Practical Reinforcement Learning -
Experiences in Lot Scheduling Application

Hannu Rummukainen ∗ Jukka K. Nurminen ∗∗

∗ VTT Technical Research Centre of Finland, P.O. Box 1000,
FI-02044 VTT, Finland (e-mail: Hannu.Rummukainen@vtt.fi)
∗∗ VTT Technical Research Centre of Finland, P.O. Box 1000,
FI-02044 VTT, Finland (e-mail: Jukka.K.Nurminen@vtt.fi)

Abstract: With recent advances in deep reinforcement learning, it is time to take another
look at reinforcement learning as an approach for discrete production control. To investigate
the approach in practice, we applied proximal policy optimization (PPO), a recently developed
algorithm for deep reinforcement learning, to the stochastic economic lot scheduling problem.
The problem involves scheduling manufacturing decisions on a single machine under stochastic
demand, and despite its simplicity remains computationally challenging. We implemented two
parameterized models for the control policy and value approximation, a linear model and a
neural network, and used a modified PPO algorithm to seek the optimal parameter values.
Benchmarking against the best known control policy for the test case, in which Paternina-
Arboleda and Das (2005) combined a base-stock policy and an older reinforcement learning
algorithm, we improved the average cost rate by 2 %. Our approach is more general, as we do
not require a priori policy parameters such as base-stock levels, and the entire policy is learned.

Keywords: Reinforcement learning; Stochastic economic lot scheduling; Learning control;
Stochastic control; Monte Carlo simulation; Neural networks; Machine learning.

1. INTRODUCTION

Reinforcement learning has attracted a lot of attention lately with several successful applications in the field of
games, most notably with the AlphaGo software beating top human players in the game of Go: Silver et al. (2016),
Metz (2017). In this work we investigate applying recent advances in reinforcement learning to production control.

Reinforcement learning covers a variety of computational methods to learn to maximize a goal function by trial
and error from incomplete observations of the controlled system, using the formal framework of Markov decision
processes. Sutton and Barto (2018) trace the origins of reinforcement learning to psychology of animal learning,
optimal control, and temporal-difference methods, with the different threads coming together in the 1980’s.

Bertsekas (1995) noted that neural networks can be used to approximate the value (utility) of an observed system
state when the state space is large. Before AlphaGo, Mnih et al. (2013) combined deep neural networks and
reinforcement learning to successfully play video games from observations of screen pixels alone. The combination,
known as deep reinforcement learning, profits from the body of modern deep learning research, including advances
in both software tooling and hardware capabilities.

While the success stories with games are impressive, the question arises whether the same approach can be used in
real world decision making situations. Real life problems are often fuzzier and more unpredictable than games
that operate by well defined rules. Moreover, Henderson et al. (2018) report that deep reinforcement learning
results can be highly sensitive to parameter values and implementation details. As far as we know, there are only
few real-world applications of reinforcement learning in production use. Gauci et al. (2018) indicate that Facebook
uses reinforcement learning to manage notifications sent to users and administrators. Sutton and Barto (2018)
state that the personalized recommendation system of Theocharous et al. (2015) is in production use at Adobe.

Recent work on scheduling by reinforcement learning is predominantly in discrete time as in Mao et al.
(2016); Waschneck et al. (2018) consider a continuous-time discrete-event model, but they apply discrete-time control.

Our aim in this paper is to look at how state-of-the-art reinforcement learning can be applied to production control in a discrete-event setting, and what kind of challenges a practitioner will encounter. To this end, we performed experiments on an idealized lot scheduling problem that is based on real-world production planning issues. Our reasons for selecting the case study were that the model relies on a discrete-event simulation of a manufacturing system; the model is relatively simple yet challenging to solve; and finally there is a published application of an older reinforcement learning algorithm to the problem. The main contributions of this paper are:

(1) Reinforcement learning based solution for lot scheduling, which is more general than the approach of
Paternina-Arboleda and Das (2005) (Section 3)
Table 1. Parameters for the simulation model

<table>
<thead>
<tr>
<th>Product</th>
<th>Set-up time (h)</th>
<th>Production rate (pcs/h)</th>
<th>Holding cost ($/h)</th>
<th>Backlog cost ($/h)</th>
<th>Set-up cost ($/set-up)</th>
<th>Mean demand (pcs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>45</td>
<td>1</td>
<td>25</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>30</td>
<td>1</td>
<td>12.5</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>0.15</td>
<td>35</td>
<td>1</td>
<td>20.0</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

