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

Manuel Schneckenreither and Stefan Haeussler

Department of Information Systems, Production and Logistics Management,
University of Innsbruck, Austria
email: manuel.schneckenreither@uibk.ac.at,stefan.haeussler@uibk.ac.at

Abstract. 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 (acquisition) 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 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

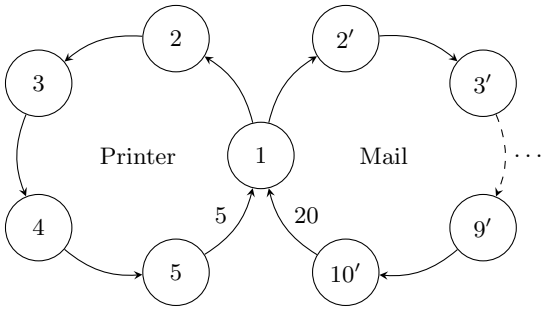

Fig. 1: A MDP with a single action choice in state 1, i.e. two different deterministic policies (Mahadevan, 1996c, adapted).

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

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

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

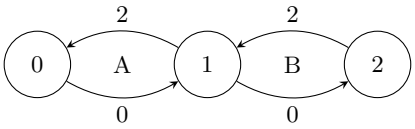

Fig. 2: A MDP with two gain-optimal deterministic policies, π_A going left in 1 is also bias-optimal, while π_B is not (Adapted from Mahadevan, 1996b).

is called gain-optimal. Mahadevan (1996a) shows that n-discount-optimality for $n = -1$ describes gain-optimality.

Further, for an unichain aperiodic[1] 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 $(0, 2, 0, 2, \ldots)$.

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

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

[1] 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.

where the error term $e^{\pi}_{\gamma}(s)$ exists of infinitely many subterms and Puterman (1994) shows that $\lim_{\gamma \to 1} e^{\pi}_{\gamma}(s) = 0$. 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 $V^{\pi}_{\gamma}(s)$. 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 $e^{\pi}_{\gamma}(s)$. 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).

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

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

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

The error term $e^{\pi}_{\gamma}(s)$ is given by $e^{\pi}_{\gamma}(s) = \sum_{n=1}^{\infty} W^{\pi}_{n}(s)$ 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 $\rho^{\pi}(s), V^{\pi}(s), W^{\pi}_{1}, \ldots, W^{\pi}_{M-1}$ are unique, whereas W^{π}_{M} 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 algorithm, 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 $X^{\pi}_{\gamma}(s) = V^{\pi}(s) + u$, 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 ARA-DiRL 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 [**MS:** reformulate avg RL constraints using s' etc.?] 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) 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

Algorithm 1 Near-Blackwell-optimal deep RL for unichain MDPs

1: 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 $0 < \alpha, \gamma < 1$, and discount factors $0 < \gamma_0 < \gamma_1 \leqslant 1$, where $\gamma_1 = 1$ is usually a good choice.
2: **while** the stopping criterion is not fulfilled **do**
3: 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.
4: 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.
5: **if** a non-random action was chosen or $p_{\mathsf{expl}} > p_{\mathsf{learn}}$ **then**

$$\rho^\pi \leftarrow (1-\alpha)\rho^\pi + \alpha[r_t + \max_a X^\pi_{\gamma_1}(s_{t+1}, a; \theta_X^W) - X^\pi_{\gamma_1}(s_t, a_t; \theta_X^W)]$$

6: Sample random mini-batch of experiences $(s_i, a_i^{\mathsf{rd}}, a_i, r_i, s_i)$ from M and do

$$y_{i,\gamma_0} \leftarrow r_t + \gamma_0 \max_a X^\pi_{\gamma_0}(s_{t+1}, a; \theta_X^T) - \rho^\pi$$
$$y_{i,\gamma_1} \leftarrow r_t + \gamma_1 \max_a X^\pi_{\gamma_1}(s_{t+1}, a; \theta_X^T) - \rho^\pi$$

7: Update the average reward adjusted discounted state-values using the sum of the gradients on $(y_{i,\cdot} - X^\pi_\cdot(s_{t+1}, a; \theta_X^W))^2$ wrt. network parameters θ_X^W.
8: Every C steps exponential smoothly set target network: $\theta_X^T \leftarrow (1-\gamma)\theta_X^T + \gamma\theta_X^W$
9: Set $s \leftarrow s'$, $t \leftarrow t+1$ and decay parameters

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 exponentially adapts the parameters of the worker network. [**MS:** describe network initialisation method (Uniform, HeAtAl, etc.)] [**MS:** Extensions: overestimate rho, rho minimum, Multiple Agents (shared state, shared rho?), init-phase (0.5 for expSmthRewRate, always adapt rho, same rho for all agents): exp smooth reward as rho, replay memory actions, n-step (describe replay memory), gradient clipping] [**MS:** Side info: Replay memory also stores (disallowed) filtered action indices for optimisation purposes and on episodic tasks needs to store a bool indicating an episode end. Uses Int8 for state features values.]

