Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

USE OF SIMULATION-AIDED REINFORCEMENT LEARNING FOR OPTIMAL SCHEDULING OF OPERATIONS IN INDUSTRIAL PLANTS

Satyavrat Wagle
Aditya A. Paranjape

Software Systems and Services Research Area
TCS Research
Tata Consultancy Services Limited
Pune, 411013, INDIA

ABSTRACT

In this paper, we present an algorithm based on reinforcement learning for scheduling the operations of an industrial plant which is modeled as a network of machines on a directed acyclic graph. The algorithm is assumed to have access to a high-fidelity simulator of the plant, but not a mathematical model. The algorithm is designed to optimize an objective function over a moving window, similar to receding horizon control, for a typical industrial plant which converts raw material into finished products. The delivery schedule for the incoming raw material is assumed to be known but subject to uncertainty. A novel feature of our technique is the use of *schedule moments* to train the algorithm to handle a large class of incoming delivery schedules.

1 INTRODUCTION

In this paper, we present an algorithm for scheduling the operations of an industrial plant consisting of a network of machines. We restrict our technique to networks which are modeled as directed acyclic graphs (DAGs). Raw material enters the "root" of this DAG, and the finished products leave at the "leaves." The schedule of the incoming raw material is known with some uncertainty (in a sense that will be made precise later). The objective of this paper is to schedule the processing of the raw material over a prescribed time window such that the net productivity of the plant is maximized, measured typically as the net profit.

It is assumed that the scheduling algorithm has access to only a high-fidelity simulator of the plant, rather than a mathematical model of the plant. This situation is actually representative of real-world industrial operations. When the physical mathematical models of the individual processes are taken into account, the plant operations form a dynamical system with extremely large orders. It is well-known that designing controllers for such systems is extremely challenging. Moreover, the control inputs are vastly diverse, covering the transport rate of material between individual machines or processors on the one hand and low-level operating parameters of individual machines or processors on the other. As a result, it is not uncommon for operating schedules to be prepared by highly experienced and skilled individuals, often using low-fidelity models and linear programming (LP) algorithms as a first iterate. These schedules are then certified using high-fidelity simulators.

Our algorithm is intended to play the role of a stand-alone scheduler in its own right, or act as a high-performance recommender system to human schedulers.

1.1 Prior Work

The problem considered in the paper shares common ground with well-known problems in decision theory, such as supply chain optimization, flow-shop optimization (Pinedo 2012) and dynamic resource allocation problems. Applications include manufacturing plants, chemical refineries, defence resource allocation (Bertsekas, Homer, Logan, Patek, and Sandell 2000) and even epidemics (Brandeau, Zaric, and Richter 2003). Most of the techniques developed for these problems, especially at a theoretically rigorous level, deal with the time of completion of a process (the makespan). When the plant is poorly characterized or too complex to control analytically, approaches based on approximate dynamic programming, or reinforcement learning, are invoked (Zhang and Dietterich 1995; Zhang and Dietterich 1996).

In contrast to problems dealing with makespan, we are interested in problems that deal with reward (defined in a suitable sense) maximization. Although LP-based techniques (Pinto, Joly, and Moro 2000; Mendez, Grossmann, Harjunkoski, and Kaboré 2006) as well as model predictive control (Al-Othman, Lababidi, Alatiqi, and Al-Shayji 2008; Brandeau, Zaric, and Richter 2003) have been applied successfully to such problems, these techniques rely on knowing the plant model reasonably accurately (if not exactly). In our setting, on the contrary, we assume that the plant model is not known and needs to be learned in a suitable sense through trial-and-error. This places our problem firmly in the remit of approximate dynamic programming, or reinforcement learning (RL). RL-based techniques have been proposed for scheduling problems in the context of the usual makespan minimization (Zhang and Dietterich 1995; Zhang and Dietterich 1996) and even missile defence systems (Bertsekas, Homer, Logan, Patek, and Sandell 2000).

Model predictive control (MPC) is a widely-used control technique for industrial plant operations. Although MPC and RL solve the same problem, the two differ at a technical level in terms of the nature of their policy: MPC provides, by design, an *open-loop* control signal while RL provides a closed-loop (stationary) *policy* (Ernst, Glavic, Capitanescu, and Wehenkel 2009). Recent attempts to bring together MPC and learning have involved either learning an explicit system model that can be exploited by MPC (Williams, Wagener, Goldfain, Drews, Rehg, Boots, and Theodorou 2017), or using MPC to generate training sets for neural network-based policy generators which operate at a fraction of the computational cost of MPC (Zhang, Kahn, Levine, and Abbeel 2016). RL techniques can be used to augment typical MPC algorithms (Negenborn, De Schutter, Wiering, and Hellendoorn 2005), wherein MPC is used (on an approximate model) until the value function estimates required by RL are sufficiently reliable. Thereafter, the computationally simpler RL-generated policy can replace the one generated by MPC.