(2) Modification of the proximal policy optimization algorithm (PPO) for a semi-Markov decision process (Section 3.3)

(3) Results of the reinforcement learning based solution with the modified PPO algorithm in a computational experiment (Section 4)

(4) Key findings from our experiments (Section 4.3)

2. LOT SCHEDULING PROBLEM

The stochastic economic lot scheduling problem (SELSP) involves scheduling production of multiple standardized products on a single machine that requires significant time and effort to switch from one product to another. Thus, producing large lots of one product at a time is more efficient than switching products quickly according to customer demand; but on the other hand holding large lots in stock can also be costly. Customer demand is not known in advance, and is assumed to be random. The objective is to decide when to produce which products, in order to optimize a performance indicator such as expected customer waiting time, or expected production and stock holding costs. We address the problem variant in which all demand must be eventually satisfied: when the stock of a product is empty, orders of that product are queued in backlog. Orders in backlog incur additional cost over time.

Winands et al. (2011) survey known approaches to the stochastic economic lot scheduling problem. The problem is commonly addressed by various structured control policies; for example, in a fixed-sequence base-stock policy the current product is switched in a predetermined order, and each product is manufactured until its stock level reaches a predetermined base-stock level. Such a cyclical base-stock policy is determined by 1) the product sequence and 2) the base-stock levels (one scalar parameter per product). Finding the best-performing product sequence and base-stock levels is in itself a non-trivial optimization problem. Several deterministic versions of the problem have been shown to be NP-complete by Gallego and Shaw (1997).

Our case study is based on a version of the SELSP studied by Anupindi and Tayur (1998). Continuing with the same case, Paternina-Arboleda and Das (2005) applied reinforcement learning to the product sequencing decisions of a base-stock policy: Most of the production decisions are made by the base-stock policy and not the reinforcement learning algorithm. Moreover, the base stock levels were provided as fixed input parameters to the reinforcement learning system, and had to be determined by exhaustive search within a limited region. In contrast, our solution learns a general policy that makes an independent decision on every step, and the policy is not limited by predetermined policy parameters such as base-stock levels.

In the simulation model of the case study there are $N = 3$ different products. Customer orders arrive at independent random intervals, with log-normal distribution of mean 2 h and standard deviation 1 h. The amount of each product requested in a customer order is also distributed as log-normal, with product-specific means, and squared coefficient of variation fixed at 0.25. All product-specific parameters of the problem are shown in Table 1.

Control actions are chosen at the following discrete times:
- Immediately after each item has been manufactured.
- At the arrival of an order when the machine is idle.

Let $X_i(t)$ be the inventory level of product $i$ at time $t$, with negative inventory levels representing backlog, i.e. customer orders that are waiting to be fulfilled. At time $t$, the controller observes the inventory levels $X_i(t)$ for $i = 1, ..., N$ and the currently set-up product $P(t)$. The controller has a choice between $N + 1$ possible actions: either leave the machine idle, or produce one item of products $1, ..., N$. If the choice is to produce a different product than $P(t)$, a set-up is performed. Let $U_i(t)$ be the number of set-ups for product $i$ initiated up to time $t$.

Three kinds of costs are considered:
- Each product item in stock incurs a holding cost $h_i$ per time unit (h).
- Each ordered but undelivered product item incurs a backlog cost $b_i$ per time unit.
- Each set-up to product $i$ incurs a cost $u_i$.

The average cost over time horizon $T$ is given by

$$\frac{1}{T} \sum_{i=1}^{N} \left( u_i U_i(T) + \int_{t=0}^{T} \left( h_i [X_i(t)]^+ + b_i [X_i(t)]^- \right) dt \right)$$

where $[x]^+ = \max(0, x)$ and $[x]^− = \min(0, x)$.

The goal is to find a control policy minimizing the expectation of average cost (1) over the long run, as $T \to \infty$. However, following Schulman et al. (2017) we run reinforcement learning with a discounted cost objective, to focus the learning algorithm on the near-future consequences of actions. We apply a discount factor of $\gamma = 0.9$ per hour.

The discounted cost over infinite time horizon is given by

$$\int_{t=0}^{\infty} \gamma^t \sum_{i=1}^{N} \left( h_i [X_i(t)]^+ + b_i [X_i(t)]^- + u_i \frac{d}{dt} U_i(t) \right) dt,$$

where the differential $\frac{d}{dt} U_i(t)$ is interpreted as a sum of Dirac delta functions centred at each $t$ where $U(t)$ increases, i.e. the times of set-up actions.

