Computers and Chemical Engineering 141 (2020) 106982

Contents lists available at ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/compchemeng

A deep reinforcement learning approach for chemical production scheduling

Christian D. Hubbs[a], Can Li[a], Nikolaos V. Sahinidis[a,*], Ignacio E. Grossmann[a], John M. Wassick[b]

[a] Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15123, United States
[b] Dow Chemical, Digital Fulfillment Center, Midland, MI 48667, United States

ARTICLE INFO

Article history:
Received 13 February 2020
Revised 13 June 2020
Accepted 17 June 2020
Available online 19 June 2020

Keywords:
Machine learning
Reinforcement learning
Optimization
Scheduling
Stochastic programming

ABSTRACT

This work examines applying deep reinforcement learning to a chemical production scheduling process to account for uncertainty and achieve online, dynamic scheduling, and benchmarks the results with a mixed-integer linear programming (MILP) model that schedules each time interval on a receding horizon basis. An industrial example is used as a case study for comparing the differing approaches. Results show that the reinforcement learning method outperforms the naive MILP approaches and is competitive with a shrinking horizon MILP approach in terms of profitability, inventory levels, and customer service. The speed and flexibility of the reinforcement learning system is promising for achieving real-time optimization of a scheduling system, but there is reason to pursue integration of data-driven deep reinforcement learning methods and model-based mathematical optimization approaches.

1. Introduction

Industrial chemical operations in the modern day convert thousands of tons of raw material inputs into thousands of tons of product worth tens of millions of dollars each day. Complex questions regarding resource allocation must be asked and answered continuously (Harjunkoski et al., 2014). What to produce? When to produce it? How much to produce? How much to sell now vs how much to store in inventory and for how long? Shobrys and White (2002) estimated that "good" answers to these decisions can further enhance profit margins. Businesses are also faced with increased pressure from competition and innovation, thereby forcing modifications to production strategies to remain competitive (Harjunkoski et al., 2009). Moreover, these decisions must be made in the face of significant uncertainty. Production delays, plant shutdowns or stoppages, rush orders, fluctuating prices, and shifting demand are all sources of uncertainty that render a previously optimal schedule, sub-optimal or even infeasible (Li and Ierapetritou, 2008; Gupta and Maravelias, 2016).

Optimization under uncertainty has received significant attention by the process systems engineering community and a plethora of approaches have been developed. While explicitly accounting for uncertainty can greatly improve the result of the model above deterministic formulations, this improved result comes with a trade-off (Huang and Ahmed, 2009). Models including uncertainty lead to significantly higher computational costs due to a large number of scenarios in cases where discrete uncertainty is present. Computational costs of models which represent uncertainty as continuous probability distributions are driven by integration requirements (Balasubramanian and Grossmann, 2003). Here, we will only focus on the primary areas of research that are directly relevant to the planning and scheduling problem we are pursuing (see Sahinidis (2004) for a fuller discussion of optimization under uncertainty).

Two primary methods to address planning and scheduling under uncertainty have emerged over the years: robust optimization and stochastic optimization (Grossmann et al., 2016). Robust optimization ensures a schedule is feasible over a given set of possible outcomes of the uncertainty in the system (Bertsimas et al., 2011). Lin et al. (2004) provides an example of robust optimization for scheduling a chemical process modeled as a continuous time state-task network (STN) with uncertainty in the processing time, demand, and raw material prices.

The stochastic optimization approach deals with uncertainty in stages whereby a decision is made, then the uncertainty is revealed which enables a recourse decision to be made given the new information. For scheduling applications, Jung et al. (2004) de-

* Corresponding author.
E-mail address: niksah@gmail.com (N.V. Sahinidis).

https://doi.org/10.1016/j.compchemeng.2020.106982

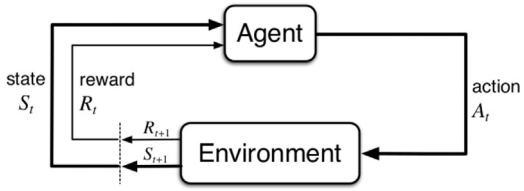

Fig. 1. Diagram of reinforcement learning system.

velops a multi-stage stochastic optimization model to determine safety stock levels to maintain a given customer satisfaction level with stochastic demand. Sand and Engell (2004) developed a two-stage stochastic mixed-integer linear program to address the scheduling of a chemical batch process with a rolling horizon while accounting for the risk associated with their decisions.

In this work, we use reinforcement learning to address the uncertainties in the production planning and scheduling problem and illustrate its application in an industrial, single-stage, continuous chemical manufacturing process. Reinforcement learning (RL) recasts the learning problem as a Markov Decision Process (MDP). Discrete, stochastic MDPs were first introduced by Bellman (1957) and consist of a series of sequential states with probabilistic transitions between them. Transitions to these states are influenced by decisions made at each prior. Thus the production scheduling problem we are pursuing here can be cast as a Partially Observable MDP (POMDP) where states are defined as inventory levels, demand, and so forth, with decisions being what to schedule next. The next state is a consequence of previous decisions and the realization of any random variables in the system.

As illustrated in Fig. 1, reinforcement learning involves an *agent* which takes *actions* based on observations and information – known as the *state* – it receives from the *environment* (Sutton and Barto, 2018). In the case of planning and scheduling, the agent is the scheduling algorithm and its actions are scheduling decisions (i.e. what to produce, when to produce, how much, etc.). The environment is the plant, factory, or machine that will do the processing, and the state can be defined as inventory levels, demand, the existing schedule, or whatever information is deemed relevant to developing a schedule. The goal of the agent is to maximize a *reward* which is a scalar signal it receives through interacting with the environment. The reward is given by a *reward function*, which is the reinforcement learning equivalent to an objective function in traditional optimization parlance. The agent learns a *policy* to map states to actions in order to maximize the reward signal the agent receives from the environment.

Reinforcement learning has been used with good results in scheduling problems, although literature on the topic remains sparse. One of the earliest papers on RL methods and scheduling comes from Zhang and Dietterich (1995) where the *TD*(λ) algorithm was applied to train a neural network to schedule NASA's space shuttle pay load processing (Sutton, 1988). In comparison with simulated annealing, Zhang et al. showed better results in less computational time using RL (Metropolis et al., 1953). Schneider et al. (1998) implemented a dynamic programming approach to achieve near-optimal results on a stochastic job shop scheduling problem based on an industrial facility. Riedmiller and Riedmiller (1999) used a multilayer perceptron and an RL technique – Q-learning – to learn local policies to minimize the tardiness for a flexible job shop scheduling problem (Watkins, 1989). Stockheim et al. (2003) applied Q-learning to a job acceptance system whereby, the RL agent would learn a good policy for accepting new jobs, which would then be scheduled using a deterministic scheduling algorithm. Martinez et al. (2011) utilized a tabular Q-learning approach for a flexible job shop scheduling problem and showed superior results for Q-learning ant colony optimization and tabu search, yet under-performed relative to a genetic algorithm. Mortazavi et al. (2015) used discretized Q-learning to develop a four-echelon supply chain simulation consisting of a retailer, distributor, manufacturer, and supplier. The Q-learning system is able to adapt and learn based on non-stationary demand described by a Poisson distribution with a mean that changes deterministically on a 12-week cycle. Palombarini et al. (2018) used the *SARSA*(λ) algorithm to develop logical if/else repair operators to build robustness into schedules (Singh and Sutton, 1996).

Aside from scheduling, other areas of chemical engineering which involve uncertainty beyond planning and scheduling are amenable to RL applications. For example, Morinelly and Ydstie (2016) used Q-learning to develop an adaptive control system that learns the optimal policy (a Linear Quadratic Regulator) to control a discrete linear system with uncertain dynamics. Badgwell et al. (2018) drew parallels to RL and MPC for process control, highlighting the potential RL has to supplement human knowledge (e.g. operators in a control plant monitoring operations), serve as a replacement for existing process control technology, integrate with existing MPC frameworks, and simplify operations of a complex chemical plant. Lewis and Liu (2013) expounded upon application areas of RL in engineering with a particular emphasis on process controls.

In recent years, the field of deep reinforcement learning (DRL) – which employees multi-layer neural networks (or deep neural networks) for value and policy approximations – has emerged and provided extraordinary results on problems once thought to be decades away. For example, in 2016 Google DeepMind developed AlphaGo to defeat the European Go champion, Lee Sedol (Silver et al., 2016). The following year, these same techniques were used to defeat the world champion Ke Jie and a new system was developed – AlphaGo Zero – which learned without human input and easily defeated the previous AlphaGo system 100-0 (Silver et al., 2017). A thorough overview of the developments in DRL and applications of the technique is provided by Li (2017).

Given the success of DRL in large problems and the amenability of planning and scheduling problems to MDP representations (Schneider et al., 1998), it seems natural to extend these techniques to industrial planning and scheduling models. The literature on DRL in this area, however, is severely lacking – perhaps not surprising given the recency of many of the DRL accomplishments. Regardless, there is some research in this area, notably Oroojlooyjadid et al. (2017) applied a single, deep Q-network (DQN) to each of the four stages of the beer game (retailer, wholesaler, distributor, and manufacturer) to obtain near-optimal results. Mao et al. (2016) considered an application of the REINFORCE algorithm to assigning jobs on a large-scale computational cluster showing that this technique is superior to existing scheduling heuristics in their domain (Williams, 1992). Lee et al. (2018) provide an overview of machine learning techniques, including reinforcement learning, for the process systems engineering field.

We conduct simulated experiments based on data provided by our industrial partners. We seek to determine the applicability of DRL to production scheduling under uncertainty in comparison to traditional optimization approaches. Given the computational costs associated with stochastic programming and other techniques to optimize under uncertainty and the need for regular updates based on new data, these models are difficult to implement in industrial production planning and scheduling settings of the sort explored within this paper. This paper is divided into six sections. Section 2 describes the industrial problem and data we have modeled our experiments on. Section 3 provides the details of our DRL approach as well as various MILPs. Section 4 consists of an illustrated problem to clarify the concepts and enable easy repro-

Table 1
Product data for simulated reactor.