1.2 Contribution

We develop a reinforcement learning (RL)-based controller for designing flow-shop schedules that maximize a prescribed objective function. We remind the reader that the need for RL stems primarily from the constraint that we have access only to a reliable (and possibly high-fidelity) simulator of the plant rather than an explicit model. A key requirement is that the controller be able to work with a large class of raw material receipt schedules. In order to meet this requirement, we train the controller using a handful of temporal moments of schedules instead of actual schedules. This reduces the burden of training considerably, while adding robustness to the controller against off-design schedules. Although moments have been used in other areas to represent patterns (Prokop and Reeves 1992), this is the first instant, to the best of our knowledge, where the concept has been invoked reduce the dimensionality of the training set for learning algorithms.

The rest of the paper is organized as follows. We present the broad problem formulation in Sec. 2. We present the RL algorithm in Sec. 3, including the formulation of the temporal moments. In Sec. 4, we present the model which forms the basis of the simulation example. The results of the simulation, as well as the accompanying discussion, are presented in Sec. 5.

2 PROBLEM FORMULATION

A typical industrial plant consists of machines, or components, connected through transfer channels. We are interested in the case where the machines and the transfer channels together defined a directed acyclic graph (DAG), with the machines acting as its nodes and the transfer channels as its edges. The root nodes, in particular, are areas where the raw material is received, while the leaf nodes are those from which finished products are dispatched.

Let $u_i[k]$ denote the state of the i^{th} edge (transfer channel) at the k^{th} time instant, and let $s_j[k]$ denote state variable of the j^{th} node at that time. The state variables would typically measure the inventory levels at a machine, but other state variables may be defined, depending on the problem. We assume that the state changes per a deterministic Markov process of the form

$$s[k+1] = F(s[k], u[k]) \tag{1}$$

where $s[k]$ may include exogenous states with possibly unknown (hidden) dynamics. The plant incurs costs for processing the raw material at each stage as well as for running the transfer channels. The plant earns a revenue only against the value of the dispatched product. More specifically, we define the operating cost at the k^{th} instant as

$$c[k] = \alpha_r(u[k], s[k]) \tag{2}$$

where $u[k]$ is the vector whose elements are $u_i[k]$ and $s[k]$ is defined likewise. The function $\alpha_r : (u,s) \mapsto \mathbb{R}^+$. The cost $c[k]$ might additionally include costs from running machines with excess or inadequate raw material, which is captured in the dependence of α_0 on $s[k]$.

Let $d[k]$ denote the quantity of product that is shipped out. This is simultaneously a source of revenue $\rho(d[k])$ as well as cost $\alpha_s[d[k])$. We define the net profit at time k as

$$p[k] = \rho(d[k]) - c[k] - \alpha_s(d[k]) \tag{3}$$

Definition 1 (Constraints) We express the constraints compactly as $C(u,s) \leq 0$. Each individual constraint is of the form $f_i(s[k], u[k]) \leq 0$ falls into one of the following categories:

1. The holding capacity of a node is bounded above (and, as in chemical plants, also from below)
2. The carrying capacity of an edge is bounded
3. The number of incoming/outgoing edges that may be active simultaneously at a node

For all practical purposes, it may be assumed that the industrial plant runs for an infinite period of time. Industrial plants are run in accordance with business decisions, which include schedules for the finished products as well as the raw material. Since both of these schedules are available only for a finite horizon of time, it makes sense to design optimization and control algorithms over a moving time window. This leads us to the well-known philosophy of model predictive control (MPC).

Problem 1 (Control problem at time $t = 0$) Let $\tau_h > 0$ denote an appropriately chosen time horizon. The control objective at time $t = 0$ is to find $u^*[k]$, $k \in [0, \tau_h]$, which satisfies

$$u^*[0 : \tau_h] = \arg \max_{u[0:\tau_h]} \sum_{k=0}^{T} \gamma^k p[t+k], \text{ s.t. } C(u,s) \leq 0 \tag{4}$$

where $\gamma \in (0, 1]$ is a suitably chosen discount factor.

The policy computed at $t = 0$ is used only for the time window $[0, \tau_w]$, where $\tau_w < \tau_h$, and a new policy is computed for subsequent implementation. This policy is designed as follows.

Let t denote an instant in time when a control policy is to be computed for the time window $[t, t+\tau_w]$. The control actions computed at time $t - \tau_w$ for the time interval $[t, t+\tau_h-\tau_w]$ are available to the controller

and used as a baseline to design the control signal for the time window $[t, t+\tau_w]$. We denote these baseline control values by $u_{\text{ref}}[t+k]$ for $k = 0, \ldots, \tau_h - \tau_w$. We expect the policy obtained at time t to stay close to u_{ref}, in the spirit of a regularity condition.