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,

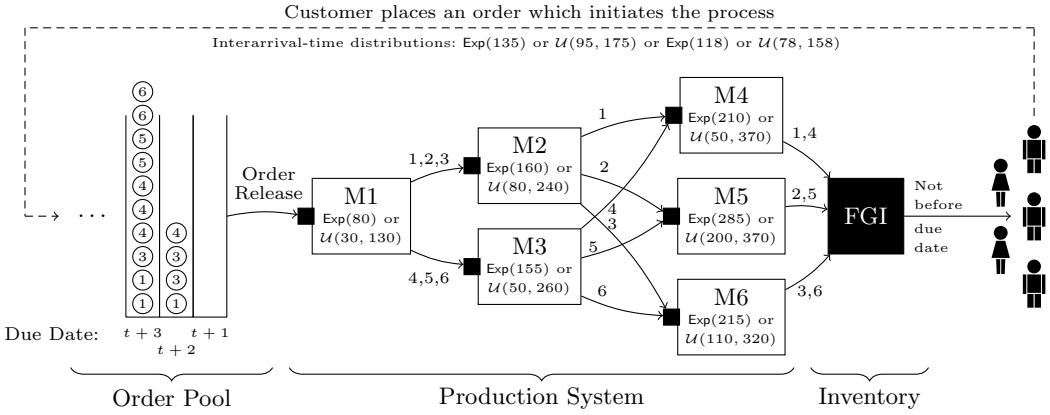

Fig. 3: Production System of the Simulation Model with routing, processing time distributions and demand interarrival time distributions.

this section is largely based on the corresponding section of Schneckenreither et al. (2020).

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 given under the corresponding node labels of the machines. To simulate high or moderate variance the processing times for all machines are drawn from an exponential or an uniform distribution, respectively. 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 10 periods, that is each order arriving within period t will be due at the end of period $t + 10$. The due date slack of 10 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 high and moderate 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 $\mathcal{U}(95, 175)$, or 80% with $\mathsf{Exp}(118)$ and $\mathcal{U}(78, 158)$. 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 Rules

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. Note that we set lead times for each product type and not for every order as done by Ackerman (1963).

3.3 Algorithm Setup

[**MS:** Algorithm adaptions]

Markov Decision Process. The underlying MDP is unichain and looks as follows. Both, state space $\mathcal{S}$ and action space $\mathcal{A}$ are discrete. Any state $s \in \mathcal{S}$ of the *state space* is composed of the following information for each product p:

- The currently set lead time $LT_p \in \{1, 2, \ldots, \mathsf{dds}\}$ (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 $OP_{p,d} \in \mathbb{N}$ 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 $Q_i \in \mathbb{N}$ standing for the number of orders for each queue i.
- Counters $FGI_{p,d} \in \mathbb{N}$ of orders in the finished goods inventory divided in time buckets with $d \in \{-5, -4, -3, \ldots, \mathsf{dds}\}$, which stands for the number of periods until the due date. Orders with a due date with more than 5 periods ago are listed in the counter FGI_{-5}.
- Counters $S_{p,d} \in \mathbb{N}$ of shipped orders from the last period divided in time buckets with $d \in \{-5, -4, -3, \ldots, \mathsf{dds}\}$, which stands for the number of periods until the due date. Orders with a due date with more than 5 periods ago are listed in the counter S_{-5}.

The algorithm implicitly learns a function which maps the current state of the production system to 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 (either all or no orders with the same due date and product type are released).

The *actions space* is composed of two independent decisions. These are the relative changes of the lead times to the currently set lead times $LT_p \in \{1, 2, \ldots, \mathsf{dds}\}$ for each product p. Recall that $p = 2$. Furthermore, we restrict

the action space for each state s_{t+1} according to the last set lead time LT_p from state s_t by restricting the change of the lead time for consecutive periods to a maximum of 1. Thus, if LT_p is the current lead time for product p the action space for this product is given by $\{1, 2, \ldots, \mathsf{dds}\} \cap \{LT_p - 1, LT_p, LT_p + 1\}$. Put differently the algorithm can increase or decrease the lead time by 1 or leave it as it is, as long as it acts within the discrete action space given by the set $\{1, 2, \ldots, \mathsf{dds}\}$ for each type of product. Thus the action space over all products is given by the full enumeration of available actions of the individual products.

Reward. In each period the agent chooses an action which generates 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. These costs are consist of the number of backorders, the current WIP level and the number of orders in the inventory.

Adapted Algorithm. The algorithm depicted above is designed as a tabular version. We have adapted it to use neural networks for the approximation of the different kind of state-action pair value functions. This is required due to the exponential growth of the state space. The implementation collects the observed information for each state-action function in a replay memory and randomly selects 128 data points out of a collection of $30k$ memories in each period and trains these on the neural network. This technique is called experience replay memory (Lin, 1993; Mnih et al., 2015) and ought to overcome we instabilities. Nonetheless, this approximation results in the fact that the algorithm can be unstable or may even diverge (Tsitsiklis and Van Roy, 1997). Additionally we use a target and a worker network, whereas the target network is overwritten every $10k$ steps by the worker network. This has been proposed by Mnih et al. (2015) and should further stabilize the algorithm.