Product	Run Rate (MT/Day)	Average Standard Margin ($/MT)	Curing Time (Days)
A	218	28	1
B	237	39	1
C	259	40	1
D	246	44	1

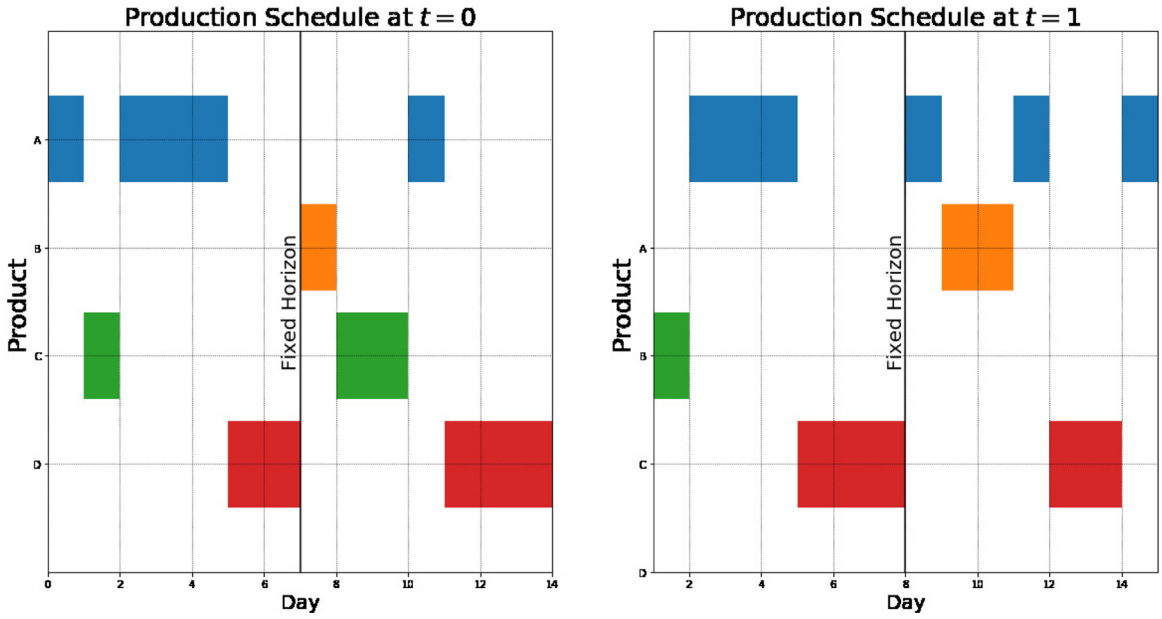

Fig. 2. Receding horizon schedule.

ducibility for the reader. Sections 5 and 6 cover the results and conclusions, respectively.

2. Problem statement

We consider a continuous, chemical manufacturing process with a single stage and single production unit operating under stochastic demand modeled after a site owned and operated by Dow Inc.[1] The goal of the scheduling algorithms is to build a production schedule for the full planning horizon of K days (product data is given in Table 1, all reported mass quantities are given in metric tons, MT). The schedule is fixed for the first H days (fixed horizon) in accordance with operating procedures provided by the plant. This is done in order to provide operating stability for the production facility and maintain commitments to down-stream customers where deliveries have been confirmed. This makes the schedule inflexible, particularly in cases where new orders are entered into the system for high-priority customers with a short lead time. This variation in demand may change a previously optimal schedule to become a sub-optimal schedule.

This process is repeated each day throughout the simulation horizon (see Fig. 2 for an illustration). Additionally, there are losses due to type changes, which the model must take into account. These losses are minimized by moving to and from products with similar production temperatures and compositions. Making large jumps in processing temperature, for example, will yield production of off-grade material, which is outside of product specifications and cannot be sold for the same price as prime product thus reducing the margin and negatively impacting the profitability of the facility. Because off-grade is typically sold off on spot markets when it is made at whatever price is available, we do not take into account off-grade revenue and consider it as lost production. These transition losses are modeled by reducing the quantity produced in the next period. Thus, if product D is selected at time t and product B at $t+1$, the model will incur transition losses total to the day's production quantity, effectively losing an entire day of production for B to off-grade material. The model may continue with product B in the next period, at which point it will register a transition from B to B and no off-grade losses will be incurred (Table 2).

Table 2
Product transition losses (MT).

		Transition To			
		A	B	C	D
Transition From	Start-Up	2	3	5	5
	A	0	0	72	0
	B	45	0	76	75
	C	0	237	0	75
	D	45	237	0	0

Demand is represented by an order book and is generated at the beginning of the simulation according to a fixed statistical profile. The demand is revealed to the planning models when the current day matches the order entry date that is associated with each order in the system. This provides limited visibility to the models of future demand and forces it to react to new entries as they are made available. Furthermore, demand is divided into two types: actual demand and forecasted demand. These are distinguished by a binary flag in the order book (see Table 3 for an example).

The actual demand determines specific orders that must be satisfied, while the forecasted demand does not, but instead serves

[1] All dollar values, order quantities, profit margins, and so forth are shown for illustrative purposes only and do not represent actual values of Dow Inc.

Table 3
Sample order book excerpt.

Document Number	Document Creation Date	Order Due Date	Product	Order Quantity (MT)	Order Margin ($/MT)	Forecast Flag
1	14	20	B	25	36	0
2	-5	3	D	25	45	1
3	23	25	C	25	42	0
4	20	26	A	25	27	0
5	15	22	B	25	39	1

Table 4
Simulation parameters.

Parameter	Value	Description
H	7	Fixed Planning Horizon
K	15	Lookahead Planning Horizon
D	90	Number of Periods in Simulation
η	12%	Percentage of Annual Working Capital Cost
α	25%	Daily Late Shipment Penalty

as an estimate of the anticipated demand for the scheduling algorithms. Following the set-up of our real-world problem, the forecast is provided as an input to the model, and is simply a high-level number that indicates the anticipated total demand for each product in a month. Traditionally, the planning and scheduling role has been performed by humans, thus they can work with a high-level estimate and update their schedules as more information comes in each day. Additionally, lower-level forecasts that attempt to predict specific order entry and shipment dates are too inaccurate to be of any value, whereas a reasonable estimate of total monthly demand can be forecasted. In the reinforcement learning case, the forecast can simply be an entry in the state description, however, more complex, disaggregation methods need to be explored to combine a high-level demand forecast with actual, low-level orders. Further information on the specific strategies employed can be found in Section 3.5.

The model is discretized in time to one day time intervals and the scheduling problem takes place over a 90 day horizon. The planning models must operate in the presence of a fixed planning horizon H, meaning the schedule cannot change for the next H days out. For example, if $H = 7$ and a schedule is made on January 1st for the 1st-15th, then the schedule cannot be altered between the 1st and the 7th. All scheduling algorithms thus build a schedule for a K day planning horizon (where $K \geq H$) on a rolling horizon and freeze the schedule for the next H days as seen in Table 4.

Production quantities are discretized into 24-hour periods based on their run rates (MT/day). Additionally, a 24-hour curing time is required for each product before it can be shipped to allow for cooling and degassing. Thus, any given product scheduled to begin production on January 1st will not be available to be shipped until January 3rd, and thus orders cannot be satisfied until then. The transition losses are applied to daily production runs, so that transition from A to C, for example, yields a loss of 72 MT of prime production.

A late penalty (α) of 25% of the order value is assessed for each day that an order is late. Thus an order worth $1,000 and shipped on time is worth the full, $1,000 but only $750 if shipped one day late, $500 if two days, and so forth.

The goal of the reinforcement learning and optimization methods are to maximize the profitability of the site over the simulation period. The reward/objective function is given as:

$$\max z = \sum_i \sum_t \sum_n V_{in} S_{itn} - \eta \sum_i \sum_t \beta_i I_{it} \tag{1}$$

where V_{in} indicates the discounted standard margin for order n and product i, and S_{itn} denotes the shipped amount for order number n for each time period t corresponding to product i. This represents the income – although it may become a loss if there are sufficient late penalties to pay – while the costs are related to carrying large inventory I_{it}, with β_i being the average variable standard margin for each product and η being a fractional, capital cost multiplier.

The approaches are compared using the total profitability and other key performance indicators such as overall tardiness, prime production rate, number of type changes, and inventory levels. All solutions are validated against a simulated production environment with identical demand and stochastic elements to enable a direct comparison of the results.

3. Methods

Five classes of models are developed for comparison: a DRL model, deterministic MILP on a rolling horizon, a deterministic MILP on a shrinking horizon, a stochastic MILP on a rolling horizon, and a perfect information MILP which optimizes over the entire horizon and with no uncertainty. The DRL model is trained using the Advantage Actor-Critic algorithm (abbreviated A2C, see Section 3.1.1 for details). The deterministic model is tested with and without a forecast. The shrinking horizon method was introduced in order to develop better, long-range planning such as that which is needed in this problem. The stochastic MILP is developed and tested to compare an optimization model under uncertainty with the DRL approach. Finally, the perfect information MILP is used to give an optimistic upper-bound on model performance for benchmarking purposes. All solutions are validated in simulation to provide the results.

3.1. Reinforcement learning model

Machine learning is typically divided into three categories: supervised learning, unsupervised learning, and reinforcement learning (Bishop, 2006). Supervised learning deals with labeled data and is used to address classification and regression problems. Unsupervised learning seeks to find patterns in the structure of data such as with clustering algorithms. Reinforcement learning is agent-based whereby an agent interacts with an environment in order to maximize a reward.

The agent is modeled using a deep neural network with twelve hidden layers and 256 nodes for each layer with exponential linear unit (ELU) activation functions:

$$ELU(x) = \begin{cases} x & \text{if } x > 0 \\ \Lambda(e^x - 1) & \text{if } x \leq 0 \end{cases} \tag{2}$$

$$\sigma(x)_m = \frac{e^{x_m}}{\sum_{m=1}^{M} e^x} \quad \forall m \in M \tag{3}$$

where Λ is a constant set to 1 (Eq. (2)) - and a softmax output (Eq. (3)). The network outputs a probability distribution over the set of possible actions A. This probability distribution is sampled to yield a_t, or the product to make at time t in the schedule. The schedule is built by successive forward propagation of the state through the network to yield distributions, which are sampled to

generate a schedule. At each time step of the simulation a reward, R_t, is returned as feedback to the model to train on at the end of the episode.

3.1.1. Reinforcement learning algorithm

We implemented a version of the Advantage Actor-Critic (A2C) algorithm (Mnih et al., 2016). This is a policy gradient algorithm that learns a value function approximation (the critic) and a policy function (the actor).

The critic learns to approximate the value of the current policy ($\hat{v}$) - according to its parameterization ($\theta_{\hat{v}}$) - which is the expectation of the sum of the future discounted rewards:

$$\hat{v}(s_t, \theta_{\hat{v}}) \approx \mathbb{E}\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots + \gamma^T R_{t+T}\right] \tag{4}$$

where s_t is the state at time t, R_t is the reward received at time t from transitioning from s_t to s_{t+1}, and γ is a discount factor where $0 < \gamma \leq 1$, which causes current rewards to be more valuable than future rewards.

The actor learns a stochastic policy (π) parameterized by θ_π that is designed to take the action that has the greatest probability of maximizing the sum of the future rewards. The policy is thus parameterized by θ_π and yields a probability distribution over the set of available actions at time t:

$$\theta_\pi = \arg\max_\theta \sum_{t=1}^{T} \mathbb{E}\left[R(s_t, a_t)\right] \tag{5}$$

where R indicates the reward function evaluated by taking action a at state s and time t. The action a_t is sampled from the set of available actions A_t according to the probabilistic output of the policy network during training to encourage exploration. During testing, the maximum value is selected instead. It is important to note that the reward function is identical to the objective function that the algorithm seeks to maximize or minimize over the planning horizon, however we provide feedback at every time step by evaluating the reward function for each step as shown below in Eq. (6).

$$R(s_t, a_t) = \sum_i \sum_n V_{in} S_{itn} - \eta \sum_i \beta_i I_{it} \tag{6}$$

Updates are made to the parameters by calculating the temporal difference error (TD-Error).

$$\Delta = R_t + \gamma \hat{v}(s_{t+1}, \theta_{\hat{v}}) - \hat{v}(s_t, \theta_{\hat{v}}) \tag{7}$$

The gradient of the TD-Error is taken with respect to the parameters, which are then updated using stochastic gradient ascent at the end of each episode, where an episode contains all of the time steps from the beginning of the simulation until its termination (Sutton and Barto, 2018).

