96	1.35	85.2	85.05	
BIL4	87.77	415.18	153.61	568.79	656.57	1.95	2.21	92.9	93.80	

Table 2: Results with moderate variability in the system due to uniformly distributed demand interarrival times. [1,2] Costs marked with the same number are not significantly different from each other.

technique BIL with fixed lead times between 1 and 4 as indicated by the number (BIL1-BIL4).

For the case of medium variability with a utilisation of 70% 70-($\mathcal{U}$) we can see that BIL2 performs best in the sense of accumulated costs (220.53). The reinforcement learning algorithm follow, where $\mathsf{ARAL}_{0.99}$ is slightly but significantly better then the other two variants (250.70, 254.97, 255.06). The other BIL variants accumulate much more costs. For the timing one can see that $\mathsf{ARAL}_{1.00}$ performs very good with a service level of 93.2%. Furthermore, the learned average reward ρ and actual average reward are very similar for both ARAL variants. However, with 70% of utilisation the overhead of the reinforcement learning variants are too big to be able to compete with the simple rule based release mechanism of BIL2 as for changing workload the utilisation changes rather linearly due to the low variability in the system.

In the case of 80-$\mathcal{U}$ the deep Q-Learning variant $\text{QL}_{0.99}$ performs best in the cost related measures (365.03 for SUM), followed by BIL3 (369.97) and $\text{ARAL}_{1.00}$ (370.99), which are not significantly different from each other. One can see the BIL2 (377.27) performs slightly worse than $\text{ARAL}_{1.00}$ and is followed by $\text{ARAL}_{0.99}$ (382.39). Again $\text{ARAL}_{1.00}$ achieves a very high service level of 93.4% with reasonable sum of cost, which is also reflected in the low Backorder costs. Furthermore, the measured and learned average reward values are quite close. Thus, the results already provide the insight that ARAL performs quite well in terms of timing for the completion of orders. This pays off when confronted with higher probabilistic systems.

For instance, with 90% utilisation and medium variable demand stream, i.e. 90-($\mathcal{U}$), we can see that $\text{ARAL}_{0.99}$ performs best (576.48), followed by $\text{QL}_{0.99}$ (586.92) and BIL3 (595.35), as well as $\text{ARAL}_{1.00}$ (602.54), where the later two are not significantly different. In this case $\text{ARAL}_{0.99}$ also achieves the best service level of 87.9% within this group of best performing algorithms. The WIPC costs, as well as the SFTT, show that the reinforcement learning agents have a higher work in process, which is clear as they try to smooth the workload of demand stream. When looking at the timing cost it becomes even more apparent that the agents perform well in timing as $\text{ARAL}_{0.99}$ performs much better (471.13) than the other two agents (488.01, 494.66) and all the BIL techniques accumulated more FGI+BO costs (507.10 for the best performing BIL, BIL3).

High Demand Variability. For results of the experiments using exponential demand interarrival times, presented in Table 3 we can see a similar pattern as before. In this case we use the tuples 70-Exp, 80-Exp, and 90-Exp to indicate the corresponding experimental setups.

For 70% utilization of the bottleneck, i.e. 80-Exp, BIL3 performs best with costs of 345.80, followed by $\text{ARAL}_{0.99}$ with 358.41, and then by $\text{QL}_{0.99}$ and BIL2 (360.70, 364.18). In terms of timing performance, again ARAL outperforms the opponents with FGI+BO costs of 296.99, and BIL3 with the second highest costs (297.17). However, the service level of BIL3 is higher with 93.7% due to the lower shop floor throughput time (SFTT, 1.48 periods for BIL3 versus 1.73 periods for $\text{ARAL}_{0.99}$).