In the model, the inventory variables $X_i$ are limited to the range $X = \{-M, \ldots, M\}$ with $M = 500$. The initial state at $t = 0$ is set as follows: we set $P(0) = 1$ and the initial inventory levels are randomly picked from the uniform distribution on the range $\{-100, \ldots, 100\}$. The initial state is likely to be unfavourable, and a successful control method should be able to recover into a state with modest inventory levels.
3. REINFORCEMENT LEARNING APPROACH

3.1 Preliminaries

In the control method, we assume that the system is a semi-Markov decision process (SMDP); at the decision times \( t \), i.e., whenever a control decision is made, the future of the process depends only on the observable state variables \( P(t) \) and \( X_i(t) \), \( i = 1, \ldots, N \), and the control decision denoted by \( A(t) \). This assumption does not strictly hold in the simulation model, since the time to the next order arrival depends on the previous arrival time — of course robustness in the face of model inaccuracies is necessary for the control method to be applicable in practice.

We denote the observable state at time \( t \) by \( S(t) = (P(t), X_1(t), \ldots, X_N(t)) \), and the state system space by \( S = \mathbb{Z}_N \times \mathcal{X}^N \). In this case the cardinality of the state space is \( 3 \cdot 1001^3 \approx 3 \cdot 10^9 \). The available actions are denoted by \( A = \{0,1,\ldots,N\} \), where 0 represents leaving the machine idle, and other values represent specific products.

A stochastic control policy \( \pi \) associates an action probability \( \pi(s,a) \in [0,1] \) for each action \( a \in A \) in each state \( s \in S \).

The state-value function \( V(s) \) of a policy \( \pi \) is defined as the expected discounted reward — in our case the expectation of equation (2) negated — if the system starts from state \( S(0) = s \) and then runs under the control policy. Under the assumption that the system is an SMDP, the state-value function \( V(s) \) is independent of the starting time: when the system is in state \( S(t) = s \) at a decision time \( t \), the expected discounted cost from \( t \) to infinity equals \( V(s) \).

3.2 State-value approximation and policies

As the state system space is large, we approximate \( V(s) \) by a linear combination of basis functions. We represent the state by \( (p,x) \) where \( x \in \mathcal{X}^N \).

We use a tiled representation related to the one Löhndorf and Minner (2013) use for a similar problem. We split the range of inventory levels \( -M, \ldots, M \) at breakpoints \( \{m_{-1}, m_{-2}, \ldots, m_J\} \) symmetrically around \( m_0 = 0 \), so that on the positive side \( m_{j+1} = m_j + \max(4, [0.02m_j]) \).

Essentially, the intervals between breakpoints grow gradually from 4 around inventory level 0, toward 10 around inventory levels \( \pm 500 \). The range of the index \( j \) is \( \{-J, \ldots, J\} \) where \( J = 93 \). We define the basis functions

\[
\phi_j(x) = \begin{cases} 
    c_{m_j-x,0} & \text{if } m_j < 0 \text{ or } (m_j = 0 \land x \geq 0), \\
    c_{m_j-x,m_j-x} & \text{if } m_j > 0 \text{ or } (m_j = 0 \land x < 0), 
\end{cases}
\]

where \( c(x,a,b) = \max(a,\min(b,x)) \). See Figure 1 for illustrations of the functions \( \phi_{-2}, \ldots, \phi_2 \). The shape of the \( \phi_j \) functions is intended to make it easy to maintain unimodality of the approximated state-value function, i.e. that the value always decreases (cost increases) towards extreme inventory levels, even if only a subset of the weights are updated on a training iteration.

The first approximate state-value function is a linear combination

\[
V_0((p,x)) = \sum_{j=-J}^{J} \theta_{p,j} V_j(x),
\]

where the weight parameters \( \theta_{p,j} \) are indexed by the currently set-up product \( p \) in addition to \( j \); in essence, we train a separate function approximation for each product \( p \).

The second approximate state-value function we consider is a two-layer neural network

\[
V_p((p,x)) = \sum_{j=-J}^{J} \theta_{p,j} V_{p,j} + H \sum_{h=1}^{N} \theta_{p,h} \sum_{j=-J}^{J} \theta_{h,j} V_{j},
\]

where \( \theta = \theta_1, \theta_2, \theta_3 \) represents a different parameterization than in (4). Note that \( [+] \) is the well-known rectified linear unit (ReLU). We achieved best results with \( H = 8 \) connections between the two layers, per each product \( p \). Unlike (4), in (5) the approximation is not independent for each product \( p \), as one of the network layers is shared.