Policy gradient algorithms were first proposed by Williams (1992), and later a proof for local optimality convergence was provided by Sutton et al. (1999). More recently, these algorithms provided the backbone for AlphaGo's historic achievements (Silver et al., 2016).

The A2C algorithm deploys an actor-critic pair represented as a deep neural network to act on a Monte Carlo simulation of the planning environment. After n-steps, the error is calculated and backpropagated through the network to update value estimation and policy of the network as discussed above. Pseudo-code for the A2C algorithm is given in Algorithm 1. All networks were implemented in PyTorch v1.0 (Paszke et al., 2017).

3.1.2. Action selection

The agent is modeled using a deep neural network (DNN) to represent the *policy* π – the function which maps states to actions. The policy is stochastic and yields a probability distribution over possible actions for each state. In typical DRL applications, the action taken at time t is immediately acted upon and the environment transitions to a new state, s_{t+1}. In the case of planning and scheduling, decisions must be made in advance for the entire planning horizon without the benefit of observing the new state. There are at least two ways to deal with this complication: the agent may sample over possible schedules for the planning horizon, or the agent may iteratively sample over all products while taking into account a model of the evolution of future states.

Algorithm 1 Advantage Actor-Critic Algorithm.

Require: A differentiable policy parameterization $\pi(a \mid s, \theta_\pi)$
A differentiable state-value parameterization $\hat{v}(s, \theta_{\hat{v}})$
Select step-size parameters $0 < \alpha_\pi, \alpha_{\hat{v}} \leq 1$
Initialize the parameters $\theta_\pi, \theta_{\hat{v}}$
1: **for** N episodes **do**:
Initialize the episode with s_0
2: **for** T steps in episode **do**:
Get action a_t from current policy π: $a_t \leftarrow \pi(s_t, \theta_\pi)$
Take action a_t and observe reward (r_t) and new state (s_{t+1})
Calculate TD error: $\Delta_t \leftarrow R_t + \gamma \hat{v}(s_{t+1}, \theta_{\hat{v}}) - \hat{v}(s_t, \theta_{\hat{v}}) \quad \forall t \in T$
3: **end for**
Update at end of episode:
Calculate actor loss: $\mathcal{L}(\theta_\pi) \leftarrow -\frac{1}{T}\sum_t^T \log\left(\pi(a_t \mid s_t, \theta_\pi)\right)\Delta_t$
Calculate policy entropy: $H\left(\pi(a_t \mid s_t, \theta_\pi)\right) \leftarrow -\sum_i \pi(a_t \mid s_t, \theta_\pi)\log\pi(a_t \mid s_t, \theta_\pi)$
Update actor: $\theta_\pi := \theta_\pi + \alpha_\pi\left(\nabla_{\theta_\pi}\mathcal{L}(\theta_\pi) + \beta\nabla_{\theta_\pi}H\right)$
Calculate critic loss: $\mathcal{L}(\theta_{\hat{v}}) = \frac{1}{T}\sum_t^T(\Delta_t - R_t)^2$
Update critic: $\theta_{\hat{v}} := \theta_{\hat{v}} + \alpha_{\hat{v}}\nabla_{\theta_{\hat{v}}}\mathcal{L}(\theta_{\hat{v}})$
4: **end for**

The first option is how DRL is usually practiced. However, in such a case as scheduling, the number of possible schedules grows exponentially meaning the action space explodes as new products are added or the planning horizon is increased. Given even a small plant with four products and a planning horizon of seven days, we have $A = P^H = 16,284$ possible schedules to sample from. This would result in needing many more samples before a good policy could be found.

The second option requires some way to predict what the future state, $\hat{s}_t$, will be given the information available at t. This is so because repeatedly passing the agent the current state to build the schedule will result in the same probability distribution over actions. To determine $\hat{s}_t$, we provide the agent a simple, first principle model with an inventory balance: $\hat{I}_{it} = I_{it-1} + \hat{p}_{it} - d_{it}$. The planned production and the available orders on the books give the agent a good first pass at estimating $\hat{s}_t$.

3.1.3. Training

The DRL approach described above is known as a *model-free* algorithm, meaning there is no transition function of the system that is given to the RL agent – it must learn a good policy through trial and error. For this reason, the model must be trained extensively by making mistakes early on, and then learning what actions yield high rewards over time. This is done through a sequence of Monte Carlo simulations of the scheduling environment to provide feedback and data for the algorithm to update the network's parameters.

3.1.4. State

The state of the system is passed to the algorithm from the environment and contains the relevant decision making information. As seen in Fig. 3, for this model, relevant information includes inventory levels, actual demand, forecast, current schedule, and simulation time. The production from the most recent time step is included as a one-hot encoding in the state because only the previ-

Table 5
Definitions of sets, indices, and variables.

Sets and Indices:	
i, j	Product indices.
t	Index for time periods.
n	Index for individual orders.
Continuous Variables:	
p_{it}	Amount of product i produced at the end of interval t (MT).
I_{it}	Quantity of inventory for product i at the end of time interval t (MT).
S_{itn}	Quantity of order n with product i during time (MT) t.
Binary Variables:	
x_{itn}	Binary associated with each order such that $x_{itn} = 1$ if product i is shipped to meet the demand for order n at the end of time interval t and is 0 otherwise.
y_{it}	Denotes the product i scheduled at the beginning of interval t, where $y_{it} = 1$ if the product is scheduled and 0 otherwise.
z_{ijt}	Indexes product transitions, where $z_{ijt} = 1$ if the transition has occurred and 0 otherwise.
Parameters:	
d_{itn}	The demand of product i at t for order number n (MT).
t_n	The due date for order n.
V_{in}	The variable standard margin minus the lateness penalty for each day the order is late ($/MT). The penalty is pre-computed to enable the value to be treated as a parameter.
δ_{ij}	Transition losses from product i to product j.
$b_i^{\max}$	Maximum production quantity for product i (MT/day).

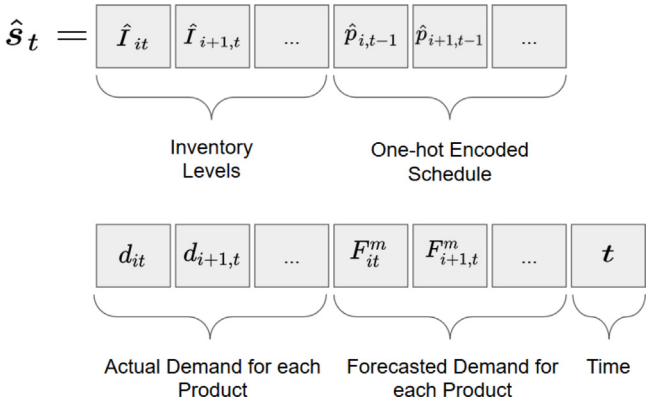

Fig. 3. The state consists of a concatenation of the projected inventories, demand, forecasted demand, one-hot encoding of the previously scheduled product, and the time.

ous scheduled product impacts the off-grade produced during the transition. Faster and more stable training was found by providing the model the current inventory plus planned production and subtracting the actual demand. Additionally, the net forecast is provided to the network for each product-month pair without pursuing any disaggregation strategies such as required in Section 3.5. Combining the state values in this way takes the relevant inputs considered by a planner and reduces the size of the input vector. The feature engineering in the net inventory means the network does not have to learn these relationships itself, which did help speed training.

3.2. Deterministic optimization model

The following model is used to describe a single-stage, continuous production plant. Production decisions are discretized to daily time intervals (t) and it takes one full day for each product i to be produced. Another full day is required for curing time so that the product may be shipped. Thus, it takes two time intervals from when a product is scheduled and production begins until it is available to be shipped and loaded into inventory. The production facility has fixed run rates associated with each product ($b_i^{\max}$) and transition losses are incurred as the plant changes from one product to another as given by the parameter δ_{ij} (see Table 2 for values). All sets, indices, and variables are laid out in Table 5.

Decision variables are given by x_{itn} and y_{it} where x_{itn} is a binary given to each unique order number n which is 1 when it is to be fulfilled with product i at time t, and 0 otherwise. y_{it} is a binary that relates the production of each product i to be produced at time t. y_{it} is equal to one when product i at time t is scheduled and 0 otherwise. Only one product may be produced at a time and an order may only be fulfilled once and only at or after the due date, i.e. no early shipments.

The model is tasked with determining which products to produce and when in order to maximize the profitability given in the objective function. Each order is specified by d_{itn} and has a ship-by date t_n, and a corresponding variable standard margin v_n. A 25% per day penalty is assessed on the variable standard margin for each day beyond the ship-by date that an order is not shipped. Additionally, there is a cost to carrying inventory which is calculated as the sum of the average value of each product currently in inventory and scaled by a working capital multiplier. Thus, the goal is to maximize the value added to the enterprise by scheduling production in order to meet orders on time (or with as little penalty as possible) while maintaining the lowest inventory possible.

The objective function is identical to the DRL agent's reward function (Eq. (1)). However, discounting caused by late penalties are pre-computed and treated as parameters in the system to avoid non-linearities which arise from the relation between the decision variables and the penalty assessment. The full, deterministic optimization model is defined by the objective function in Eq. (1), and constraints in described in the following section.

3.3. Mathematical formulation

In this work, we use a multi-period formulation. The optimization model is formulated as an MILP that develops a schedule on a receding horizon for the full planning period. The MILP generates a schedule for a time horizon consisting of K days where $K = 2H + 1$ to provide better end-state conditions, then the schedule is passed to the facility model to execute. The simulation is stepped forward one time step, and the results are fed back into the MILP model to generate a new schedule over the K-step planning horizon. The objective function is given in Eq. (1) whereby the algorithm seeks to maximize the accrued profit by satisfying orders with as little delay as possible, while minimizing inventory levels.

3.3.1. Mass balance

$$I_{it} = I_{it-1} + p_{it-2} - \sum_j \delta_{ij} z_{ijt-2} - \sum_n \sum_{t \geq t_n} S_{itn} \quad \forall i, t \tag{8}$$

Eq. (8) states that for any product i at time t, we calculate the mass balance of the system as the inventory from the previous period, plus the scheduled production from two days prior (given the two day delay for production and curing), minus any sales and transition losses that may occur due to the production type changes.

3.3.2. Transition constraints

Transition losses are incurred by the system for unfavorable transitions. These are driven by temperature disparities and the presence of additives or other reactants that may remain in the reactor and cause the resulting product to be out of spec and thus labeled as "off-grade" product. Off-grade represents lost production capacity because raw materials and time are consumed in order to produce it, yet it cannot be sold at a favorable price.

Transition constraints are enforced by the following equations:

$$\sum_i z_{ijt} = y_{jt} \quad \forall j, t \tag{9}$$

$$\sum_j z_{ijt} = y_{it-1} \quad \forall i, t \tag{10}$$

These constraints work together to set the relevant binary variable $z_{ijt} = 1$ if product i was made at time $t-1$ and j was produced in the subsequent time period t, where j is an alias of i.

3.3.3. Shipping constraints

Each order in the system has a particular due date, t_n, to determine when the order can be fulfilled. The demand can only be satisfied for times where $t \geq t_n$. In other words, early satisfaction of orders is prohibited. The product quantity and orders to be fulfilled at time t is given by the constraint:

$$S_{itn} = d_{it_n n} x_{itn} \quad \forall n, i, t \geq t_n \tag{11}$$

where the demand parameter $d_{it_n n}$ is satisfied when the corresponding binary variable $x_{itn} = 1$.