In case of 80-Exp BIL3 performs best with costs of 521.40, but is not significantly better than $\text{ARAL}_{1.00}$ with 528.54. Also $\text{ARAL}_{0.99}$ performs quite well (529.00). The service level of all three algorithm are good, but the order is reversed: $\text{ARAL}_{0.99}$ with 87.6%, $\text{ARAL}_{1.00}$ with 86.6% and BIL3 with 85.8%. Again the SFTT of the RL agents are slightly higher, i.e. 0.13-0.20 periods, than the ones of the BIL release procedures. The other three BIL methods, BIL1, BIL2 and BIL4, are performing rather bad (1,338.44, 679.24 and 591.82 respectively). Interesting to see is that in this scenario $\text{ARAL}_{0.99}$ and $\text{ARAL}_{1.00}$ handle the due date performance better than $\text{QL}_{0.99}$, as can be seen in the backorder costs (BOC). Here the former two accumulate costs of 185.69 and 198.46, while $\text{QL}_{0.99}$ faces costs of 225.15. This results in timing costs (FGI+BO) of 439.16 for $\text{ARAL}_{0.99}$ and 441.37 for $\text{ARAL}_{1.00}$, which also means a better timing cost than BIL3 with 442.37. For this scenario the actual measured average reward (average costs) of

70-Exp	WIPC	FGIC	BOC	FGI+BO	SUM	SFTT	FGIT	SL(%)	Cost.p.P	ρ^π
$\mathsf{QL}_{0.99}$	53.87	142.40	164.43	306.83	360.70 [1]	1.59	1.21	83.8	51.53	
$\mathsf{ARAL}_{0.99}$	61.42	181.70	115.29	296.99	358.41	1.73	1.41	89.3	51.20	50.79
$\mathsf{ARAL}_{1.00}$	56.58	199.31	115.71	315.02	371.60	1.64	1.50	88.7	53.09	50.58
BIL1	48.55	0.00	776.75	776.75	825.30	1.48	0.50	33.7	117.90	
BIL2	48.62	66.93	248.64	315.57	364.18 [1]	1.48	0.83	78.0	52.03	
BIL3	48.63	222.60	74.57	297.17	**345.80**	1.48	1.61	93.7	**49.40**	
BIL4	48.51	409.53	23.41	432.94	481.45	1.48	2.55	98.1	68.78	

80-Exp	WIPC	FGIC	BOC	FGI+BO	SUM	SFTT	FGIT	SL(%)	Cost.p.P	ρ^π
$\mathsf{QL}_{0.99}$	87.71	227.49	225.15	452.64	540.34 [1]	2.04	1.50	83.4	77.19	
$\mathsf{ARAL}_{0.99}$	89.84	253.48	185.69	439.16	529.00 [1,2]	2.08	1.61	87.6	75.57	70.53
$\mathsf{ARAL}_{1.00}$	87.17	242.92	198.46	441.37	528.54 [2,3]	2.03	1.57	86.6	75.51	72.94
BIL1	78.73	0.00	1,259.71	1,259.71	1,338.44	1.88	0.50	22.9	191.21	
BIL2	78.20	52.30	548.74	601.04	679.24	1.88	0.73	65.0	97.03	
BIL3	79.03	200.27	242.10	442.37	**521.40** [3]	1.89	1.38	85.8	**74.49**	
BIL4	79.38	395.49	116.94	512.44	591.82	1.90	2.23	93.8	84.55	

90-Exp	WIPC	FGIC	BOC	FGI+BO	SUM	SFTT	FGIT	SL(%)	Cost.p.P	ρ^π
$\mathsf{QL}_{0.99}$	168.66	222.15	788.66	1,010.81	1,179.46 [1]	3.13	1.37	71.5	168.49	
$\mathsf{ARAL}_{0.99}$	158.03	212.73	701.30	914.03	1,072.06 [2]	2.97	1.33	71.9	153.15	115.80
$\mathsf{ARAL}_{1.00}$	160.45	286.07	539.80	825.87	**986.32**	3.01	1.62	79.6	**140.90**	109.91
BIL1	146.84	0.00	2,349.37	2,349.37	2,496.21	2.80	0.50	13.3	356.60	
BIL2	143.76	34.11	1,412.75	1,446.86	1,590.62	2.75	0.63	47.1	227.23	
BIL3	144.28	154.15	877.45	1,031.60	1,175.88 [1]	2.76	1.10	70.6	167.98	
BIL4	147.08	335.07	622.14	957.21	1,104.29 [2]	2.80	1.81	82.5	157.76	