Neural Network Setup. We use the same setup for the neural network as done by Schneckenreither (2019). However as they use a actor-critic agent, whereas we approximate state-action-values the output of the network is a one dimensional vector as opposed to a matrix. Therefore we use a single three layer fully-connected network where the number of nodes and activation functions are 41-ReLU[2]-89-ReLU-20-ReLU-9, with the output activations being Tanh (hyperbolic tangent function). The output for both networks consist of $3^2 = 9$ nodes. This is due to the fact that all combinations of increase, decrease and no change of the lead time for each product type have to be represented. For back-propagation we use the Adam optimiser with learning rate 0.001, and beta-values $\beta_1 = 0.9$, and $\beta_2 = 0.999$.

[2] ReLU stands for rectified linear unit.

Parameter	α	β	δ	γ	ϵ	p_{exp}	ξ
Value	0.5	0.15	0.15	0.15	0.5	1.0	0.15

Table 1: Parameter Setup.

4 Preliminary Results

This section provides preliminary results for the described setup and gives an overview of the current performance of the algorithm compared to the conventional static lead time setting algorithms.

Currently we evaluate two different kinds of reward reporting as they have been proposed by Schneckenreither (2019). They find that it is beneficial to overcome the time offset imposed by the production system. That is, although orders are immediately released to the production system and thus generate a response in terms of a reward to the agent, the full impact of the chosen action takes several periods to materialise. For instance, by deciding not to release an order, which later becomes a backorder, it is beneficial to reward the actions actually responsible for the backorder instead of the last action (only). Thus they propose to keep track of the orders currently in the order pool until all fully traversed through the production system and reward the actions according to all orders in the order pool.

The parameter setup is given in Table 1. The parameters are exponentially decayed with rate 0.05 and $350k$ steps[3]. We implemented the deep Q-network algorithm proposed by Mnih et al. (2015), which is parameterised by learning rate δ and a discount factor of 0.99.

Table 2 shows the results, where we evaluate the average reward and discounted reinforcement learning algorithm by providing either direct reward feedback, that is the reward accumulated in the next period, without keeping track of any orders, as MLShipped and MLShipped_DQN respectively. Further both algorithms are evaluated with keeping track of the order pool orders, such that each action is responsible for all orders in the order pool. These are labelled MLOrdPool and MLOrdPool_DQN. For comparisons reasons we provide $\mathsf{BIL1} - \mathsf{BIL6}$ and immediate release (interval release).

The results show that at the current time we are facing issues with scaling the algorithm to the order release problem. Likely this is due to parameterisation issues, which we are investigating currently. However, it can be seen that i) the technique of keeping track of the order pool orders leads to better policies and ii) that despite the parameterisation issues average reward reinforcement learning finds better solutions than its discounted counterpart. However, the least costs are generated by BIL3, which shows that there is room for improvement for the reinforcement learning policies.

Interesting is also to see, that all reinforcement learning variants are releasing rather late, leading to low finished goods inventory costs (FGIC), but high

[3] The decay is defined as $0.05^{(t/350k)}$, where t is the current period

Algorithm/KPI	SUM	BOC	FGIC	WIPC	TARD	σTARD	SFTT
MLShipped	1777.74	1322.04	94.49	361.20	2.24	1.52	3.44
MLOrdPool	1593.03	1318.65	14.01	260.36	2.11	1.42	2.68
MLShipped_DQN	1940.62	1570.40	6.30	363.92	2.35	1.59	3.95
MLOrdPool_DQN	1618.88	1306.20	40.68	272.00	2.08	1.38	2.58
BIL1	1736.98	1510.42	0.00	226.56	2.04	1.26	2.16
BIL2	1081.79	769.86	86.89	225.03	1.87	1.18	2.15
BIL3	**922.36**	362.59	334.47	225.30	1.74	1.07	2.15
BIL4	1062.68	155.72	681.03	225.93	1.63	0.99	2.15
BIL5	1380.04	65.45	1088.94	225.64	1.67	1.02	2.15
BIL6	1777.77	25.04	1529.77	222.96	1.61	0.85	2.13
Immediate Release	2209.97	12.08	1974.72	223.16	1.58	0.63	2.14

Table 2: Evaluation results where the monetary units are in k-values.

backorder costs (BOC). Furthermore, the agents based on the order-pool orders, namely MLOrdPool and MLOrdPool_DQN, are able achieve lower work-in-process costs (WIPC) and also lower tardiness, than the other the agents rewarded directly on the shipped orders.