Because there may be situations where all demand cannot be satisfied, we relax the constraint on the binary shipping variables such that they can be less than or equal to 1 for all orders n:

$$\sum_i \sum_{t \geq t_n} x_{itn} \leq 1 \quad \forall n \tag{12}$$

This constraint also ensures that each order is only satisfied once, if it is satisfied at all.

3.3.4. Production constraints

Because this model is restricted to a single-reactor production system, only one product may be produced at a time. This is enforced by the assignment constraint:

$$\sum_i y_{it} = 1 \quad \forall t \tag{13}$$

which forces the production variable $y_{it} = 1$ for each time period. This also ensures that the system does not shut-down as per operating procedures, so it must always be producing one of the available products.

Each product has a given run-rate, $b_i^{\max}$, which is fixed and not at the planners discretion for this facility. The next equation ensures that the production values p_{it} do not exceed this run-rate if the product is selected, or else is 0.

$$0 \leq p_{it} \leq b_i^{\max} y_{it} \quad \forall i, t \tag{14}$$

Shipment quantities (S_{itn}), inventory levels (I_{it}), and production quantities (p_{it}) are positive, continuous variables, while binaries are associated with individual order shipments (x_{itn}), production decisions (y_{it}), and product transitions (z_{ijt}). These requirements are expressed by:

$$S_{itn}, I_{it}, p_{it} \geq 0 \tag{15}$$

$$x_{itn}, y_{it}, z_{ijt} \in \{0, 1\} \tag{16}$$

3.4. Stochastic optimization

In addition to the deterministic models, a stochastic programming model was developed to hedge against the uncertainty in the demands of different products. In stochastic programming, it is assumed that the probability distributions of the uncertain parameters are known *a priori* (Birge and Louveaux, 1997). The uncertainties are usually characterized by some discrete realizations of the uncertain parameters as an approximation to the continuous probability distribution. In this scheduling problem, we can generate different realizations of demand using the forecast models, which are described in detail in Section 3.5. Each realization of the demand is called a *scenario*. The set of scenarios is denoted by $\omega \in \Omega$. The scenarios generated from the forecast models are assumed to have equal probabilities.

We use a special case of stochastic programming called two-stage programming for our scheduling problem. Specifically, stage 1 decisions are made 'here and now' at the beginning of the period, and are then followed by the resolution of uncertainty. Stage 2 'wait and see' decisions, or recourse decisions, are taken as corrective actions at the end of the period. Stage 1 decisions usually correspond to the decisions that decision-maker needs to fix right now. For this problem, the scheduler needs to fix the decisions for the next H days from now. Therefore, we treat the decisions corresponding to the first H days from now as the first stage decisions, and the decisions from day $H+1$ to day K as the second stage decisions in the rolling horizon framework.

Compared with the deterministic MILP model, we add index ω to all the decisions variables to distinguish the decisions corresponding to different scenarios. The objective function seeks to maximize the expected profit over all the scenarios:

$$\max \mathbb{E}(z) = \sum_\omega \tau_\omega \left(\sum_n \sum_i \sum_t V_{in\omega} S_{itn\omega} - \eta \sum_i \sum_t \beta_i I_{it\omega} \right) \tag{17}$$

where τ_ω is the probability of scenario ω. We generate 10 different forecasts in the test case, i.e., $\tau_\omega = 0.1$ for all $\omega \in \Omega$.

The constraints of the stochastic model are given in the following subsections. Most of the equations are obtained by extending the equations of the deterministic model to each scenario $\omega \in \Omega$. The only addition is the non-anticipativity constraints, which ensure that the 'here and now' decisions are the same across all the scenarios.

3.4.1. Mass balance

The next equation ensures that for any product i at time t, the inventory carries over from the previous period in the particular scenario ω:

$$I_{it\omega} = I_{it-1\omega} + p_{it-2\omega} - \sum_j \delta_{ij} z_{ijt-2\omega} - \sum_{n\omega} \sum_{t \geq t_{n\omega}} S_{itn\omega} \quad \forall i, t, \omega \tag{18}$$

The potential shipped quantities are given by $S_{itn\omega}$ and differ for each scenario. Again, any losses due to type changes are given by δ_{ij} and found in Table 2.

Table 6
Definitions of sets, indices, and variables for the stochastic optimization model.

Sets and Indices:	
i, j	Product indices.
t	Index for time periods.
ω	Index for the scenarios
n	Index for the individual orders in each scenario.
Continuous Variables:	
$p_{it\omega}$	Amount of product i produced at the end of interval t (MT).in scenario ω.
$I_{it\omega}$	Quantity of inventory for product i at the end of time interval t (MT) in scenario ω.
$S_{itn\omega}$	Quantity of order n with product i during time (MT) t in scenario ω.
Binary Variables:	
$x_{itn\omega}$	Binary associated with each order such that $x_{itn\omega} = 1$ if product i is shipped to meet the demand for order n at the end of time interval t and is 0 otherwise.
$y_{it\omega}$	Denotes the product i scheduled at the beginning of interval t, where $y_{it} = 1$ if the product is scheduled and 0 otherwise.
$z_{ijt\omega}$	Indexes product transitions, where $z_{ijt} = 1$ if the transition has occurred and 0 otherwise.
Parameters:	
$d_{in\omega}$	The demand of product i at t for order number n (MT) in scenario ω.
$t_{n\omega}$	The due date for order and scenario n, ω respectively.
V_{in}	The variable standard margin minus the lateness penalty for each day the order is late ($/MT). The penalty is pre-computed to enable the value to be treated as a parameter.
δ_{ij}	Transition losses from product i to product j.
$b_i^{\max}$	Maximum production quantity for product i (MT/day).

3.4.2. Transition constraints

Transition constraints are enforced by the stochastic versions of the equations found in the deterministic model:

$$\sum_i z_{ijt\omega} = y_{ijt\omega} \quad \forall j, t, \omega \tag{19}$$

$$\sum_j z_{ijt\omega} = y_{it-1\omega} \quad \forall i, t, \omega \tag{20}$$

Each of these constraints behave according to the same dynamics, but is solved for each scenario ω.

3.4.3. Shipping constraints

Each scenario that is generated yields differing orders at various times over the planning horizon. These orders are all given an individual due date, $t_{n\omega}$, that must be met in order to avoid incurring any penalties. As in the deterministic model, early satisfaction of the demand is proscribed.

$$S_{itn\omega} = d_{it_{n\omega}n\omega} x_{itn\omega} \quad \forall i, n, \omega, t \geq t_{n\omega}, \omega, n \tag{21}$$

Similarly, orders may go unfulfilled or be satisfied late, all the while incurring a penalty. This is enabled through the following constraint, which is solved for all scenarios.

$$\sum_i \sum_{t \geq t_n} x_{itn\omega} \leq 1 \quad \forall n, \omega \tag{22}$$

3.4.4. Production constraints

The following constraints are the stochastic versions of the production constraints, yielding the same effects as Eqs. (13) and (14), albeit for all the various scenarios ω.

$$\sum_i y_{it\omega} = 1 \quad \forall t, \omega \tag{23}$$

$$0 \leq p_{it\omega} \leq b_i^{\max} y_{it\omega} \quad \forall t, \omega \tag{24}$$

3.4.5. Non-anticipativity constraints

Non-anticipativity constraints are provided as follows:

$$y_{it\omega} = y_{it\omega+1} \quad \forall \omega, t \leq H \tag{25}$$

These constraints ensure that the first stage production decision variables over the initial fixed horizon H remain consistent across all scenarios ω.

The corresponding variables used in the stochastic model for shipment quantities ($S_{itn\omega}$), inventory levels ($I_{it\omega}$), and production quantities ($p_{it\omega}$), are all positive, continuous variables as given in equation the subsequent equation:

$$S_{itn\omega}, I_{it\omega}, p_{it\omega} \geq 0 \tag{26}$$

Binary variables are given by $x_{itn\omega}$ for the individual orders, $y_{it\omega}$ for the production decisions, and $z_{ijt\omega}$ for the transitions.

$$x_{itn\omega}, y_{it\omega}, z_{ijt\omega} \in \{0, 1\} \tag{27}$$

A complete description of the sets, indices, variables, and parameters for the stochastic model are given in Table 6.

3.5. Forecast disaggregation

The performance of the model is predicated on meeting specific orders with a given order quantity and value. In addition, a high-level forecast is also provided to assist planning decisions where orders are absent. Orders are entered daily, but there are often times when there is excess capacity particularly at the beginning of each month before most orders are entered. Because specific orders are difficult to forecast, greater accuracy can be achieved forecasting total demand at a monthly level, albeit with a given amount of error. Thus, the model must make schedules with two demand signals: actual demand comprised of confirmed orders which, are entered into the system on a daily basis and must be satisfied, along with a high-level forecasted product demand that remains fixed for each month. To integrate these two signals, we developed a disaggregation approach to break the monthly forecast into individual orders distributed over the month as seen in Fig. 4.

The first step taken to disaggregate the forecast is to calculate the *net forecast*, which is simply the total forecast for the period in question minus the sum of the actual demand.

$$F_{net,it}^m = \max\left(F_{it}^m - \sum_{i,n,t \in m} d_{it_c n}, 0\right) \quad \forall i, t \leq t_c \tag{28}$$

If the actual demand is greater than the forecasted demand, the net forecast is 0 and the forecast is discarded because it underestimated the actual demand and provides no new information.

The net forecast is then broken into individual orders and distributed over the remaining horizon. Three basic disaggregation

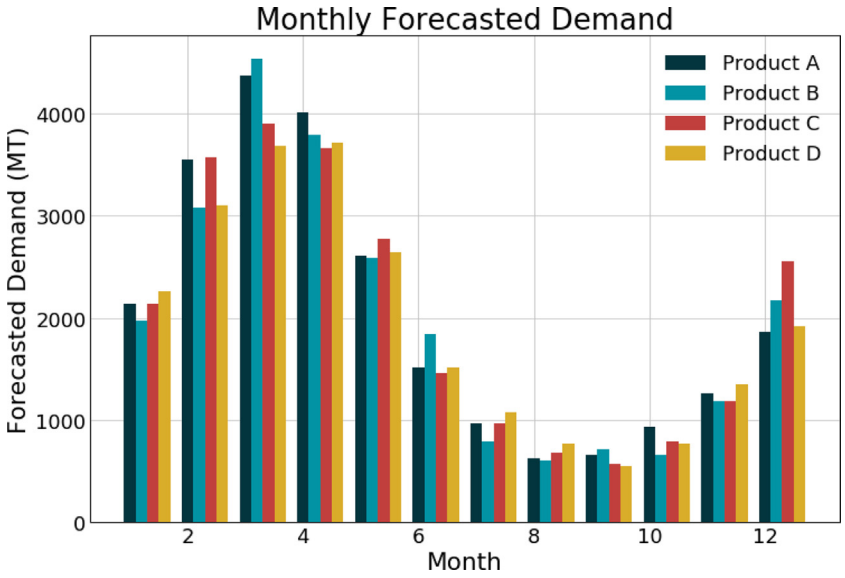

Fig. 4. The forecast is provided at a high level by product and by month while the detailed scheduling must take into account specific orders and the forecast.

approaches were developed, a uniform forecast disaggregation (see Fig. 5), a smoothed forecast disaggregation (see Fig. 6), and a stochastic forecast disaggregation (see Fig. 7).