Table 3: Results with high variability in the system due to exponentially distributed interarrival times of the demand. [1,2,3] Costs with the same number are not significant different from each other.

about 75.5 for both ARAL variants and learned average rewards of 70.53 for $\mathsf{ARAL}_{0.99}$ and 72.94 for $\mathsf{ARAL}_{1.00}$ are a little bit offset. This probably results from the higher variability in the system.

Finally, for the case of 90-Exp $\mathsf{ARAL}_{1.00}$ performs much better than the other variants with cots of 986.32. It is followed by $\mathsf{ARAL}_{0.99}$ with costs of $1,072.06$ and rather far, but not significantly, afterwards follows BIL4 $(1,104.29)$. As before the shopfloor throughput time (SFTT) is higher for the RL variants, and with that of course also the work in process costs (WIPC). The backorder costs of $\mathsf{ARAL}_{1.00}$ (539.80) are the lowest of all tested methods. Like for the overall costs also in terms of timing costs (FGI+BO) $\mathsf{ARAL}_{1.00}$ performs best (825.87), followed by $\mathsf{ARAL}_{0.99}$ (914.03) and BIL4 (957.21). BIL3 and $\mathsf{QL}_{0.99}$ perform quite similar in the sense of accumulated costs, $1,175.88$ and $1,179.46$ respectively. But the other measures show that $\mathsf{QL}_{0.99}$ has a better timing performance, thus also

a slightly high service level with 71.5% as compared to BIL3 with 70.6%. The best service level is achieved by BIL4 with 82.5%, before $\mathsf{ARAL}_{1.00}$ 79.6%, but with a high cost as the difference of SUM is 117.97. Interesting to see is that the measured average reward and the learned average reward are very different for both ARAL variants (about 30-40). This is probably due to the very high variability in the system, but might also be an indicator that the agents need more time for learning. However, due to the high computational complexity we encountered we leave this to future work.

Overall we can see that the RL agents, and especially the established ARAL algorithm performs very well in systems with high variability. In case of the less variable systems the results reveal that there is still potential for the improvement of reinforcement learning variants.

5 Conclusion

This paper is one of the first papers that use reinforcement learning for setting lead times to release orders into the production system. We present a novel average reward adjusted reinforcement learning algorithm that uses an artificial neural network to approximate the state-action value functions. We add several extensions to this algorithm, including n-step learning, and a special experience replay memory that stores its experiences according to the actions takes to prevent catastrophic forgetting. Additionally we invent *independent agents* to reduce the action space by independently choosing a subset of possible actions, while operating on the same state. Additionally we adapt the deep Q-Learning algorithm of Mnih et al. (2015) to be able to present a fair comparison between the algorithms, where we also use several backward infinite loading techniques.

The results suggest that for high variability the RL learning agents, and here especially the novel ARAL variant, performs very well. In case of medium variability the achieved results are as good as with the BIL variants. And for low variability one can see that the RL agents still have room for improvement, as BIL outperforms them. When comparing deep Q-Learning to the established ARAL algorithm, we can see that for all cases, except one (80-$\mathcal{U}$), ARAL performs better.

Therefore, in the future we plan to develop ARAL further to a actor-critic state-of-the-art reinforcement learning algorithm. We hypothesise that due to the more correct state value estimations of ARAL as compared to deep Q-Learning, cf. Schneckenreither (2020), the critic will provide a better feedback of the value function update to the actor. This should lead to better policies than with standard actor-critic algorithms, e.g. A3C **??**. Furthermore, one limitation of this proof-of-concept study is that due to the computational complexity we restricted our scope to flow-shop production systems only. In subsequent studies we plan to test the algorithms on job-shop queuing systems as well. Additionally, we want to exploit hierarchical production planning (Schneeweiβ, 1995) by implementing agents on different hierarchical levels which can bargain with each other. For instance, one agent might be responsible for medium term planning order re-