5 Conclusion

This paper introduces deep theoretical insights in reinfocement learning and provides first evidence that average reward reinfocement learning is better applicable to operations research problems than the discounted framework. The paper describes an application of an order release model based on reinforcement learning. The performance is tested on a multi-product, two-stage hypothetical flow shop and is measured by cost, delivery and lead time related measures. We show that in the current version the machine learning approach is not able to outperform all other tested order release approaches. In the future we expect to be able to yield better results by investigating the parameterisation and resolving current issues when scaling average reinforcement learning to complex scenarios.

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.

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.

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`.

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.

Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

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 Gerold. Order release planning with predictive lead times: A machine learning approach. *The International Journal of Production Research*, 2020. Accepted.

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.

M. Thuerer, M. Stevenson, and T. Qu. Job sequencing and selection within workload control order release: an assessment by simulation. *International Journal of Production Research*, 50:5048–5062, 2016.

John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

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 N-Discount-Optimality

This part proves (or rather ought to prove, as this is only the first version of such a proof) Blackwell-optimality of the average reward adjusted reinforcement learning algorithm shown in Algorithm 1. The idea is to prove n-discount optimality for $n = -1$, $n = 0$, and finally for $n \geqslant 1$. But first recall the definition of n-discount-optimality.

Definition *Due to Veinott (1969) for MDPs 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*

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

A.1 (−1)-Discount-OptimalityMahadevan (1996a).

Here we provide insights in the algorithm and n-discount-optimality, where we concentrate on $n = -1$. Recall that the discounted state value can be expanded by the Laurent series expansion: $V_\gamma^\pi = \frac{\rho^\pi(s)}{1-\gamma} + V^\pi + e_\gamma^\pi(s)$, where $\lim_{\gamma \to 1} e_\gamma^\pi(s) = 0$

$$\begin{aligned}
\lim_{\gamma \to 1} (1-\gamma)^1 \cdot (V_\gamma^{\pi^\star}(s) - V_\gamma^{\pi}(s)) &\geqslant 0 \\
\lim_{\gamma \to 1} (1-\gamma) \cdot (\frac{\rho^{\pi^\star}(s) - \rho^\pi(s)}{1-\gamma} + V^{\pi^\star}(s) - V^\pi(s) + e_\gamma^{\pi^\star}(s) - e_\gamma^\pi(s)) &\geqslant 0 \\
\lim_{\gamma \to 1} (\rho^{\pi^\star}(s) - \rho^\pi(s) + (1-\gamma) \cdot (V^{\pi^\star}(s) - V^\pi(s) + e_\gamma^{\pi^\star}(s) - e_\gamma^\pi(s)) &\geqslant 0 \\
\rho^{\pi^\star}(s) - \rho^\pi(s) &\geqslant 0
\end{aligned}$$

That is a policy π which ought to be (-1)-discount-optimal has to maximise the average reward $\rho^\pi(s)$ over all states s.

A.2 0-Discount-OptimalityMahadevan (1996a).

Note that a policy can only be 0-discount-optimal if it is -1-discount-optimal, thus $\rho^{\pi^\star}(s) = \rho^\pi(s)$. Furthermore, recall that $\lim_{\gamma \to 1} e_\gamma^\pi(s) = 0$.

$$\begin{aligned}
\lim_{\gamma \to 1} (1-\gamma)^0 \cdot (V_\gamma^{\pi^\star}(s) - V_\gamma^{\pi}(s)) &\geqslant 0 \\
\lim_{\gamma \to 1} (\frac{\rho^{\pi^\star}(s) - \rho^\pi(s)}{1-\gamma} + V^{\pi^\star}(s) - V^\pi(s) + e_\gamma^{\pi^\star}(s) - e_\gamma^\pi(s)) &\geqslant 0 \\
\lim_{\gamma \to 1} (\frac{0}{1-\gamma} + V^{\pi^\star}(s) - V^\pi(s)) &\geqslant 0 \\
V^{\pi^\star}(s) - V^\pi(s) &\geqslant 0
\end{aligned}$$

Therefore a 0-discount-optimal policy π has to maximise the bias values $V^\pi(s)$ for all states s.

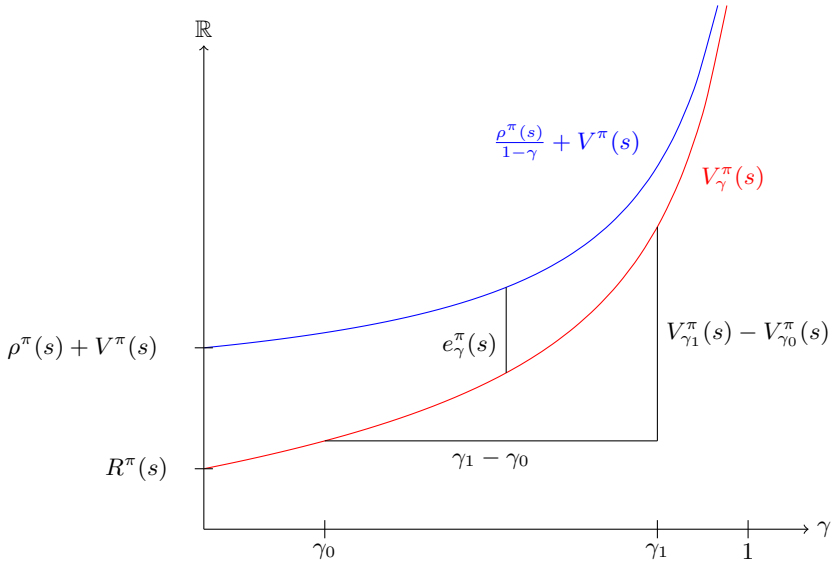

Fig. 4: Visualisation of the monotonically increasing state values for a state s and the corresponding decreasing error term $e^\pi_\gamma(s)$ as γ approaches 1.