To apply a policy gradient approach, the action probabilities of a stochastic control policy are also defined as parameterized functions of the current state. The functions are structured identically with the state-value functions: e.g. with the state-value function (4) we use

\[
\pi_\theta((p,x),a) = \sum_{j=-J}^{J} \theta_{a,p,j} \phi_j(x),
\]

where the weight parameters \( \theta_{a,p,j} \) are specific to the action \( a \in A \) and independent of the parameters of (4).

With the state-value function (5), the parameters \( \theta_{h,j} \) of the inner layer are shared between both the state-value approximation and action probabilities, and all other parameters are independent. The overall structure of the neural network is illustrated in Figure 2. Without the shared layer, the network is equivalent to the linear model of equations (4) and (6).

In comparison, the first state-value approximation scheme of Löhndorf and Minner (2013) is simpler than ours: the basis functions are piecewise constant instead of piecewise linear, with the intervals between breakpoints constant. In their second state-value approximation scheme they consider piecewise constant functions indexed by subsets of the inventory levels, in particular pairs of inventory levels — in our notation \((p,x_i,x_j)\) for all pairs \(1 \leq i < j \leq N\). Both of their approximations are linear combinations of basis functions, unlike our nonlinear model (5).
Fig. 2. Structure of the neural network that produces the state-value approximation $V(s)$ and action probabilities $\pi(s, a)$ for $a \in \mathcal{A}$, for any given state $s = (p, x) \in \mathcal{S}$. The incoming arrows at each node are aggregated as a parameterized linear combination. The encoding layer generates values of the basis functions $\phi$. The shared layer outputs are passed through the ReLU function, indicated by $R$ in the figure. There is a separate output layer for each currently set-up product ($\times 3$) and each output variable ($\times 5$). The output selection nodes pick for each output variable the value corresponding to the currently set-up product.

3.3 Algorithm

We use the proximal policy optimization (PPO) algorithm of Schulman et al. (2017) with ad-hoc modifications for the semi-Markov setting. The algorithm seeks the optimal approximation parameters $\theta$ by iterating these steps:

1. Run $N_{\text{parallel}} = 8$ independent simulations from random initial states under the previous stochastic policy $\pi_{\text{old}}$ for a fixed amount of simulation time $T_{\text{rollout}} = 5$ d.
2. Repeat $N_{\text{opt}} = 4$ times: Optimize a surrogate objective $L^{PPO'}$ with respect to parameters $\theta$, by iterating the Adam stochastic gradient descent method of Kingma and Ba (2015) over each simulation result in batches of 64 simulation steps.
3. Set $\theta_{\text{old}} \leftarrow \theta$.

Each simulation run in step 1 of the algorithm is called a rollout. Let $K$ be the number of simulation steps in a rollout (typically 2000–3000 in our tests); note that $K$ is a random variable. A rollout is described by the following variables, indexed by simulation step $k = 1, \ldots, K$:

- $t_k$ Decision time
- $s_k$ Observed state $S(t_k)$
- $a_k$ Selected action $A(t_k)$
- $r_k$ Reward on $[t_k, t_{k+1})$, including any set-up at $t_k$
- $V_{\text{old}}(s_k)$ State-value approximation $V_{\text{old}}(s_k)$

Each simulation is run past the fixed end time $T_{\text{rollout}}$ until the next control decision is needed, at which time we observe $t_{K+1}, s_{K+1}$ and $v_{K+1}$. In addition, we use the shorthand $\tau_k = t_{k+1} - t_k$ for the elapsed simulation time over $l$ steps starting from state $s_k$.

The surrogate objective for step 2 of the algorithm is defined in parts, following the original PPO algorithm of Schulman et al. (2017). The key point is that we redefine the generalized advantage estimator of Schulman et al. (2016) for continuous-time discounting, with rather straightforward changes.