The following figures provide illustrations of how this is achieved. The top left panel shows the actual demand that must be met as of the first day of the simulation when only a few orders have been entered into the system. The top right panel shows how the net forecast is divided according to the uniform, smoothed, or stochastic approaches. The bottom panel then shows the net or resultant forecast, which is reached by combining the actual demand with the disaggregated, high-level forecast values. This is then treated as the demand (d_{it_nn}) by the MILP and does not alter the formulation of the deterministic or stochastic models.

3.5.1. Uniform forecast disaggregation

The uniform disaggregation approach is straightforward. The monthly net forecast is calculated, and the forecasted demand is spread evenly over the remaining days in the month. This leads to some days that may already have high demand receiving additional demand (as can be seen on day 6 in Fig. 5). This can lead the model to over estimating the short-term demand due to the abnormally large spikes.

3.5.2. Smoothed forecast disaggregation

The smoothed forecast disaggregation attempts to level out the demand over the course of the month by calculating the average demand level, and evening out each of the days until it reaches that point. Days that currently have above average actual demand receive no additional forecasted demand, whereas days that are lacking actual demand are provided forecasted demand.

3.5.3. Stochastic demand forecast

The stochastic MILP requires multiple scenarios to be developed and optimized over. To address this, we keep the actual demand constant across all scenarios and adjust the forecast by sampling from a probability distribution over the remaining days.

3.6. Shrinking horizon model

To incorporate additional information from the forecast, we also implement a schedule with a shrinking horizon such as found in Balasubramanian and Grossmann (2004), which approximates a multistage stochastic programming problem through a sequence of two-stage programming problems with a shrinking horizon. In our case, a deterministic model is solved for the entire planning horizon. As in the rolling horizon models, the first H days are fixed and the K day schedule is passed to the simulator, and the model proceeds to the next time period where it is re-optimized based on updated demand information. This process is repeated for the entire, remaining simulation horizon. The shrinking horizon model uses the smoothed forecast disaggregation method discussed above in order to forecast future orders.

Unlike for the rolling horizon model, which optimizes over a fixed time range, each subsequent sub-model becomes smaller than the previous in the shrinking horizon case. The model is relatively large, particularly early on in the simulation, making it difficult to scale to longer horizons or for a larger number of products or greater demand. This could be addressed by finding a middle ground, and using a rolling-horizon model with a larger lookahead horizon with a shrinking horizon model as seen in Fig. 8 for times where the lookahead horizon is larger than the remaining simulation time.

4. Example

The methods and model discussed in this paper have been tailored to address the specific scheduling needs of an existing site. The business constraints imposed on this problem differ from the existing literature enough to motivate a simplified example to ensure clarity in the mind of the reader. To this end, we will examine a scaled-down, single-stage reactor with only two products and a handful of orders to better understand the decision making problems faced by the models. Further, supporting code for this model can be accessed on GitHub.[2] For this simplified example, we take the values given in Tables 7–8, and assume all product transitions occur without off-grade losses. Additionally, note that in this case, the reactor produces a single order per day of either product A or product B, and has a five-day fixed planning horizon, and a ten-day lookahead horizon that must be maintained. For simplicity, we will further assume no forecast information is available to the model at

[2] See https://github.com/hubbs5/public_drl_sc.

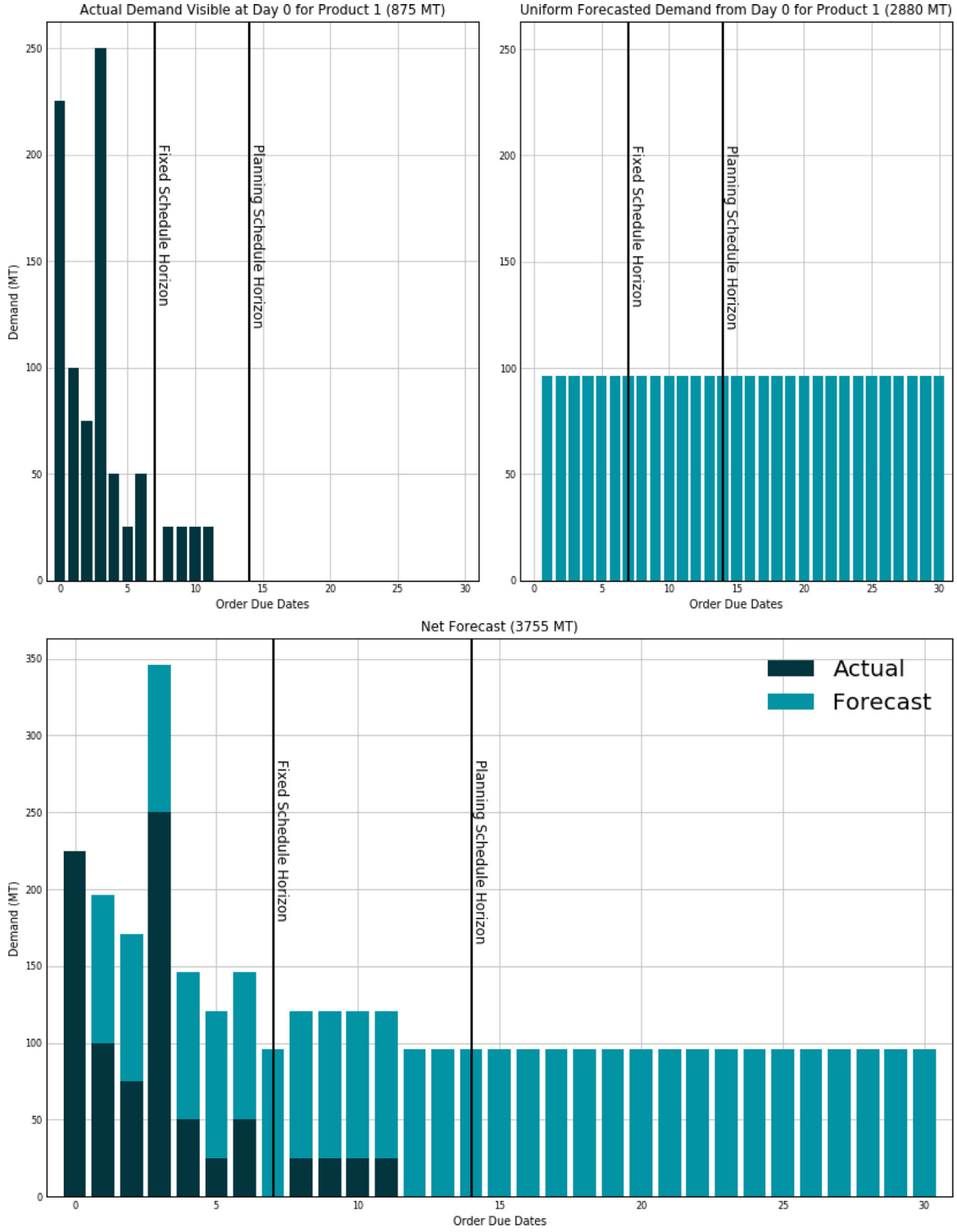

Fig. 5. Forecast disaggregation for a uniform forecast.

Table 7
Product data for simulated reactor.

Product	Run Rate (Orders/Day)	Average Standard Margin ($/Order)	Curing Time (Days)
A	1	10	1
B	1	15	1

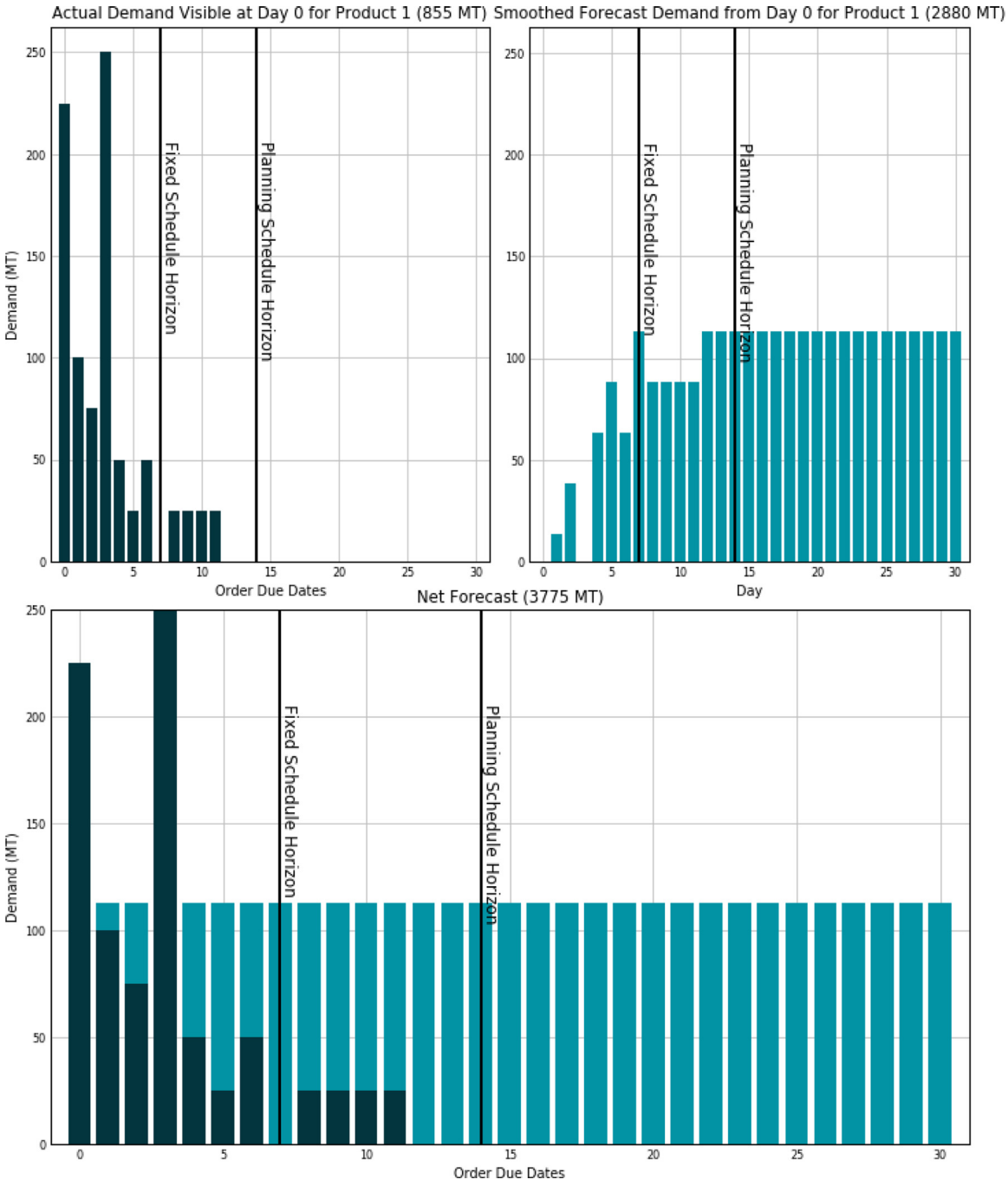

Fig. 6. Smoothed forecast disaggregation.