A.3 n-Discount-Optimality for n $\geqslant$ 1.

The following derivation provides an important insight to the analysis. Recall that for unichain MDPs the average reward $\rho^\pi(s)$ for all states s is equal and stated as ρ^π. Furthermore, as the policy is 0-discount-optimal $V^{\pi^\star}(s) = V^\pi(s)$.

$$\lim_{\gamma\to 1}(1-\gamma)^{-n} \cdot (V^{\pi^\star}_\gamma(s) - V^\pi_\gamma(s)) \geqslant 0$$

$$\lim_{\gamma\to 1}(1-\gamma)^{-n} \cdot (\frac{\rho^{\pi^\star}(s)}{1-\gamma} + V^{\pi^\star}(s) + e^{\pi^\star}_\gamma(s) - \frac{\rho^\pi(s)}{1-\gamma} - V^\pi(s) - e^\pi_\gamma(s)) \geqslant 0$$

$$\lim_{\gamma\to 1}(1-\gamma)^{-n} \cdot (e^{\pi^\star}_\gamma(s) - e^\pi_\gamma(s)) \geqslant 0$$

As stated before $e^{\pi^\star}_\gamma(s)$ approaches 0 as $\gamma \to 1$. Therefore, it is important to note that for $n > 0$ we analyse the case when γ is strictly less than 1. This means that the number of actions to reach a desired goal is taken into account. Therefore, a ∞-discount-optimal policy π has to maximise the error term $e^\pi_\gamma(s)$ for all states s. That is, an ∞-discount-optimal algorithm has to choose the action with the largest error term $e^\pi_\gamma(s)$ once $\gamma \to 1$. However, as the error term depends on infinitely many sub-terms simply estimating these and summing up does not work.

Therefore, the idea is to use two discounted state-values with different γ's to infer the slope of the error term. Figure 4 illustrates the idea. Here γ_1 and γ_0

are used to estimate the difference of $V^\pi_{\gamma_1}(s)$ and $V^\pi_{\gamma_0}(s)$. As the average reward and the bias values are equal (recall 0-discount-optimality) the difference is a direct estimate of the slope of the error term. Thus the larger the increase of the difference of the discounted state values when comparing 0-discount-optimal actions the better the action. However, for this to hold true we have to prove strict monotonicity of $V^\pi_\gamma(s) = \frac{\rho^\pi(s)}{1-\gamma} + V^\pi(s) + e^\pi_\gamma(s)$ under increasing γ. In the sequel we assume $\rho^\pi(s) \geqslant 0$. Note that this assumption is easily met by adding a constant reward for any action in cases where $\rho^\pi(s) < 0$. This assumption eases the following analysis and makes sense as usually we are interested in problems which accumulate positive average reward, e.g. profit or points, over time.

Remark 4. Nonetheless, it is an important insight that if $\rho^\pi(s) < 0$ then $\frac{\rho^\pi(s)}{1-\gamma} + V^\pi(s)$ approaches negative infinity (as opposed to positive infinity, cf. Figure 4). Therefore, in such cases the following optimisation objectives would need to be inverted.

Lemma 5. *The error term $e^\pi_\gamma(s) = \sum_{m=1}^\infty (\frac{1-\gamma}{\gamma})^m \cdot y_m$ strictly monotonically decreases as γ increases.*

Proof. Note that $e^\pi_\gamma(s) = \sum_{m=1}^\infty (\frac{1-\gamma}{\gamma})^m \cdot y_m$, where the sign of y_m alternates in each step. This is due to the fact that $y_m = \mathbb{E}_\pi[y_m] - y_{m-1}$ for all $m \geqslant 1$ (Puterman, 1994, p.346), that is y_m only depends on y_{m-1} and the underlying (stationary) policy. Further y_0 is the bias value $V^\pi(s)$. We consider three cases:

1) $\gamma = 0.5$: The factor $(\frac{1-\gamma}{\gamma})^m$ is 1.
2) $0.5 < \gamma < 1$: The factor $(\frac{1-\gamma}{\gamma})^m$ is < 1 and strictly monotonically converging towards 0.
3) $0 < \gamma < 0.5$: The factor $(\frac{1-\gamma}{\gamma})^m$ is > 1 and strictly monotonically converging towards ∞.