lease decision at the top level, while another agent concentrates on short-term scheduling at the base level. At the start of the process a proposed order release plan is generated by the first agent. Accordingly the agent from the base level performs a preliminary scheduling of these orders and may report very expensive orders as feedback. This information is then taken into consideration on the top level which adapts the order release plan accordingly. Here the top level agent minimises the overall costs by load leveling on an aggregated model, while at the bottom level the agent (mainly) minimises costs backorder costs. Finally, although the introduced actions space reduction method, named *independent agents*, works very well, it remains to be shown that the achievable policies are different to the ones for agents without this adaption.

[**MS:** todo]

Bibliography

S.S. Ackerman. Even-flow a scheduling method for reducing lateness in job shops. *Management Technology*, 3:20–32, 1963.

M Emin Aydin and Ercan Öztemel. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2-3):169–178, 2000.

Marc G Bellemare, Joel Veness, and Michael Bowling. Investigating contingency awareness using atari 2600 games. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300, 1995.

J. W. M. Bertrand. The effect of workload dependent due-dates on job shop performance. *Management Science*, 29(7):799–816, 1983.

Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

David Blackwell. Discrete dynamic programming. *The Annals of Mathematical Statistics*, 344:719–726, 1962. doi: 016/j.cam.2018.05.030.

S.-H. Chung and H.-W. Huang. Cycle time estimation for wafer fab with engineering lots. *IIE Transactions*, 34(2):105–118, 2002. doi: 10.1080/07408170208928854. URL `https://doi.org/10.1080/07408170208928854`.

Tapas K. Das, Abhijit Gosavi, Sridhar Mahadevan, and Nicholas Marchalleck. Solving semi-markov decision problems using average reward reinforcement learning. *Management Science*, 45(4):560–574, 1999.

Stefanos Doltsinis, Pedro Ferreira, and Niels Lohse. Reinforcement learning for production ramp-up: A q-batch learning approach. In *2012 11th International Conference on Machine Learning and Applications*, volume 1, pages 610–615. IEEE, 2012.

S. T. Enns and P. Suwanruji. Work load responsive adjustment of planned lead times. *Journal of Manufacturing Technology Management*, 15(1):90–100, 2004.

L. F. Gelders and L. N. Van Wassenhove. Hierarchical integration in production planning: Theory and practice. *Journal of Operations Management*, 3(1):27–35, 1982.

Joren Gijsbrechts, Robert N Boute, Jan A Van Mieghem, and Dennis Zhang. Can deep reinforcement learning improve inventory management? performance and implementation of dual sourcing-mode problems. *Performance and Implementation of Dual Sourcing-Mode Problems (December 17, 2018)*, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Ronald A Howard. Dynamic programming and markov processes. 1964.

J. Hoyt. Dynamic lead times that fit today's dynamic planning (quoat lead times). *Production and Inventory Management*, 19(1):63–71, 1978.

S.-C. Kim and P. M. Bobrowski. Evaluating order release mechanisms in a job shop with sequence-dependent setup times. *Production and Operations Management*, 4(2):163–180, 1995. doi: 10.1111/j.1937-5956.1995.tb00048.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1937-5956.1995.tb00048.x`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

M. Knollmann and K. Windt. Control-theoretic analysis of the lead time syndrome and its impact on the logistic target achievement. *Procedia CIRP*, 7: 97–102, 2013.

Averill M. Law and W. David Kelton. *Simulation Modeling & Analysis*. McGraw-Hill, Inc, New York, 3rd edition, 2000.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Sridhar Mahadevan. An average-reward reinforcement learning algorithm for computing bias-optimal policies. In *AAAI/IAAI, Vol. 1*, pages 875–880, 1996a.

Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22:159–195, 1996b.

Sridhar Mahadevan. Optimality criteria in reinforcement learning. In *Proceedings of the AAAI Fall Symposium on Learning Complex Behaviors in Adaptive Intelligent Systems*, 1996c.