Table 8
Simulation parameters.

Parameter	Value	Description
H	5	Fixed Planning Horizon
K	10	Lookahead Planning Horizon
D	10	Number of Periods in Simulation
η	10%	Percentage of Annual Working Capital Cost
α	25%	Daily Late Shipment Penalty

this time. Finally, this example will use the same objective function as the actual problem described in Eq. (1).

At day 0, the model has no inventory and must begin producing to meet the demand given in Table 9. The schedule must be developed for the next 10 days according with the first 5 days fixed according to the business rules, even though only enough demand for four days is available.

Given the information we have, the optimal solution would be to simply make product A. This can be seen by inspection. However, we can optimize the deterministic MILP model given in Section 3.3 over this horizon as well, which yields the schedule provided in Fig. 9.

Likewise, given the orders available to use at day 0 and the schedule above, we can view the predicted inventory over the scheduling horizon.

We see in Fig. 10 that net inventory reaches 0 by day 5 and continues to grow at one order per day beyond that.

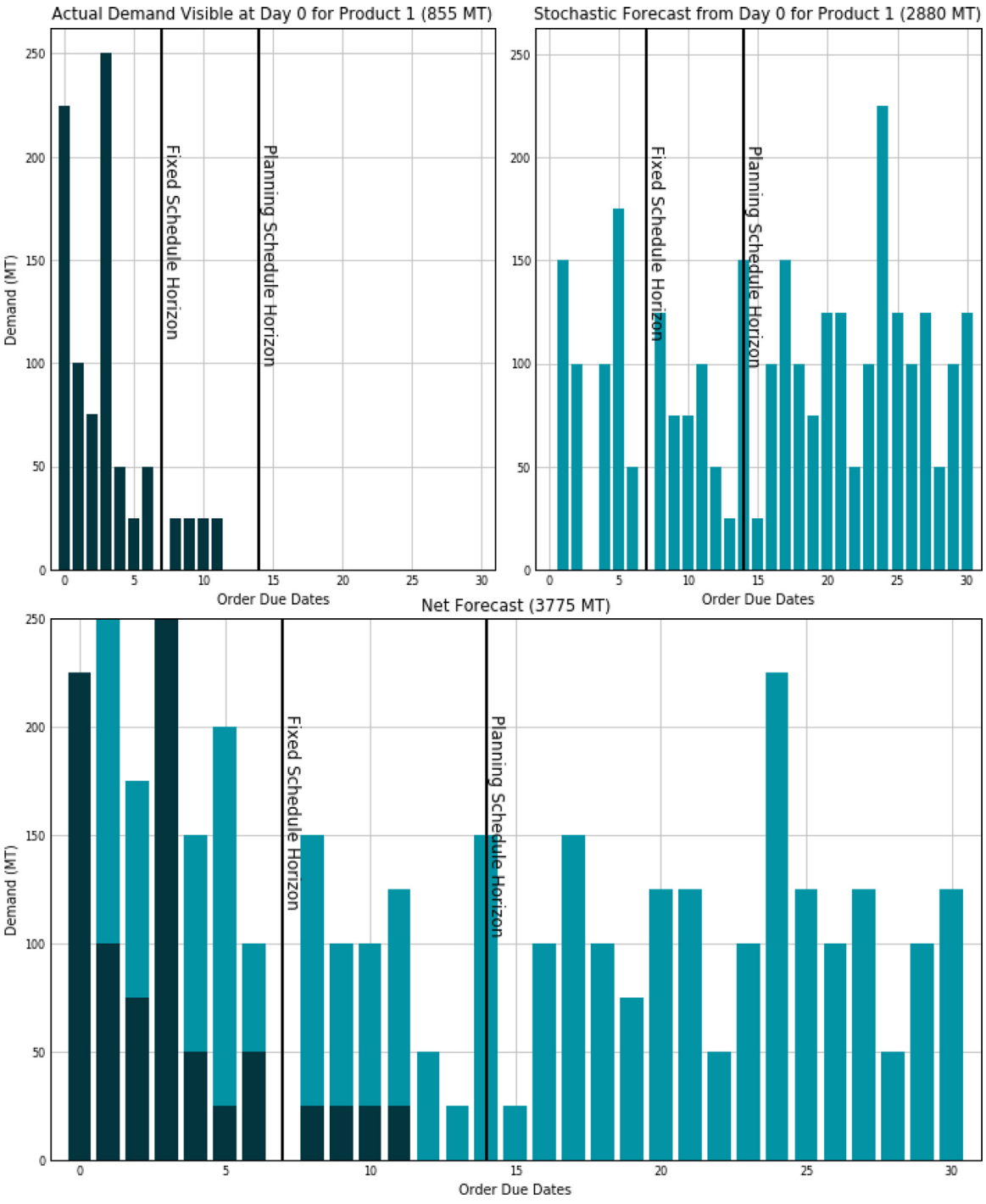

Fig. 7. Stochastic forecast disaggregation.

Table 9
Example order book available at day 0.

Document Number	Document Creation Date	Order Due Date	Product	Order Quantity	Order Margin ($/Order)	Forecast Flag
1	0	3	A	1	10	0
2	0	3	A	1	10	0
3	0	5	A	1	10	0
4	0	4	A	1	10	0

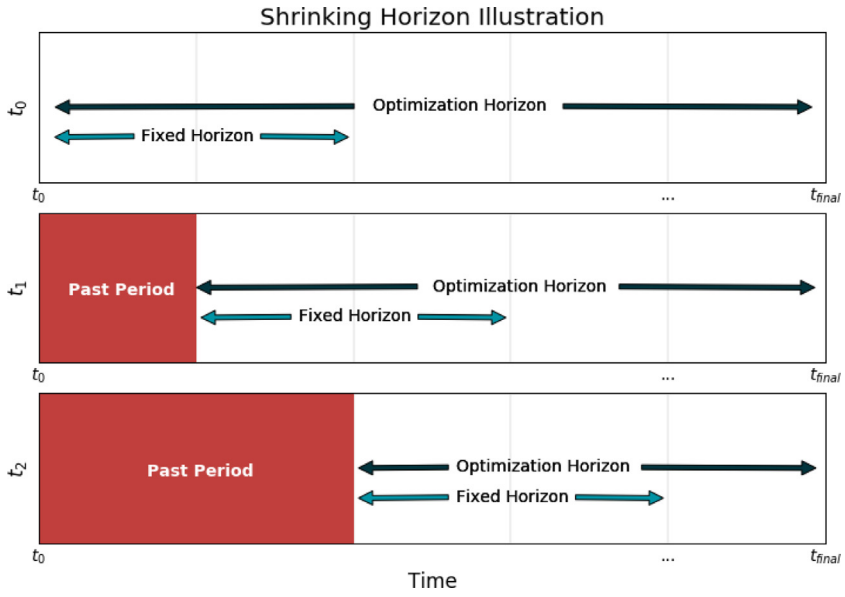

Fig. 8. Illustration of shrinking horizon model.

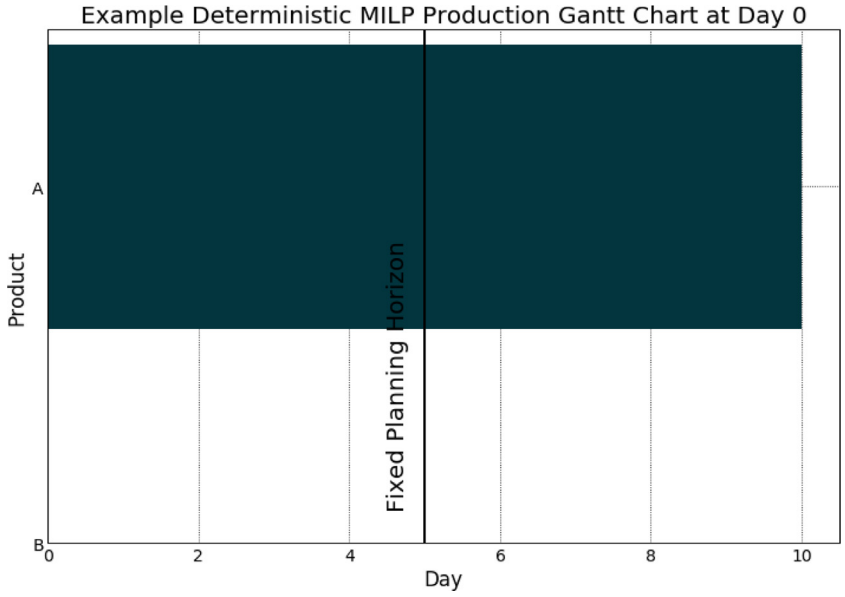

Fig. 9. Optimal schedule for simplified example.

Table 10
Example order book available at day 1.

Document Number	Document Creation Date	Order Due Date	Product	Order Quantity	Order Margin ($/Order)	Forecast Flag
1	0	3	A	1	10	0
2	0	3	A	1	10	0
3	0	5	A	1	10	0
4	0	4	A	1	10	0
5	1	4	B	1	15	0
6	1	5	B	1	15	0

Moving to day 1, new orders are entered into the system, revealing that the model has not scheduled any product to account for demand corresponding to product B. Moreover, these orders fall too close to the fixed horizon to meet customer requests, and will therefore incur a penalty as seen in Table 10.

Re-optimizing at Day 1 while respecting the previous scheduling decisions, we get a new optimal schedule in Fig. 11.

Our predicted inventory is shown in Fig. 12. In this case, the new orders force the net inventory negative, indicating that losses will be accruing on the relevant orders. In complex planning and

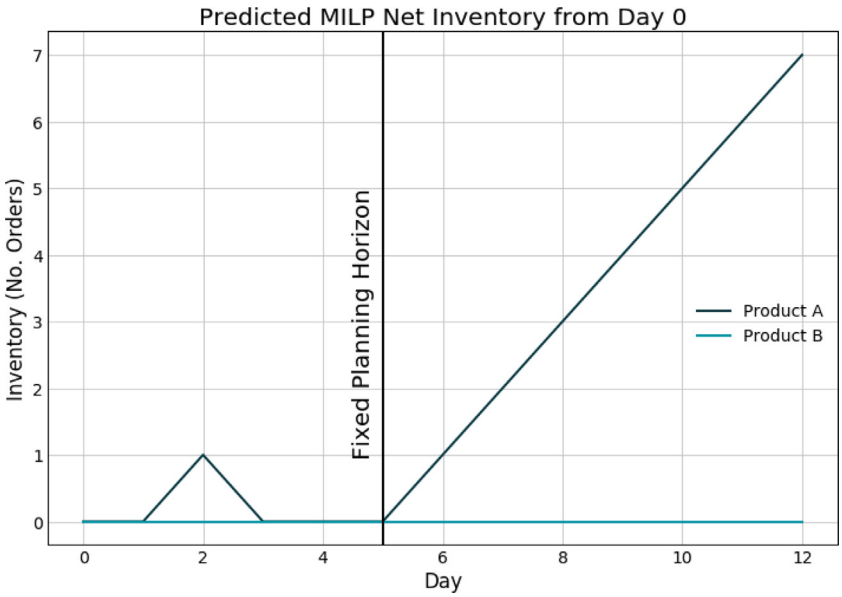

Fig. 10. Predicted net inventory based on optimal schedule and order book at day 0.