Case 1) As $\frac{1-\gamma}{\gamma} = 1$ the term reduces to $e^\pi_\gamma(s) = \sum_{m=1}^\infty y^\pi_m$. We use the definition

$$y^\pi_m = (-1)^m H^{m+1}_\pi h_\pi \ ,$$

where H_π is the deviation matrix defined as

$$H_\pi = \underset{N\to\infty}{\text{C-lim}} \sum_{t=0}^{N-1} P^t - P^\star \ ,$$

C-lim the Cesaro limit[4], $P^\star$ the stationary matrix defined as $P^\star = \text{C-lim}_{N\to\infty} P^N$ and h_π a column vector holding the bias values of the states (Miller and Veinott,

[4] Recall that the Cesaro limit (with degree 1) is only required in periodic MDPs to ensure stationary matrices. For aperiodic MDPs they can be read as normal limits. (Puterman, 1994, p.592ff)

1969; Puterman, 1994). Thus, we get

$$e_\gamma^\pi(s) = \sum_{m=1}^{\infty} y_m^\pi(s) = \sum_{m=1}^{\infty} y_{2m}^\pi(s) + y_{2m+1}^\pi(s) = \sum_{m=1}^{\infty} H^{2m+2}h - H^{2m+1}h$$
$$= \sum_{m=1}^{\infty} (H^{2m+2} - H^{2m+1})h \; .$$

It suffices[5] to show that $H^{2m+2} - H^{2m+1} < 0$ for all $m \geqslant 1$ to prove strict monotonicity of $e_\gamma^\pi(s)$. However, this directly follows as for MDPs with at least two states $H - 1 < 0$ in

$$e_\gamma^\pi(s) = \sum_{m=1}^{\infty} (H^{2m+2} - H^{2m+1})h = \sum_{m=1}^{\infty} H^{2m+1}(H-1)h \; .$$

Case 2) As the factor $(\frac{1-\gamma}{\gamma})^m$ is < 1 and strictly monotonically decreasing for an increasing $m \geqslant 1$, we use the same argument as in the previous case (Case 1) with which the claim follows.

Case 3) This part of the proof is open!

Corollary 6. *For any two* 0*-discount-optimal stationary policies* π_1, π_2 *with stationary transition matrices* P_1 *and* P_2 *respectively, the differences of the slopes of the error terms* $e_\gamma^{\pi_1}(s)$ *and* $e_\gamma^{\pi_2}(s)$ *for any* γ *is dependent on* H_{π_1} *and* H_{π_2} *and thus* P_1 *and* P_2 *solely.*

Proof. Note that due to 0-discount-optimality the bias values $h_1 = h_2$ are equal for both policies. Therefore as for any given H the error term is determined by $e_\gamma^\pi(s) = \sum_{m=1}^{\infty}((\frac{1-\gamma}{\gamma})^{2m+1}H^{2m+2} - (\frac{1-\gamma}{\gamma})^{2m}H^{2m+1})h = \sum_{m=1}^{\infty}(\frac{1-\gamma}{\gamma})^{2m}H^{2m+1}(\frac{1-\gamma}{\gamma}H - 1)h$ the claim follows directly.

Corollary 7. *The error term function* $e_\gamma^\pi(s)$ *for* $\gamma \to 1$ *and for any given policy* π *is continuous.*

Proof. The claim follows directly from Corollary 6 and due to the shape of the term $e_\gamma^\pi(s) = \sum_{m=1}^{\infty}(\frac{1-\gamma}{\gamma})^m H^{m+1}h$.

Theorem 8. *If* $\rho^\pi(s) > 0$ *then* $V_\gamma^\pi(s) = \frac{\rho^\pi(s)}{1-\gamma} + V^\pi(s) + e_\gamma^\pi(s)$ *is strictly monotonically increasing as* γ *approaches* 1*.*

Proof. Due to $\rho^\pi(s) > 0$ and $0 < \gamma < 1$ the term $\frac{\rho^\pi(s)}{1-\gamma}$ strictly monotonically approaches infinity as $\gamma \to 1$. Furthermore, by Lemma 5 and as $\lim_{\gamma\to 1} e_\gamma^\pi(s) = 0$ the theorem follows immediately.

[5] This claim has not been proven yet!

Theorem 9. *If $\rho^\pi(s) \geqslant 0$ and for γ-values γ_0, γ_1 with $0.5 \leqslant \gamma_0 < \gamma_1 < 1$ a Blackwell-optimal agent chooses the action a with expected future state s' that maximises the expected*[6] *discounted state value difference $\Delta V^\pi_\gamma(s') = V^\pi_{\gamma_0}(s') - V^\pi_{\gamma_1}(s')$ of the set of 0-discount-optimal actions available and for which*

a) $\frac{\rho}{1-\gamma_0} + V^\pi(s') < V^\pi_{\gamma_0}(s')$ *holds, or if no such actions exists, then for which*
b) $\frac{\rho}{1-\gamma_0} + V^\pi(s') \geqslant V^\pi_{\gamma_0}(s')$ *hold.*

Remark 10. In Theorem 9 we have chosen to base the decision on the values generated with γ_0 as the error term is greater with smaller γ-values.