Problem 2 (Control problem at time $t > 0$) Let $\tau_h > 0$ denote an appropriately chosen time horizon. The control objective at time t is to find $u^*[t+k]$, $k \in [0, \tau_h]$, which satisfies

$$u^*[t : t+\tau_h] = \arg \max_{u[t:t+\tau_h]} \sum_{k=0}^{\tau_h} \left(\gamma^k p[t+k] + \gamma_u^k[t+k] \|\Delta u[t+k]\|^2 \right), \text{ s.t. } C(u,s) \leq 0 \tag{5}$$

where $\gamma, \gamma_u \in (0, 1]$ are suitably chosen discount factors and $\Delta u = u - u_{\text{ref}}$. The values of γ_u are chosen so that $\gamma_u[t+\kappa] = 0$ for $\kappa > \tau_h - \tau_w$.

It is evident that solving Problem 1 at $t = 0$ and Problem 2 at future instants of time yields a control law which can be used, at least in principle, for the entire lifetime of the plant.

It is well-known, however, that the plant model needs to be known well in order to solve optimal control problems. When the plant model is not available, it becomes essential to invoke a control law which works exclusively with the available state information and short-term forecasts. Controllers based on reinforcement learning achieve precisely as much, although these controllers need to be trained on (reliable and high-fidelity) plant models ahead of time in order to ensure safe and reliable operation.

3 REINFORCEMENT LEARNING-BASED SCHEDULER

In this section, we design a reinforcement learning algorithm for Problem 2 and note that Problem 1 can be viewed as a special case with $\gamma_u \equiv 0$. We start by describing a moment-based representation for input schedules, and then describe the RL algorithm.

3.1 Moment-Based Representation of Receipt Schedules

The policy determined by the RL algorithm would depend on the expected receipt of raw material as well as the demand for finished products. These schedules may be viewed as hidden states within $s[k]$ of the system (1).

While it might be beneficial, in principle, to optimize the controller for as large a set of receipt/demand schedules as possible, it is not expedient to do as much for *every* conceivable schedule. As an illustration of the complexity of doing so, for a simple case involving just two receipt channels, each of which is either on or off, there are 4^{τ_w} possible schedules (at each instance, the input belongs to the set $\{00, 01, 10, 11\}$), where τ_w is the horizon for which the control signal is to be designed. In order to train the controller, it is essential to seek a balance between exposing it to a large number of schedules on the one hand, and managing the number of training runs on the other.

We seek this balance by introducing a state m whose elements are the *temporal moments* of a candidate schedule. For illustration, we restrict ourselves to just the receipt schedule (assuming that there is no defined demand schedule for the finished product, although this can be absorbed readily into our framework). More specifically, at the beginning of each instant when the control inputs are computed, we compute the moments

$$m_j^i = \begin{cases} \sum_{t=1}^{\tau_w} r_i[t], & j = 0 \\ \dfrac{\sum_{t=1}^{\tau_w} t^j r_i[t]}{\tau_w^j \sum_{t=1}^{\tau_w} r_i[t]} & \text{otherwise} \end{cases} \qquad i \in \{1, 2\} \tag{6}$$

where τ_w is the horizon over which the control inputs are computed. The term m_0^i yields the total amount of incoming raw material for the i^{th} channel. The first moment m_1^i yields the time instant by which half the amount of raw material would have been received. Higher order moments provide more nuanced

information about raw material receipt, and any given schedule can be represented *exactly* in terms of the first T moments.

As an illustration, consider an on-off-type receipt schedule on a single channel and spanning five units of time. The two receipts $[0,1,0,1,0]$ and $[1,0,0,0,1]$ have the same values for the zeroth and first moments (2 and 0.6, respectively). The second moments differ, and are given by $13/25$ and $2/5$, respectively.

To minimize the amount of computation required for training, and to ensure that the scheduler is able to work with a large number of schedules, we add the first three moments (i.e., $j = 0,1,2$) to the state of the system. This necessarily makes the scheduler slightly sub-optimal, but it also makes the scheduler robust to an off-design receipt schedule.

3.2 The RL Algorithm

We start by defining the value function at time t in terms of the state s_t as

$$V[t] \equiv V(s[t]) = \sum_{k=0}^{T-t-1} \gamma^k p[t+k], \tag{7}$$

Recall that an optimum trajectory (s^*, u^*) satisfies the Bellman equation

$$V(s^*,u^*) = \max_u (p(s^*,u) + \gamma V(s_n^*, u_n^*)) \tag{8}$$

where s_n denotes the successor state to s, and likewise for u_n. The corresponding map $\pi^* : s \mapsto u^*(s)$ is the optimal policy.

Our approach for solving the Bellman equation and determining the optimal control input uses a combination of Monte-Carlo (MC) simulation and batch actor-critic approach. For background information about these techniques, the reader may refer to (Sutton and Barto 2018) (see Ch. 5 and Sec. 13.5). We use a neural network (NN)-based function approximator for encoding the value function as well as the policy. We implement two sets of deep neural networks (DNNs) as part of our architecture, as illustrated in Fig. 1. The *estimator network* approximates $V(s)$. It consists of two individual DNNs for approximating the reward and the cost, respectively. The outputs of these individual DNNs yields the predicted value of $V(s)$. The policy network encodes the policy. Its output is a set of probability-like values assigned to individual actions. The actual action is chosen through a soft-max action over the probabilities.