Sridhar Mahadevan. Sensitive discount optimality: Unifying discounted and average reward reinforcement learning. In *ICML*, pages 328–336, 1996d.

Sridhar Mahadevan and Georgios Theocharous. Optimizing production manufacturing using reinforcement learning. In *FLAIRS Conference*, pages 372–377, 1998.

Sridhar Mahadevan, Nicholas Marchalleck, Tapas K. Das, and Abhijit Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 202–210, 1997.

H. Mather and G. W. Plossl. Priority fixation versus throughput planning. *Production and Inventory Management*, 19:27–51, 1978.

B. L. Miller and A. F. Veinott. Discrete dynamic programming with a small interest rate. *The Annals of Mathematical Statistics*, 40(2):366–370, 1969. ISSN 00034851. URL `http://www.jstor.org/stable/2239451`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Julia Pahl, Stefan Voss, and David L. Woodruff. Production planning with load dependent lead times: an update of research. *Annals of Operations Research*, 153(1):297–345, 2007. ISSN 0254-5330. doi: 10.1007/s10479-007-0173-5. URL <GotoISI>://WOS:000247203300013.

Carlos D Paternina-Arboleda and Tapas K Das. Intelligent dynamic control policies for serial production lines. *Iie Transactions*, 33(1):65–77, 2001.

Martin L Puterman. Markov decision processes. j. *Wiley and Sons*, 1994.

G. J. Ragatz and V. A. Mabert. An evaluation of order release mechanisms in a job-shop environment. *Decision Sciences*, 19:167–189, 1988.

M. Schneckenreither. Blackwell-optimal reinforcement learning. Working paper., 2019.

M. Schneckenreither and S. Haeussler. *Reinforcement Learning Methods for Operations Research Applications: The Order Release Problem*, page 46. Springer International Publishing, 2019. ISBN 978-3-030-13708-3.

Manuel Schneckenreither. Average reward adjusted discounted reinforcement learning: Near-blackwell-optimal policies for real-world applications. *arXiv preprint arXiv:2004.00857*, 2020.

Manuel Schneckenreither, Stefan Haeussler, and Christoph Gerhold. Order release planning with predictive lead times: a machine learning approach. *International Journal of Production Research*, 2020. doi: 10.1080/00207543.2020.1859634.

Christoph Schneeweiβ. Hierarchical structures in organisations: A conceptual framework. *European Journal of Operational Research*, 86(1):4–31, 1995.

B. Selcuk, J. C. Fransoo, and T. G. De Kok. The effect of updating lead times on the performance of hierarchical planning systems. *International Journal of Production Economics*, 104(2):427–440, 2006.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

Prasad Tadepalli and DoKyeong Ok. Model-based average reward reinforcement learning. *Artificial intelligence*, 100(1-2):177–224, 1998.

I. P. Tatsiopoulos and B. G. Kingsman. Lead time management. *European Journal of Operational Research*, 14(4):351–358, 1983. ISSN 0377-2217.

Arthur F. Veinott. Discrete dynamic programming with sensitive discount optimality criteria. *The Annals of Mathematical Statistics*, 40(5):1635–1660, 1969. doi: 10.1214/aoms/1177697379.

Yi-Chi Wang and John M Usher. Application of reinforcement learning for agent-based production scheduling. *Engineering Applications of Artificial Intelligence*, 18(1):73–82, 2005.

Bernd Waschneck, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, and Andreas Kyek. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, 72(1):1264–1269, 2018.

H. P. Wiendahl. *Load-Oriented Manufacturing Control*. Springer, Berlin, 1st edition, 1995. ISBN 9783642633430.

Günther Zäpfel. *Produktionswirtschaft: Operatives Produktions-Management*. de Gruyter, 1982.

Wei Zhang and Thomas G Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer, 1995.