Proof. First observe that due to strict monotonicity of $\frac{\rho^\pi(s')}{1-\gamma}$ and $e^\pi_\gamma(s')$ and as $\lim_{\gamma\to 1} e^\pi_\gamma(s') = 0$ the discounted state values $V^\pi_\gamma(s')$ and the sum of average reward values $\frac{\rho^\pi(s')}{1-\gamma} + V^\pi(s')$ converge with increasing γ. Thus for any state s' either $e^\pi_\gamma(s') > 0$ or $e^\pi_\gamma(s') < 0$ for all γ-values. That is $V^\pi_\gamma(s')$ converges to $\frac{\rho^\pi(s)}{1-\gamma} + V^\pi(s')$ for $\gamma \to 1$ from above or below. Clearly just considering the slope of the error term is insufficient as for any states s_1, s_2 it may happen to be that $e^\pi_\gamma(s_1) > 0$ while $e^\pi_\gamma(s_2) < 0$. Thus we split the decision in cases, where in case *a)* we consider all states s_1 with $e^\pi_\gamma(s_1) > 0$ and in case *b)* we states s_2 with $e^\pi_\gamma(s_2) < 0$. Note that $e^\pi_\gamma(s_1) > e^\pi_\gamma(s_2)$ which explains the prioritisation of case *a)* over case *b)*.

Within these two sets of states we investigate the slopes of the error term.

[**MS:** Todo: Show that two points are sufficient for a proxy of the slope.] Thus the slope can be approximated by

$$\frac{\Delta V^\pi_\gamma(s)}{\Delta\gamma} = \frac{e_{\gamma_1}(s) - e_{\gamma_0}(s)}{\gamma_1 - \gamma_0} \,.$$

As $\gamma_1(s) - \gamma_0(s)$ is constant the difference of the error terms $e_{\gamma_1}(s) - e_{\gamma_0}(s)$ is sufficient to compare the slopes of the error term and can be computed by

$$e_{\gamma_1}(s) - e_{\gamma_0}(s) = \frac{\rho^\pi(s)}{1-\gamma_1} - \frac{\rho^\pi(s)}{1-\gamma_0} - V^\pi_{\gamma_1}(s) + V^\pi_{\gamma_0}(s) \,. \tag{5}$$

However, as all possible future states are gain-optimal $\rho^\pi(s)$ is equal for all states s under consideration. Therefore, the difference $V^\pi_{\gamma_0}(s) - V^\pi_{\gamma_1}(s)$ suffices for comparison purposes between any two states s_1, s_2:

$$e_{\gamma_1}(s_1) - e_{\gamma_0}(s_1) - e_{\gamma_1}(s_2) + e_{\gamma_0}(s_2) = V^\pi_{\gamma_0}(s_1) - V^\pi_{\gamma_1}(s_1) - V^\pi_{\gamma_0}(s_2) + V^\pi_{\gamma_1}(s_2) \,.$$

Then due to Corollaries 6 and 7 we conclude that $\lim_{\gamma\to 1}(1-\gamma)^{-n}\ (V^{\pi^\star}_\gamma(s) - V^\pi_\gamma(s)) \geqslant 0$, where γ is the discount factor and $V^\pi_\gamma(s)$ the value function.

[6] To ease readability and as we aim for model-free RL we have dropped the expectations in the formulas.

Remark 11. Note that this means a Blackwell-optimal agent collects the rewards as soon as possible, as it maximises the error term $e^\pi_\gamma(s)$.

Thus under the assumption of correct approximations and by strict monotonicity of e^π_γ we conclude Blackwell optimality of the algorithm. Figure 4 visualises the discounted state values and the idea of estimating the slope of $e^\pi_\gamma(s)$ in comparison to $\frac{\rho^\pi(s)}{1-\gamma} + V^\pi(s)$.

A.4 Refinements

The main limitation of the straightforward approach of Theorem 9 is that it requires accurate estimations of the bias values when splitting the actions into the two sets with increasing and deceasing error term value. However, to be able to estimate $V^\pi(s)$ accurately enough one usually has to go through a tedious process of investigating many different parameterisations, which is a cumbersome task, especially if the correct values are not known. Furthermore, the process of finding the correct values is computationally very expensive in comparison to finding the deviations between any two values.

Therefore, we have developed another approach of selecting the best possible action which does not depend on correct bias values. In this approach we utilise Equation 5 which allows a determination of the slope simply by rectifying the additional slope imposed due to the term $\frac{\rho^\pi(s)}{1-\gamma}$. As $V^\pi_\gamma(s)$ and $\rho^\pi(s)$ are computationally cheap they can be usually accurately computed.

Therefore in the implementation we propose to replace Theorem 9 by following the following one.