The pseudo-code for our algorithm is given in Algorithm 1. It iterates over the steps described in the subsections below.

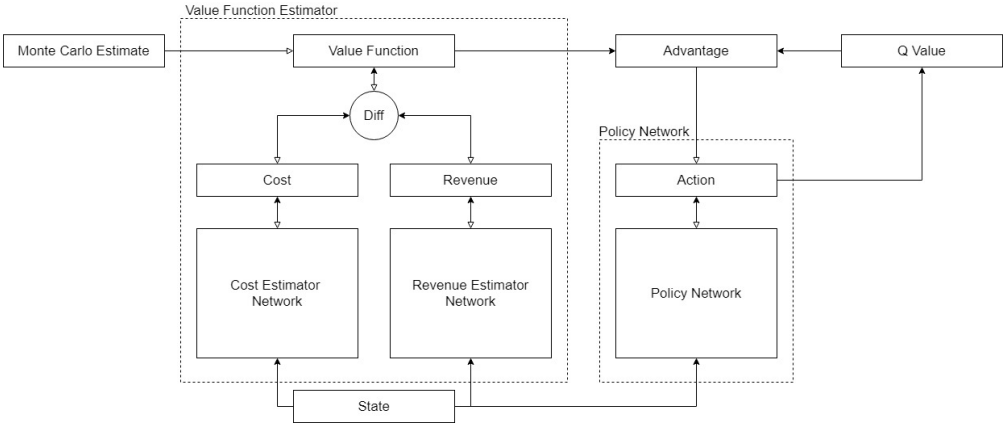


Figure 1: The NN-based architecture for implementing the RL algorithm.

Algorithm 1 RL algorithm for training the scheduler

Initialize: V_0, π_0, n_m (number of moments)
Max number of iterations, $k_{\max}$, and episodes $E_{\max}$
while Algorithm converged or $k > k_{\max}$ **do**
  for Episode number $= 1 : E_{\max}$ **do**
    Generate a random receipt schedule and compute $m = m^{1,2}_{0:n_m}$
    Run a simulation with policy $\pi^{\varepsilon}_k(s)$ with $s[0]$ drawn randomly
    For each visited s, with time of first visit τ_s, append reward $\sum_{k=\tau_s}^{T} \gamma^{k-\tau_s} p[k]$ to $R(s)$
    For each visited (s,u), with time of first visit τ_s, append reward $\sum_{k=\tau_s}^{T} \gamma^{k-\tau_s} p[k]$ to $R^{(u)}(s)$
  end for
  for each visited s **do**
    $V_{\text{eval}}(s) = \text{average}(R(s))$; $V_{\text{pred}}(s) = V_k(s)$
  end for
  for each visited state-action pair (s,u) **do**
    $Q_{\text{eval}}(s,u) = \text{average}(R^{(u)}(s))$
    Draw N_p random samples $a \sim \pi^{\varepsilon}(s)$
    for Each drawn pair (s,a) **do**
      Perform one-step simulation and calculate $Q(s,a) = p(s,a) + \gamma V_{\text{pred}}(s')$
      Compute the loss $Q(s,a) - Q_{\text{eval}}(s,u)$ for the N_p samples
    end for
  end for
  Update Estimator Network and Policy Network
  Update iteration number: $k \leftarrow k+1$
end while
Output: converged policy $\pi^*(s)$

3.2.1 Off-Policy MC-Based Policy Evaluation

We run a batch with a fixed number of episodes and randomly chosen initial conditions. Each episode lasts $T \geq \tau_w$ units of time. The episodes could be run with either a fixed policy across all episodes or one may use different policies for different episodes. Let π denote the policy being evaluated (i.e., the current iterate of the policy). Its ε-greedy version, denoted by π^{ε}, assigns the following probability of choosing an action a

$$\phi^{\varepsilon}_a(s) = \frac{\varepsilon}{|\mathscr{A}|} + (1-\varepsilon)\phi_a(s)$$

where $\phi_a(s)$ is the probability of choosing the same action under the original policy and $|\mathscr{A}|$ is the total number of actions. The ε-greedy approach would ensure that actions with low probabilities are explored adequately, particularly if the initialization of the policy network is biased towards actions that may eventually turn out to be unsuitable.

We calculate the average value of $V(s)$ over N episodes in a batch *for each visited* state s and each visited pair $(s.u)$ using

$$V_{\text{eval}}(s) = \frac{\text{sum}(R(s))}{\sum_{i=1}^{N} \delta_{is}},\ Q_{\text{eval}}(s,u) = \frac{\text{sum}(R^{(u)}(s))}{\sum_{i=1}^{N} \delta_{i(s,u)}},\quad s \text{ visited} \tag{9}$$

where $R(s)$ is the vector of first-visit rewards for s, and $\text{sum}(R(s))$ is the sum of the elements of $R(s)$. Finally, $\delta_{is} = 1$ if s is visited during that episode. The terms $R^{(u)}(s,u)$ and $\delta_{i(s,u)}$ are defined similarly.