A Deep Q-Learning Algorithm

Algorithm 3 presents the detailed deep Q-Learning algorithm. It is very similar to the established ARAL algorithm that is given in Algorithm 2. However, it does not compute the average reward. Furthermore, it also operates on a single estimation of the overall future discounted reward.

Algorithm 3 Deep Q-Learning

Initialise state s_0 and network parameters θ_X^T, θ_X^W randomly, set exploration rate $0 < p_{\mathsf{expl}} \leqslant 1$, target network adaption rate $0 < \gamma \ll 1$ and discount factor $0 < \gamma_1 < 1$
while the stopping criterion is not fulfilled **do**
  for each independent agent $i \in \mathsf{ag}_1, \mathsf{ag}_2, \ldots, \mathsf{ag}_n$ **do**
    With probability $\frac{p_{\mathsf{expl}}}{n}$ choose action $a_{t,i}$ randomly or otherwise one of the set defined by $\min_{a \in \mathcal{A}(i)} X_{\gamma_1}^{\pi}(s_t, a; \theta_X^W)$.
  Carry out action $a_t = (a_{t,\mathsf{ag}_1}, \ldots, a_{t,\mathsf{ag}_n})$, observe reward r_t and resulting state s_{t+1}. Store experience $(s_t, a_t, a_t^{\mathsf{rd}}, r_t, s_{t+1})$ in the experience replay memory M.
  if $t \mod n_{\mathsf{step}} \equiv 0$ **then**
    Reset gradient: $d\theta_X^W \leftarrow 0$
    Sample m random mini-batches of n_{step} consecutive experiences from M numbered by $1 \leqslant i \leqslant n_{\mathsf{step}}$, each with $(s_i, a_i^{\mathsf{rd}}, a_i, r_i, s_{i+1})$.
    for each mini-batch **do**
      for each experience, starting with the latest seen experience n_{step} **do**
        if experience n_{step} or $\neg a_i^{\mathsf{rd}}$ or $p_{\mathsf{expl}} > p_{\mathsf{learn}}$ **then**
          $\mathsf{target}_{\gamma_1} \leftarrow \min_a X_{\gamma_1}^{\pi}(s_{i+1}, a; \theta_X^T)$
        $\mathsf{target}_{\gamma_1} \leftarrow r_i + \gamma_1 \mathsf{target}_{\gamma_1}$
        $d\theta_X^W \leftarrow d\theta_X^W + \partial(\mathsf{target}_{\gamma_1} - X_{\gamma_1}^{\pi}(s_{i+1}, a; \theta_X^W))^2 / \partial\theta_X^W$
    Perform update of θ_X^W using $d\theta_X^W / (m \cdot n_{\mathsf{step}})$
  Every C steps exponential smoothly set target network: $\theta_X^T \leftarrow (1-\gamma)\theta_X^T + \gamma\theta_X^W$
  Set $s \leftarrow s'$, $t \leftarrow t+1$ and decay parameters

B Statistical Analysis of the Results

Table 4 and Table 5 provide the statistical results of the computation results. We used the Friedman test with a significance level of $p = 0.05$ for a statistical analysis of the mean sum of costs. As expected the omnibus null hypotheses (all samples are from the same distribution) are rejected for all cases. Therefore, we conducted pairwise Conover post-hoc tests adjusted by the Benjaminyi-Hochberg FDR method (Benjamini and Hochberg, 1995) to reduce liberality. These results are shown in Table 4 and Table 5 for the uniformly and exponentially distributed interarrival times of the demand respectively. The values given in bold are below the significance level of 5% and therefore indicate which row-column tuples are not statistically significant.