To begin from the basics, we use the discounted Bellman residual $\delta_k = r_k + \gamma^{\tau_k} v_{k+1} - v_k$, which indicates the advantage of choosing action $a_k$ and then following the policy matching the value function, as opposed to following the policy immediately. For longer action sequences we have the analogous $m$-step advantage estimator

$$
\hat{A}_k^{(m)} = \sum_{l=0}^{m-1} \gamma^{\tau_l} \delta_{k+l} = -v_k + r_k + \gamma^{\tau_k} r_{k+1} + \ldots + \gamma^{\tau_{K-l}} r_{k+m} + \gamma^{\tau_{K-m}} v_{k+m}.
$$

As in Schulman et al. (2016), with small $m$ we may suffer from high bias due to inaccuracies in the value approximation $v$, and with large $m$ we have high variance from the reward terms $r$ and durations $\tau_k$. The generalized advantage estimator of Schulman et al. allows a compromise between both kinds of estimates, where the trade-off is controlled by a parameter $\lambda$. We use $\lambda = 0.98$.

We define the generalized advantage estimator with continuous-time discounting as

$$
\hat{A}_k^{(\gamma, \lambda)} = (1 - \lambda) \left( \hat{A}_k^{(1)} + \lambda \hat{A}_k^{(2)} + \lambda^2 \hat{A}_k^{(3)} + \ldots \right)
$$

$$
= (1 - \lambda) \left( \delta_k (1 + \lambda + \ldots) + \gamma^{\tau_k} \delta_{k+1} (1 + \lambda^2 + \ldots) + \gamma^{2\tau_k} \delta_{k+2} (1 + \lambda^3 + \ldots) + \ldots \right)
$$

$$
= \sum_{l=0}^{\infty} \lambda^l \gamma^{\tau_k} \delta_{k+l}.
$$

From here on, we define objective function components essentially identically to Schulman et al. (2017). In particular, the components are defined using the empirical mean over simulation steps, denoted $E_k$, which is not weighted by simulation time. We found this to work significantly better than an empirical mean over simulation time.

The generalized advantage estimator is used in the clipped surrogate objective

$$
L^C(\theta) = E_k \left[ \hat{A}_k^{(\gamma, \lambda)} \min \left( \rho_k(\theta), c(\rho_k(\theta), 1-\epsilon, 1+\epsilon) \right) \right]
$$

where the probability ratio $\rho_k(\theta) = \pi_\theta(s_k, a_k)/\pi_{\text{old}}(s_k, a_k)$ compensates for sampling under the old policy $\pi_{\text{old}}$ instead of $\pi_\theta$, and $\epsilon = 0.2$ is a regularization parameter.

The value loss objective penalizes errors in the state-value approximation, and is defined by

$$
L^{VF}(\theta) = E_k \left[ \left( V_{\theta}(s_k) - \gamma \sum_{l=0}^{K-k} \gamma^{\tau_{k+l}} r_{k+l} \right)^2 \right].
$$

Finally, the modified PPO surrogate objective in step 2 of the algorithm is defined as

$$
L^{PPO'} = L_k^C(\theta) - c_1 L_k^{VF}(\theta),
$$

where we use $c_1 = 10^{-4}$ as a weight coefficient. In contrast with the original PPO algorithm, we do not include an
entropy term, since we found it ineffective in practice, and sufficient policy exploration was accomplished without it.

In addition to the surrogate objective (11), we keep track of the average reward rate during training. We find the 100-rollout interval with the best average reward rate, and finally output $\theta$ from the last rollout of that interval.

4. EXPERIMENTS

4.1 Procedure

We implemented the simulation model and the controller in the Python programming language. The controller is based on the implementation of the PPO algorithm of Schulman et al. (2017) from DeepChem (2016), which we modified to work in continuous time as in Section 3.3.

In all reported tests the modified PPO algorithm is run for 10 000 000 training steps (controller decisions), corresponding to approx. 4000 rollouts. As the algorithm is stochastic, we repeated the training 11 times, and report the results of the best training run and the median run. The learning rate of the Adam algorithm was set at $10^{-4}$.

We present results for the following policies generated by reinforcement learning:

- **PPO1.best, PPO1.median**: The best and median policy found in 11 runs of the modified PPO algorithm, using the linear model of equations (4) and (6) for the state value and action probabilities.
- **PPO2.best, PPO2.median**: The best and median policy found in 11 runs of the modified PPO algorithm, using the two-layer neural network of figure 2 for both state-value and action probabilities.
- **PAD2005**: The best policy reported by Paternina-Arboleda and Das (2005), in which the product sequencing decisions of a base-stock policy were learned by a multi-agent reinforcement learning algorithm, and the base-stock levels 45, 15 and 20 were determined by exhaustive search. The standard deviation is derived from the 95% confidence interval 0.61 reported by Paternina-Arboleda and Das.