3.2.2 Error Calculation and Network Update

We update the NNs at end of each batch, requiring that the weights be updated to minimize the mean square error (MSE) of a suitably chosen loss term. The MSE for the estimator network is calculated for each state using

$$\delta_\theta = \frac{1}{N_s}\sum_{n=1}^{N_s}(V_{\text{pred}}(s) - V_{\text{eval}}(s))^2 \tag{10}$$

where N_s is the total number of visited states and $V_{\text{pred}}(s)$ is the estimate produced by the network.

The MSE loss for the policy network is calculated as follows. For each visited state-action pair (s,u), we conduct N_p *one-step* simulations, where N_p is a suitably chosen number. Each one-step simulation features an action drawn independently from π^ε for that simulation. We determine the Q function

$$Q(s,a) = p(s,a) + \gamma V_{\text{pred}}(s'), \quad a \sim \pi^\varepsilon(s) \tag{11}$$

where s' is the successor state found using simulation. We define the loss function for s as

$$\delta_\phi = \frac{1}{|(s,u)|}\sum_{(s,u)}\frac{1}{N_p}\sum_{n=1}^{N_p}(Q(s,a) - Q_{\text{eval}}(s,u))^2 \tag{12}$$

where Q_{eval} was defined in Eq. (9). Using the error $\delta_\theta(s)$ and $\delta_\phi(s)$ calculated above, the estimator and the policy networks are updated at the end of each batch.

3.3 Integration with MPC

The RL algorithm in Algo 1 was designed to solve Problem 2 and, by extension, Prob. 1. An MPC algorithm would use the converged policy π^* to obtain the control $u[t : t+\tau_h]$ at each instant t when the control problem is to be solved. It should be noted that this reduces the computational demand on MPC substantially, along similar lines as (Zhang, Kahn, Levine, and Abbeel 2016; Williams, Wagener, Goldfain, Drews, Rehg, Boots, and Theodorou 2017).

4 SIMULATION EXAMPLE: MODEL

We illustrate the technique described above on a fictitious but representative model of an industrial plant. In this section, we will describe the plant model and recall that the controller has access to just its simulator.

4.1 Plant Model for the Simulator

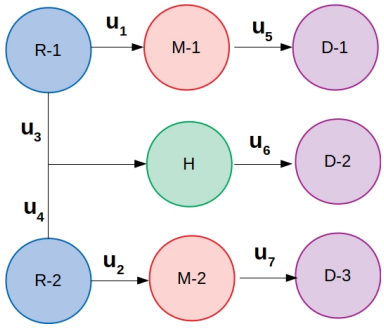


Figure 2: The simplified plant model used in the sequel.

The plant model, shown in Fig. 2 consists of four areas:

1. Holding areas for the raw material (R-1 and R-2)
2. Machine for processing pure raw material (M-1 and M-2)
3. Machine for processing mixed raw material (H)
4. Final value addition and ship-out (D-1, D-2 and D-3)

Each area can be viewed as a node on a directed graph. The edges between these nodes represent transfer channels (e.g., conveyor belts or vehicles). The weight of an edge indicates whether it is closed (0) or open (1). Certain edge weight combinations are infeasible, and these denote incompatible operations. For instance, a machine may not be able to accept raw material and dispense a finished product simultaneously. In the set up of Fig. 2, we set the constraints as $u_1u_5 = u_2u_7 = 0$, and $u_6 \times \max\{u_3, u_4\} = 0$.

In the context of this paper, scheduling involves balancing the flow of material between the machines. We assume that each area is capable of holding $S_{\max}$ units of material (raw or processed). Let the actual units held by the units be denoted by s_1, s_2 (R-1 and R-2), s_3, s_4 (M-1 and M-2), s_5 (H), s_6, s_7 and s_8 (D-1 to D-3).

The receiving areas receive raw material in discrete quantities at any given time. We denote the received amount at a time k by $r_i[k] \in \{0, r_{\max}\}$. The ship-out areas dispatch finished product in discrete quantities $d_i[k] \in \{0, \dots d_{\max}\}$. We assume that $d_{\max} = S_{\max}$ in this paper.

With this notation, we write the dynamical equation for the "mass flow" as

$$\begin{aligned} s_1[k+1] &= s_1[k] + r_1[k] - u_1[k] - u_3[k] \quad \text{and likewise for } s_2 \\ s_3[k+1] &= s_3[k] + u_1[k] - u_5[k] \quad \text{and likewise for } s_4 \text{ and } s_5 \\ s_6[k+1] &= s_6[k] + u_5[k] - d_1[k] \quad \text{and likewise for } s_7 \text{ and } s_8 \end{aligned} \tag{13}$$

where the states are bounded below by zero. In practice, this would have to be enforced by bounding the out-flow from each area. Since our intention is to create a simulator for training an RL-based optimizer, we achieve this bound as part of our control policy by imposing a penalty on the RL algorithm.