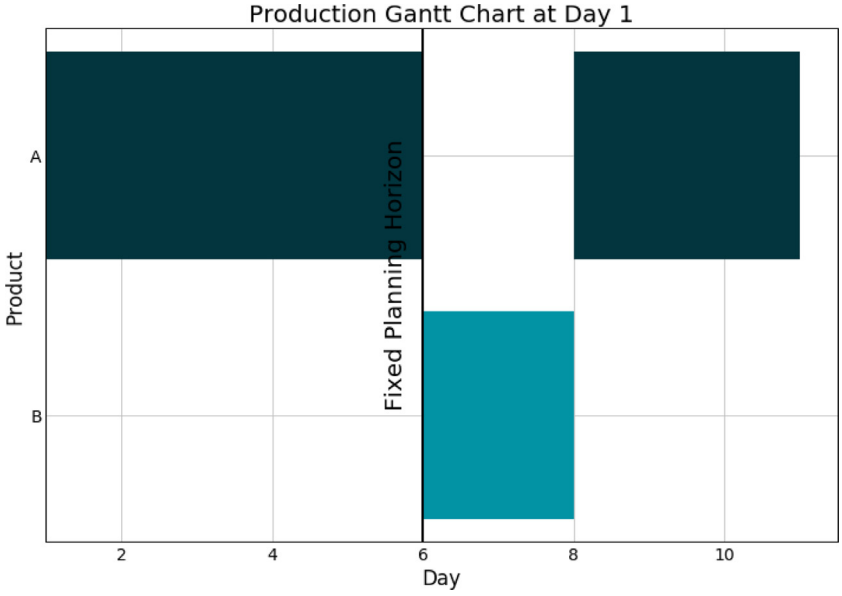

Fig. 11. Optimal schedule for simplified example from day 1.

scheduling problems, these mistakes can continue to compound over time, leading to larger divergence between the implemented scheduling decisions and the optimal.

We can do the same with the RL approach. After training the agent, a simple forward pass through the network enables us to build the schedule shown in Fig. 13. In this case, the RL has been trained on similar demand profiles, therefore it anticipates some demand for Product B to arrive, although no demand has been entered into the system at this time. This gives rise to planned inventory of Product B as shown in Fig. 14. Given that the RL system only has orders for Product A, it prioritizes those orders, but produces safety stock at the earliest possibility. This small example problem trains after roughly 5000 episodes which takes about 110 s to complete using a 2.9GHz Intel i7-7820HQ CPU.

5. Results and discussion

For the single-stage, industrial system, the RL model was subjected to 50,000 Monte Carlo training episodes. Each training episode lasted for 90 simulated days and had the same scheduling task, with different demand scenarios being generated for each episode. As shown in Fig. 15, the agent begins by randomly selecting products to slot into the schedule. The performance early on is quite poor as the agent loses roughly $500,000 per episode over the first few thousand training episodes. The agent quickly learns a productive policy, however, and begins to improve its performance over time. Because the agent continues to explore as it is trained – the entropy loss portion of the loss function encourages exploration – the average performance fluctuates over time as training continues.

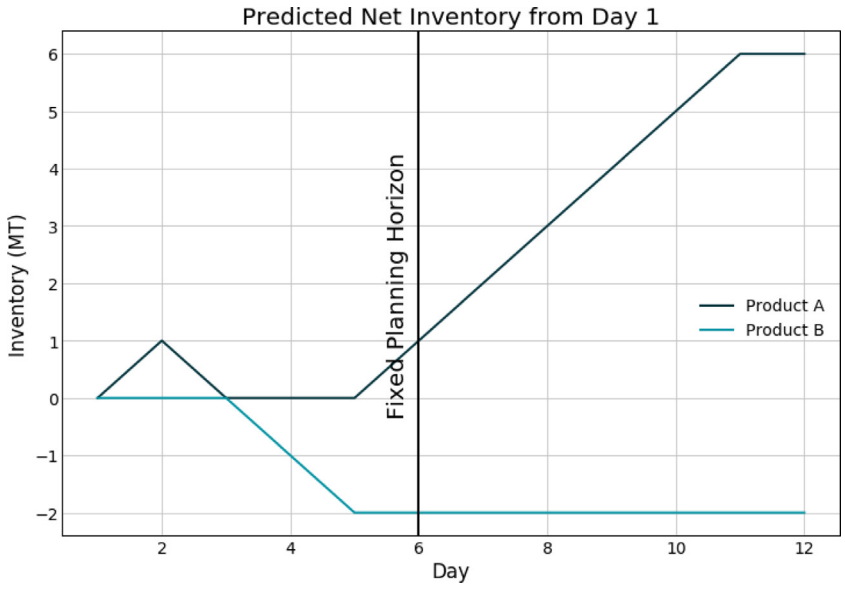

Fig. 12. Predicted net inventory based on optimal schedule and order book at day 1.

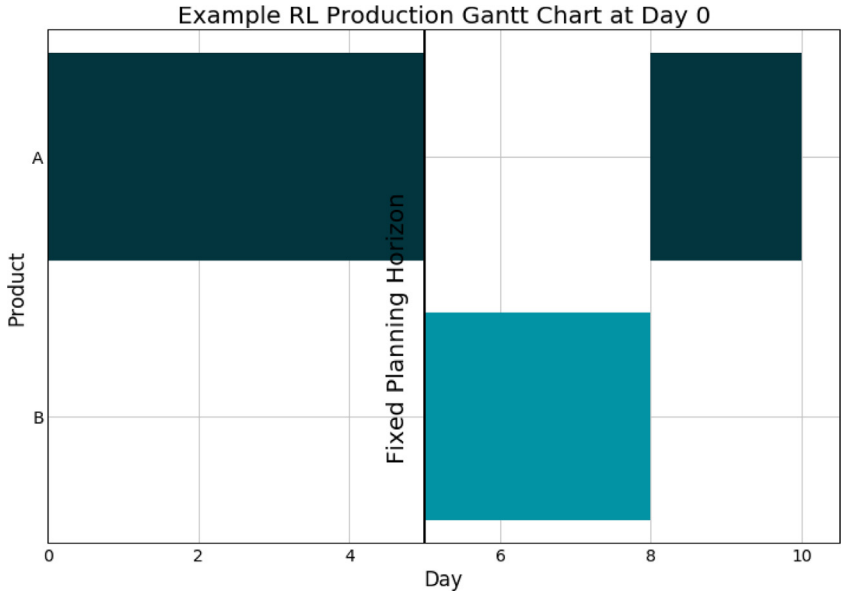

Fig. 13. RL schedule for simplified example from day 0.

The model trained quite readily and without much hyperparameter tuning. The parameters used in training the model above are given in Table 11.

5.1. DRL And MILP performance

All models discussed were tasked with scheduling the reactor over an identical 90-day period. Because of the stochastic nature of the model, we report average results as well as direct comparisons between models while holding the given scenario constant to understand differences. The MILP models were all solved to a 1% optimality gap using Gurobi 8.1 using a 2.9 GHz Intel i7-7820HQ CPU with four cores (Gurobi Optimization LLC, 2020).

Additionally, because all of the models were trained in simulation, we were able to build a deterministic model with perfect information that could optimize the schedule over the entire simulation horizon. This perfect information mixed-integer linear program (PIMILP) yields an optimistic upper bound for the system, and enables more thorough comparisons of the various approaches. This is particularly important because most DRL algorithms lack optimality guarantees.

Fig. 16 shows the performance of the DRL agent along with the deterministic models, two using the previously discussed forecast disaggregation methods, and one without any forecast. The MILP and DRL models solved 10 scenarios and were compared against the PIMILP. The reported gap is the mean absolute percentage dif-

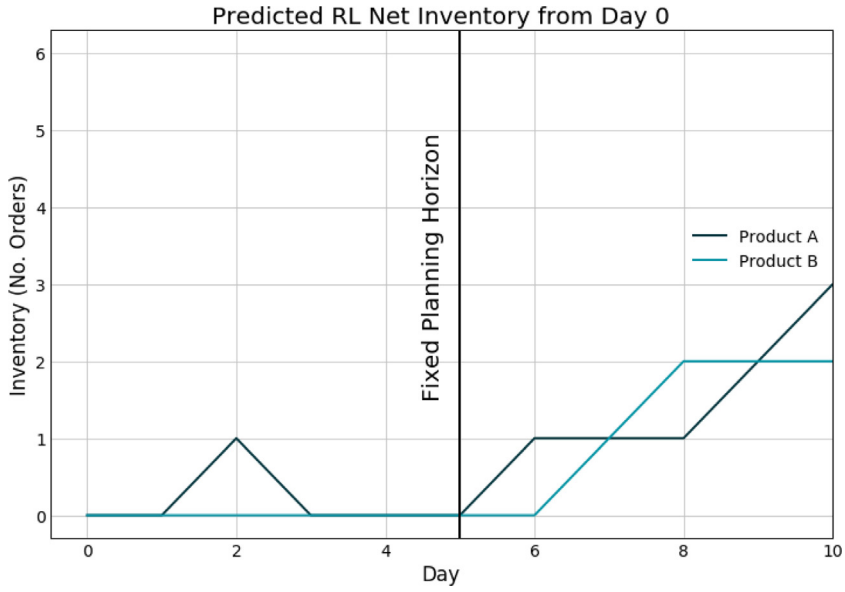

Fig. 14. RL schedule for simplified example from day 0.

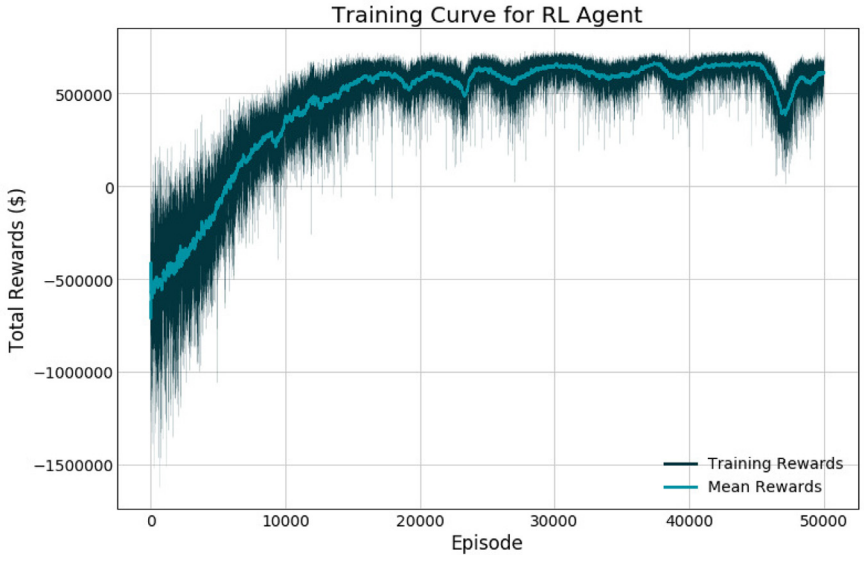

Fig. 15. Reinforcement learning training curve.

Table 11
Hyperparameter values used to train the DRL agent.