For validation purposes we have also reimplemented:

- **ABAC**: The base-stock policy determined by Anupindi and Tayur (1998) using infinitesimal perturbation analysis on a continuous model. For our discrete simulation we round the base stock levels to the nearest integers 49, 19, and 24. The product is switched in the regular sequence 1-2-1-3.

Each policy was evaluated in 30 replicated test runs, in order to determine the effect of random customer demand on performance. Each test run starts from a random initial state, runs for a warm-up period of 100 days of simulation time, and is then recorded for 400 days of simulation time. This policy evaluation procedure is from Paternina-Arboleda and Das (2005).

4.2 Results

The performance of our PPO1 and PPO2 policies, and the two policies from literature is summarized in table 2: the PPO2.best policy provides a 2% improvement over the average cost rate of the best policy of Paternina-Arboleda and Das (2005). The global optimum value is not known.

We note that the average cost rate of the ABAC policy closely matches the cost rate 100.85 reported by Anupindi and Tayur (1998), and the cost rate 100.75 of a reimplementation by Paternina-Arboleda and Das (2005). Thus the model implementations appear to be substantially equivalent. On the other hand, Paternina-Arboleda and Das report 95% confidence intervals for both the ABAC policy and the PAD2005 policy that are even smaller than the sample standard deviations in our tests—it is unclear whether this is a difference in the simulation models, in the control policy, or in the evaluation procedure.

There was significant variation between training runs of the PPO algorithm. The worst training runs of the PPO algorithm resulted in brittle policies that would work at a competitive level much of the time, but then in 1-5 out of 30 test runs the policies would let the order backlog fill up, and fail to recover. Such failures occurred on 3 out of 11 training runs with PPO1, and 2 out of 11 training runs with PPO2. The progress of the training in the best and the worst PPO2 case is illustrated in Figure 3. While the best training run shows significant variation in the cost rate and two periods of very bad cost rate after the initial phase, the worst training run does appear to produce bad cost rates more regularly. The overall progress is nevertheless similar, with the very best results reached after approx. 5 million simulation steps.

4.3 Findings

The lot scheduling application we selected appears deceptively simple, and although there are good heuristic control policies available, finding a competitive control policy by reinforcement learning remains challenging. One must acquire a good understanding of the internal operation of the algorithms to analyze learning failures and to adjust hyperparameters.

Compared to the earlier reinforcement learning solution by Paternina-Arboleda and Das (2005), we did manage to find a better policy using a more general approach. Instead of splitting the task between multiple agents with ad-hoc cost modifications like Paternina-Arboleda and Das (2005), we needed to work to find a good neural network architecture and hyperparameter values. We also needed several orders of magnitude more training iterations.

Deep reinforcement learning requires a suitable neural network architecture, and it is not clear how to devise one. We experimented with a large number of architectures, but the two-layer architecture of figure 2 is the only successful one with more than one layer: deepening the

<table>
<thead>
<tr>
<th>Method</th>
<th>Average cost rate ($/h)</th>
<th>Standard dev. ($/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPO1.median</td>
<td>102.4</td>
<td>3.2</td>
</tr>
<tr>
<td>ABAC</td>
<td>100.3</td>
<td>1.9</td>
</tr>
<tr>
<td>PPO1.best</td>
<td>96.1</td>
<td>2.9</td>
</tr>
<tr>
<td>PPO2.median</td>
<td>94.7</td>
<td>2.4</td>
</tr>
<tr>
<td>PAD2005</td>
<td>93.58</td>
<td>0.31</td>
</tr>
<tr>
<td>PPO2.best</td>
<td>91.3</td>
<td>2.8</td>
</tr>
</tbody>
</table>
structure led to only worse results. Perhaps there are neural network structures yet to be discovered which would suit manufacturing systems well, in a similar manner to how convolutional neural networks are successful in vision applications. Also, with sufficient computing power one could experiment with automated architecture search.

Reaching the final results required $11 \cdot 10^7$ training iterations. Each training run required approx. 20 h of real time on an 8-core Intel Xeon, though we did not particularly focus on training performance. In any case, the large number of training iterations means that a good simulation model of the manufacturing system is absolutely necessary — applying our reinforcement learning approach directly to a similar real-world system is not feasible.

Regardless of difficulties, our experimental application was successful. Reinforcement learning remains a promising general approach to optimal control, which for now requires research-level experimentation to apply. At present, it is best suited to applications where no existing control method is satisfactory. It may yet become more generally applicable, if research continues to make progress on more robust reinforcement learning algorithms.

REFERENCES