4.2 Revenue, Costs and Objective Function

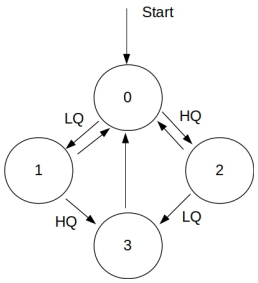


Figure 3: Transition of the flag representing the contamination in the hybrid machine. A flag of 0 indicates a clean, fresh machine.

The revenue earned by the plant is a function of the output. The values are shown in Fig. 2. In particular, the revenue from the mixed raw material machine depends on the nature of the material processed by it. The revenue is 1 per unit if it is entirely low quality, 1.8 if it is pure, and 1.2 otherwise. The state of the "contamination" in the hybrid machine is captured by a flag, and its transition takes place as per Fig. 3.

The costs accrued by the plant cover

- Cost of the raw material: the low quality and the high quality raw material cost, respectively.
- Cost of running the machines and the cost of running the transition channels: the cost is of sending a unit of material through a channel is assumed to be $\alpha u[k]$ where $\alpha = 1$ is a constant. This cost

is assumed to include part of the cost of running the machines up-stream and down-stream of the channel.
- Cost of shipping out the products: the cost of shipping out a quantity s is a nonlinear function of s, and assumed to be of the form $\beta_0 + \beta_1(1 - e^{-\beta_2 s})$. This captures the decreasing marginal cost of shipping out goods in increasing quantities.
- Overflow costs: if the net material in an area exceeds a threshold capacity, the excess must be discarded and it results in a cost proportional to the overflow.

Using the models for revenue and cost, it is straight-forward to calculate the profit accrued at each instant in time, denoted by $p[k]$. The objective of the controller is to maximize the total discounted profit over a moving window of length τ_h, given by

$$\sum_{k=0}^{\tau_h} \gamma^k p[t+k], \quad \gamma \in (0,1]$$

We set $\gamma = 1$ for the simulations, considering that the time horizon considered there (12 units of time) is not very large. The value of γ needs to be set to account for the uncertainty in the known receipt and demand schedules.

4.3 Challenges in Controlling the Plant Model

It should be recalled that the plant model is not known to the controller. However, it is instructive to understand fundamental limitations concerning the receipt schedule. We state the following results without proof, while noting that the proofs involve straight-forward book-keeping.

1. Continuous, persistent receipt on precisely one channel can be managed trivially by switching between machines 1 and 3, with the minimal switching corresponding to allowing the two machines to saturate their holding areas serially.
2. Continuous, persistent receipt on both input channels leads to overflow after no more than 10 time instants if the receipt areas (R-1 and R-2) are not allowed to hold any raw material. The overflow time is finite (i.e., not infinite) even if holding is permitted.
3. With continuous, persistent receipt on both channels and no holding in R-1 and R-2, the three machines M-1, M-2 and M-3 will process at an average rate of 5 per 4 time instants. As a result, we posit that the rate of entry of raw material at a given channel does not exceed, on average, 80% of the time.

The aforementioned properties indicate that scheduling the operations on the example plant is not a trivial problem. Moreover, an intense receipt schedule requires careful scheduling to prevent expensive overflows.

5 SIMULATION EXAMPLE: RESULTS

Since the key contribution of RL, in the context of MPC, is solving for a given time horizon (of duration τ_h), we focus our attention attention on a single time horizon. In effect, we demonstrate our algorithm on Prob. 1 for the model in the previous section.

5.1 Training of the RL Algorithm

For the particular simulation example considered here, we design the neural networks as follows. The cost estimator network and the revenue estimator network have four layers each, with 64, 64, 32 and 64, 32, 32 neurons respectively. All of the DNNs in the system utilize leaky rectified linear units (LRLUs) as activation functions apart from the last layer of the policy network which is a softmax layer. Both networks are optimized using an Adam optimizer with a learning rate of $1e^{-6}$.