Theorem 12. *If $\rho^\pi(s) \geqslant 0$ and for γ-values γ_0, γ_1 with $0.5 \leqslant \gamma_0 < \gamma_1 < 1$ a Blackwell-optimal agent chooses the action a with expected future state s' that maximises the expected error term difference $\Delta e^\pi_{\gamma_1,\gamma_0}(s') = V^\pi_{\gamma_1}(s') - V^\pi_{\gamma_0}(s') - \rho^\pi(s')(\frac{1}{1-\gamma_1} - \frac{1}{1-\gamma_0})$ of the set of 0-discount-optimal actions available and for which*

a) $\Delta e^\pi_{\gamma_1,\gamma_0}(s') < 0$ holds, or if no such actions exists, then for which
b) $\Delta e^\pi_{\gamma_1,\gamma_0}(s') \geqslant 0$ hold.

Proof. We have for any state s' either $e^\pi_\gamma(s') > 0$ or $e^\pi_\gamma(s') < 0$ for all γ-values (see proof of Theorem 9). Therefore, by definition states s' with $e^\pi_{\gamma_1,\gamma_0}(s') < 0$ have a decreasing error term value. As $e^\pi_{\gamma_1,\gamma_0}(s')$ is a direct measure for the slope of the error term, and by the characteristics on the term specified in Corollaries 6 and 7, we can conclude that the maximum expected error term difference for negative values of $e^\pi_{\gamma_1,\gamma_0}(s')$ impose the greatest $V^\pi_\gamma(s')$ when $\gamma \to 1$. The proof for $e^\pi_{\gamma_1,\gamma_0}(s') \geqslant 0$ follows the same argument. Therefore, the Theorem provides a way to find the policy $\pi^\star$ with $\lim_{\gamma\to 1}(1-\gamma)^{-n} \cdot (e^{\pi^\star}_\gamma(s') - e^\pi_\gamma(s')) > 0$ for any other policy π. □

B Bellman Optimality Formulas and Derivations (Incomplete)

Bellman optimality equations for the average reward.

$$
\begin{aligned}
V^\star(s) &= \max_{a \in A(s)} R^{\pi^\star}(s,a) \\
&= \max_{a \in A(s)} \mathop{\mathbb{E}}_{\pi^\star}[\sum_{t=0}^{\infty}(R_t(s) - \rho^{\pi^\star}(s)) \mid s = s_t, a_t = a] \\
&= \max_{a \in A(s)} \mathop{\mathbb{E}}_{\pi^\star}[R_0(s) - \rho^{\pi^\star}(s) + \sum_{t=0}^{\infty}(R_t(s_{t+1}) - \rho^{\pi^\star}(s_{t+1})) \mid s = s_t, a_t = a] \\
&= \max_{a \in A(s)} \mathbb{E}[R_0(s) - \rho^{\pi^\star}(s) + V^\star(s_{t+1}) \mid s = s_t, a_t = a] \\
&= \max_{a \in A(s)} (r(s,a) - \rho^{\pi^\star}(s) + \sum_{s_{t+1}} p(s_{t+1}|s,a) \cdot V^\star(s_{t+1}))
\end{aligned}
$$

The value of an action in a given state must equal the expected return for following an optimal policy from that state:

$$
\begin{aligned}
R^\star(s,a) &= \mathbb{E}[r(s,a) - \rho^{\pi^\star}(s) + V^\star(s_{t+1}) \mid s = s_t, a = a_t] \\
&= r(s,a) - \rho^{\pi^\star}(s) + \mathbb{E}[\max_{a_{t+1} \in A(s_{t+1})} R^\star(s_{t+1}, a_{t+1}) \mid s = s_t, a = a_t] \\
&= r(s,a) - \rho^{\pi^\star}(s) + \sum_{s_{t+1}} p(s_{t+1}|s,a)(\max_{a_{t+1} \in A(s_{t+1})} R^\star(s_{t+1}, a_{t+1}))
\end{aligned}
$$

Bellman optimal equations for the bias value.

$$
\begin{aligned}
W^\star(s) &= \mathbb{E}[R_t(s) + V^\star(s_{t+1}) | s = s_t, a = a_t] \\
&= \mathbb{E}[R_t(s) + \max_{a_{t+1} \in A(s_{t+1})} R^\star(s_{t+1}, a_{t+1}) | s = s_t, a = a_t]
\end{aligned}
$$

$$
\begin{aligned}
W^\star(s,a) &= \mathbb{E}[r(s,a) + V^\star(s_{t+1}) \mid s = s_t, a = a_t] \\
&= r(s,a) + \mathbb{E}[\max_{a_{t+1} \in A(s_{t+1})} R^\star(s_{t+1}, a_{t+1}) \mid s = s_t, a = a_t] \\
&= r(s,a) + \sum_{s_{t+1}} p(s_{t+1}|s,a)(\max_{a_{t+1} \in A(s_{t+1})} R^\star(s_{t+1}, a_{t+1}))
\end{aligned}
$$

Difference R_{n+1} to W_{n+1}?