70-$\mathcal{U}$	$\mathsf{QL}_{0.99}$	$\mathsf{ARAL}_{0.99}$	$\mathsf{ARAL}_{1.00}$	BIL1	BIL2	BIL3
$\mathsf{ARAL}_{0.99}$	3.37e-04					
$\mathsf{ARAL}_{1.0}$	**3.55e-01**	1.04e-05				
BIL1	1.94e-45	2.48e-51	6.90e-44			
BIL2	1.09e-24	3.35e-16	9.57e-27	3.66e-64		
BIL3	1.78e-20	9.27e-29	2.44e-18	1.35e-22	6.26e-47	
BIL4	6.30e-34	3.85e-41	4.83e-32	1.16e-08	3.00e-56	1.16e-08
80-$\mathcal{U}$	$\mathsf{QL}_{0.99}$	$\mathsf{ARAL}_{0.99}$	$\mathsf{ARAL}_{1.00}$	BIL1	BIL2	BIL3
$\mathsf{ARAL}_{0.99}$	4.78e-16					
$\mathsf{ARAL}_{1.00}$	5.26e-04	3.96e-08				
BIL1	1.02e-40	1.98e-21	3.92e-34			
BIL2	6.98e-10	7.12e-03	2.06e-03	1.31e-27		
BIL3	2.16e-02	8.93e-11	**2.03e-01**	1.57e-36	2.46e-05	
BIL4	6.16e-33	1.10e-11	2.74e-25	5.33e-05	5.38e-18	5.30e-28
90-$\mathcal{U}$	$\mathsf{QL}_{0.99}$	$\mathsf{ARAL}_{0.99}$	$\mathsf{ARAL}_{1.00}$	BIL1	BIL2	BIL3
$\mathsf{ARAL}_{0.99}$	**5.18e-01**					
$\mathsf{ARAL}_{1.00}$	1.94e-04	1.56e-05				
BIL1	8.40e-42	9.61e-43	1.10e-34			
BIL2	2.36e-33	1.10e-34	5.21e-25	1.56e-05		
BIL3	4.16e-04	3.64e-05	**8.19e-01**	6.27e-35	1.81e-25	
BIL4	2.33e-20	6.57e-22	1.79e-11	2.93e-18	4.39e-08	5.97e-12

Table 4: P-Values from the statistical analysis for the results of moderate variability in the system. Tuples marked bold are not significantly different.

70-Exp	$QL_{0.99}$	$ARAL_{0.99}$	$ARAL_{1.00}$	BIL1	BIL2	BIL3
$ARAL_{0.99}$	1.75e-02					
$ARAL_{1.00}$	6.15e-07	5.50e-12				
BIL1	7.62e-35	1.74e-39	1.32e-23			
BIL2	**3.33e-01**	1.02e-03	3.17e-05	6.06e-33		
BIL3	5.96e-17	1.79e-11	6.09e-29	7.00e-52	3.47e-19	
BIL4	1.16e-24	6.25e-30	1.68e-12	4.68e-06	1.56e-22	3.43e-44

80-Exp	$QL_{0.99}$	$ARAL_{0.99}$	$ARAL_{1.00}$	BIL1	BIL2	BIL3
$ARAL_{0.99}$	**7.07e-02**					
$ARAL_{1.00}$	1.74e-03	**1.72e-01**				
BIL1	2.39e-35	6.77e-39	2.38e-41			
BIL2	1.22e-25	1.00e-29	1.21e-32	1.13e-05		
BIL3	2.72e-05	1.33e-02	**2.44e-01**	3.65e-43	5.68e-35	
BIL4	2.07e-13	1.12e-17	6.55e-21	3.62e-17	6.09e-07	1.45e-23

90-Exp	$QL_{0.99}$	$ARAL_{0.99}$	$ARAL_{1.00}$	BIL1	BIL2	BIL3
$ARAL_{0.99}$	2.61e-06					
$ARAL_{1.00}$	5.03e-13	1.74e-03				
BIL1	9.64e-23	2.73e-33	2.22e-39			
BIL2	3.46e-12	1.32e-23	1.30e-30	1.23e-05		
BIL3	**6.86e-01**	1.23e-05	3.46e-12	1.32e-23	5.03e-13	
BIL4	2.45e-04	**2.38e-01**	2.60e-05	6.39e-31	6.30e-21	9.46e-4

Table 5: P-Values from the statistical analysis for the results of high variability in the system. Tuples marked bold are not significantly different.