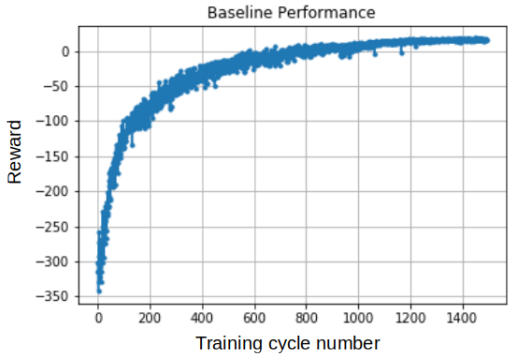
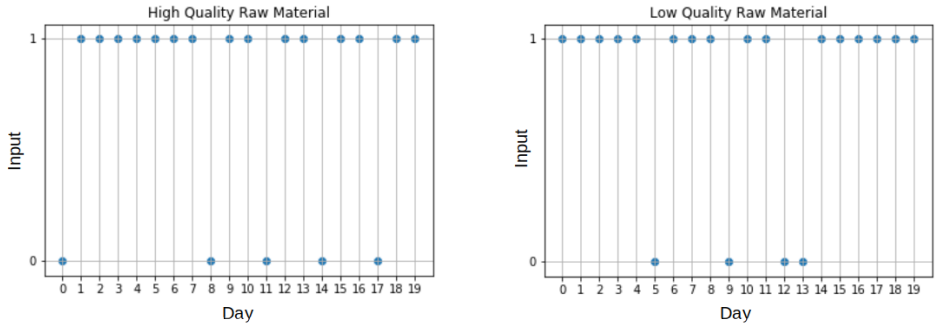


Figure 4: Baseline reward as a function of the number of training cycles.

We plot the results of a sample training run in Fig. 4. The plot shows the baseline performance ($V(s[0])$) averaged over nearly 400 training batches. The policy chosen (arbitrarily) at the start of the training leads to a large number of overflows, which reflects in the large negative value of V in the initial episodes. The subsequent policies avoid actions that lead to overflows, and the value function improves and converges after nearly 1400 training epochs to a value of around 15 units. We demonstrate the performance of the trained policy by comparing it with a raw policy obtained mid-way during training. While Fig. 4 would lead us to expect a better performance, it is worth examining the factors that permit improvement or, equivalently, are likely to cause a raw policy to fail.

5.2 Simulation



(a) High quality raw material

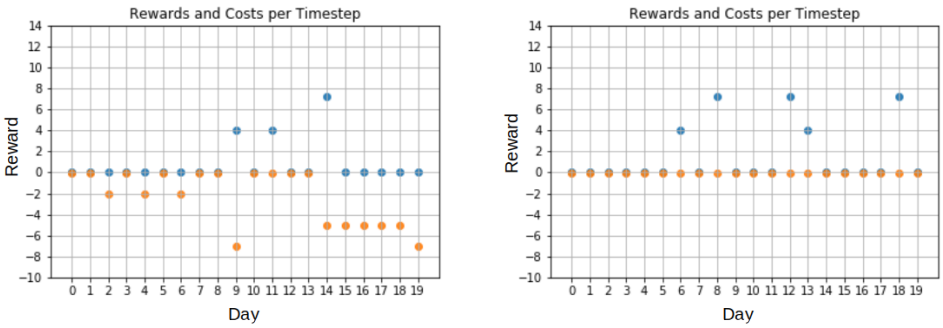
(b) Low quality raw material

Figure 5: Input receipt schedules of the raw material used for the simulation examples. Blue dot indicates whether or not the raw material was received on a given day.

Figure 5 shows the input receipt schedules used for the simulation examples. We notice that the amount of received raw material is close to the point where we expect problems to occur for an ill-designed policy, as explained in Sec. 4.3. Figure 6 compares the time histories of the reward and the overflow penalty for the two policies. We make two observations about the notation:

1. Since a reward is obtained only against a ship-out, the step rewards are zero at most time instants.
2. Likewise, the overflow penalty is zero unless one of the areas experiences an overflow.

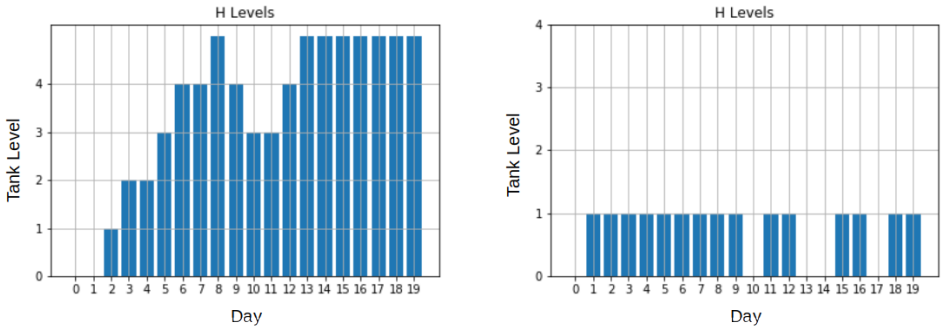
It is evident that the raw penalty incurs a large amount of overflow penalty. This is explained by the level of inventory in the area H (see Fig. 2) shown in Fig. 7.



(a) Raw policy (b) Optimized policy

Figure 6: Reward (blue dot) and overflow penalty (orange dot) incurred by the raw and the optimized policies on each of the 20 days of an episode.

The raw policy causes machine H to overflow, and the resulting overflow penalty reflects in the spikes in the plot showing the instantaneous costs. The optimized, trained policy keeps the inventory levels to manageable levels. Notice that area H is barely close to full for an optimized policy, which seems to run counter to the suggestion in Sec. 4.3 about the challenge of a high receipt volume. However, the nature of the dynamics is such that more than one bad action at this level of receipt volume precipitates a chain reaction leading to an overflow; on the other hand, sound actions keep area occupancy at low levels.



(a) Raw policy (b) Optimized policy

Figure 7: A comparison of the level of inventory in area H of Fig. 2 on each of the 20 days.

6 CONCLUSION

In this paper, we presented a plant operation scheduling technique based on reinforcement learning. The choice of RL was motivated by practical scenarios wherein schedule controllers need to be designed without access to detailed plant models and with access to just the raw material receipt schedules and high-level characteristics and operating constraints of the plant. We introduced the concept of moments as a compact and generalizable way to represent raw material receipt schedule. A Monte-Carlo-based policy iteration scheduler was designed, together with the necessary neural network-based function approximator, and demonstrated in simulation. The example plant considered in the paper is a highly simplified, yet representative, example of complex industrial plants. It is expected that the scheduling technique introduced here can be scaled readily to these complex plants.

REFERENCES

Al-Othman, W. B., H. Lababidi, I. M. Alatiqi, and K. Al-Shayji. 2008. "Supply Chain Optimization of Petroleum Organization under Uncertainty in Market Demands and Prices". *European Journal of Operational Research* 189(3):822–840.

Bertsekas, D. P., M. L. Homer, D. A. Logan, S. D. Patek, and N. R. Sandell. 2000. "Missile Defense and Interceptor Allocation by Neuro-Dynamic Programming". *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 30(1):42–51.

Brandeau, M. L., G. S. Zaric, and A. Richter. 2003. "Resource Allocation for Control of Infectious Diseases in Multiple Independent Populations: Beyond Cost-Effectiveness Analysis". *Journal of Health Economics* 22(4):575–598.

Ernst, D., M. Glavic, F. Capitanescu, and L. Wehenkel. 2009. "Reinforcement Learning versus Model Predictive Control: A Comparison on a Power System Problem". *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39(2):517–529.

Mendez, C. A., I. E. Grossmann, I. Harjunkoski, and P. Kaboré. 2006. "A Simultaneous Optimization Approach for Off-Line Blending and Scheduling of Oil-Refinery Operations". *Computers & Chemical Engineering* 30(4):614–634.

Negenborn, R. R., B. De Schutter, M. A. Wiering, and H. Hellendoorn. 2005. "Learning-Based Model Predictive Control for Markov Decision Processes". *IFAC Proceedings Volumes* 38(1):354–359.

Pinedo, M. 2012. *Scheduling - Theory, Algorithm, and Systems*. 5th ed. Cham, Switzerland: Springer.

Pinto, J. M., M. Joly, and L. F. L. Moro. 2000. "Planning and Scheduling Models for Refinery Operations". *Computers & Chemical Engineering* 24(9-10):2259–2276.

Prokop, R. J., and A. P. Reeves. 1992. "A Survey of Moment-Based Techniques for Unoccluded Object Representation and Recognition". *CVGIP: Graphical Models and Image Processing* 54(5):438–460.

Sutton, R. S., and A. G. Barto. 2018. *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge, MA: MIT Press.

Williams, G., N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. 2017. "Information Theoretic MPC for Model-Based Reinforcement Learning". In *2017 IEEE International Conference on Robotics and Automation (ICRA), May 29th - June 3rd, Singapore*, 1714–1721.

Zhang, T., G. Kahn, S. Levine, and P. Abbeel. 2016. "Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search". In *2016 IEEE International Conference on Robotics and Automation (ICRA), May 16th - 21st, Stockholm, Sweden*, 528–535.

Zhang, W., and T. G. Dietterich. 1995. "A Reinforcement Learning Approach to Job-Shop Scheduling". In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, 1114–1120. San Francisco, California: Morgan Kaufmann Publishers Inc.

Zhang, W., and T. G. Dietterich. 1996. "High-Performance Job-Shop Scheduling with a Time-delay TD (λ) Network". In *Advances in Neural Information Processing Systems 8*, edited by D. S. Touretzsky, M. C. Mozer, and M. E. Hasselmo, 1024–1030. Cambridge, Massachusetts: MIT Press.

AUTHOR BIOGRAPHIES

SATYAVRAT WAGLE is a researcher in the Software Systems and Services Research Area (SSS-RA) at Tata Consultancy Services (TCS) Research. He holds an MS in Machine Learning from Carnegie-Mellon University. His work is focused on machine learning and reinforcement learning. Email: satya.wagle@tcs.com.

ADITYA A. PARANJAPE is a senior scientist in SSS-RA at TCS Research. He received his PhD in Aerospace Engineering from the University of Illinois at Urbana-Champaign in December 2011. His areas of research include adaptive control, robotics and multi-agent systems. He has held tenure-track faculty positions at McGill University, IIT Bombay, and Imperial College London. His email address is aditya.paranjape@tcs.com.