Parameter	Value	Description
Discount Factor (γ)	0.95	The scalar value that discounts future rewards back to the present.
Entropy regularization (β)	10^{-4}	Multiplier to scale the entropy regularization value. Higher values encourage more exploration and a potentially unstable policy, whereas lower values have a smaller effect and lead to more exploitation of the current policy.
Actor Learning Rate (α_π)	5×10^{-6}	Adjusts the step size for the actor network during stochastic gradient descent updates.
Critic Learning Rate (α_ϕ)	10^{-4}	Adjusts the step size for the critic network during stochastic gradient decent updates.
Batch Size	128	Number of batches of training data collected before updating network weights and biases.
Number of Hidden Nodes per Layer	512	The network trained using a feed-forward architecture with a constant number of nodes per hidden layer.
Number of Hidden Layers	12	Number of layers between the input layer and output layer which are fixed by the environment.

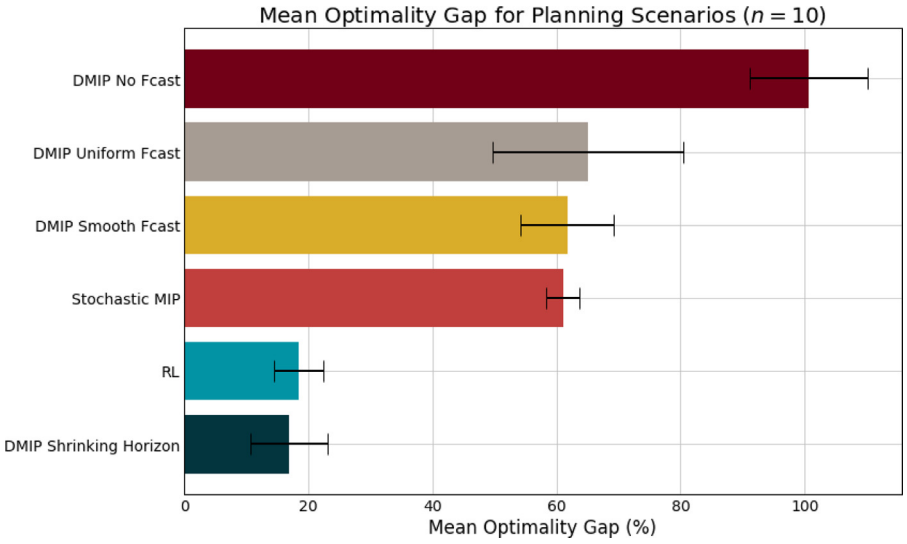

Fig. 16. Reinforcement learning compared to MILP and various forecast disaggregation methods as a percentage of optimality.

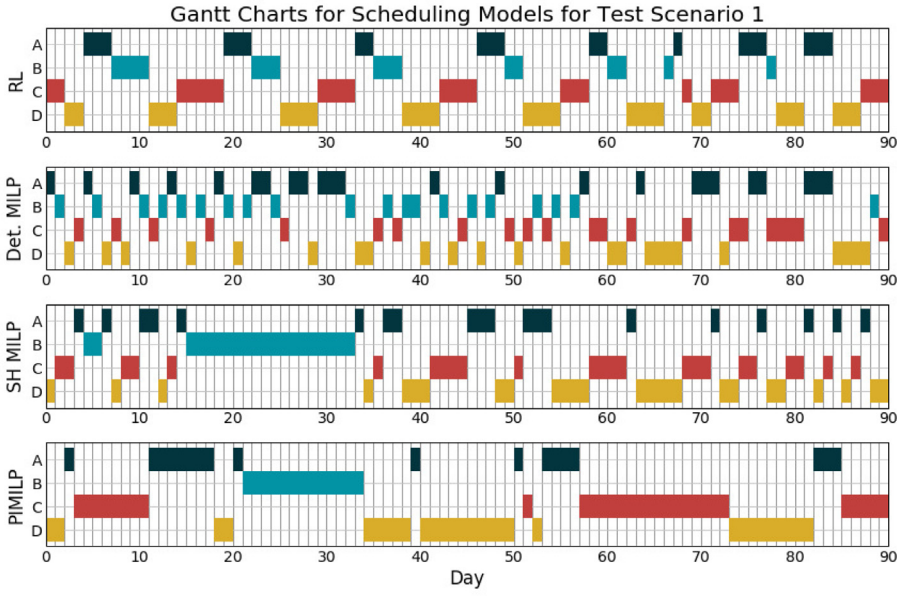

Fig. 17. Gantt chart for the RL agent, Deterministic MILP with Smoothed Demand Forecast, Deterministic MILP with Shrinking Horizon, and the Perfect Information Mixed-Integer Linear Program.

ference between each of the solution methods and the results of the perfect information model which acts as an upper bound.

The forecast disaggregation approaches greatly improved the performance over the MILP without a forecast yielding, each yielding an average gap of 62% versus the optimal compared to 100% for the model without a forecast. The stochastic program yielded solutions with a 61% gap, but much lower variance than the deterministic methods. The MILP with a shrinking horizon performed the best, reaching an average gap of 17%, however with larger variance than the DRL methods - 6.3% and 4.0% respectively - while the RL system generated schedules yielding a gap of only 19% versus the optimum. This indicates that the DRL system is capable of accounting for the uncertainty in the system by training on the simulation environment.

Examining the schedules (see Fig. 17) we see that the RL agent learns a cycle from product A to B to D to C, which yields 75 MT of lost prime production for the cycle. This is a similar cycle as the one employed by the deterministic MILP with smoothed demand forecast early in the simulation when demand is generally lower. The deterministic model has much shorter cycles, however, often switching each day, whereas the RL solution has longer campaigns as it moves through the product portfolio.

The regular switching of the deterministic model leads to lower inventory levels over the simulation horizon (Fig. 18), but it fails to build sufficient inventory levels in low-demand situations to account for increased demand later. The RL system's inventory build is comparably steady as it moves through its cycles. This leads to better product availability levels as shown in Fig. 19. Ultimately, this improved inventory management translates to fewer delays and late penalties for a better overall reward as depicted in Fig. 20. Thus, while the product availability of the RL system is higher, so are the inventory costs, which provide a buffer in the face of uncertainty.

Table 12 shows the different simulation times and model sizes for the different approaches explored in this paper. The reported times are the average simulation times to complete the 90-day test simulation. The comparison is not entirely similar given that the RL model is trained in advance on 50,000 different 90-day scenar-

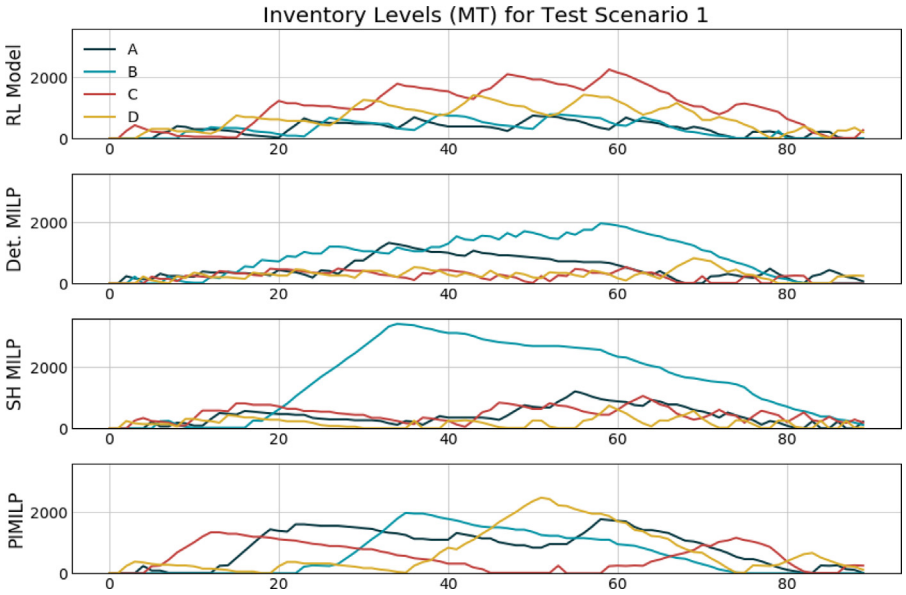

Fig. 18. Inventory over time for RL agent, Deterministic MILP with Smoothed Demand Forecast, Deterministic MILP with Shrinking Horizon, and the Perfect Information Mixed-Integer Linear Program.

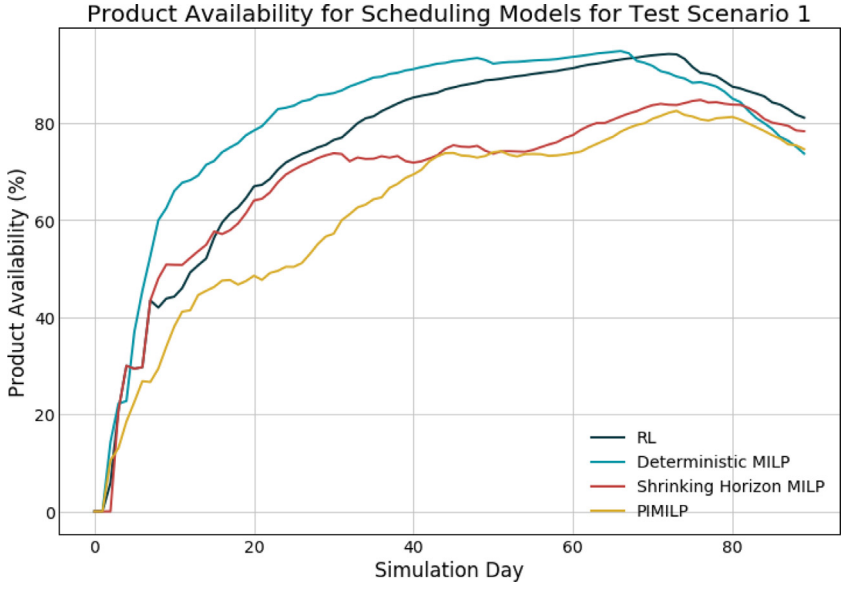

Fig. 19. Product availability for the RL agent, Deterministic MILP with Smoothed Demand Forecast, Deterministic MILP with Shrinking Horizon, and the Perfect Information Mixed-Integer Linear Program.

ios then deployed to generate schedules on the test scenarios. The computational expense, in this case, is paid up front while each additional inference during testing is computationally cheap. For example, to simulate one training episode, the RL agent takes about 0.8 s, but must do this thousands of times. For inference, the time is greatly reduced because the RL system only needs to make decisions for one episode and is not required to update the weights via the computationally intensive backpropagation algorithm. The MILP algorithms, on the other hand, are only required to solve each scenario once, but solve 90 smaller, sub-problems as the simulation moves through time.

5.2. Robustness

One of the well-known issues with model-free DRL is the lack of robustness if there are changes in the environment. This is particularly relevant for a system that is intended to be market-facing. If customers change their buying habits or prices shift significantly away from the training regime, performance will drop off.

For this reason, it is important to gauge the performance decline of the system when exposed to states and situations that the model has not seen in training. The training cases we subjected the models to begin in a period of low demand, and ramped up to a peak with higher demand in the last month versus the first and second. To test the robustness of the DRL solution, we can subject it to the same total demand and product mix, but distributed evenly over the simulation horizon to determine whether or not it performs as well (see Fig. 21 for an illustration). In short, we move from a training simulation with seasonality, to a testing environment without seasonal demand effects.

Comparing multiple scenarios under this new, uniform demand profile to the output of the PIMILP model, we can see the optimal-

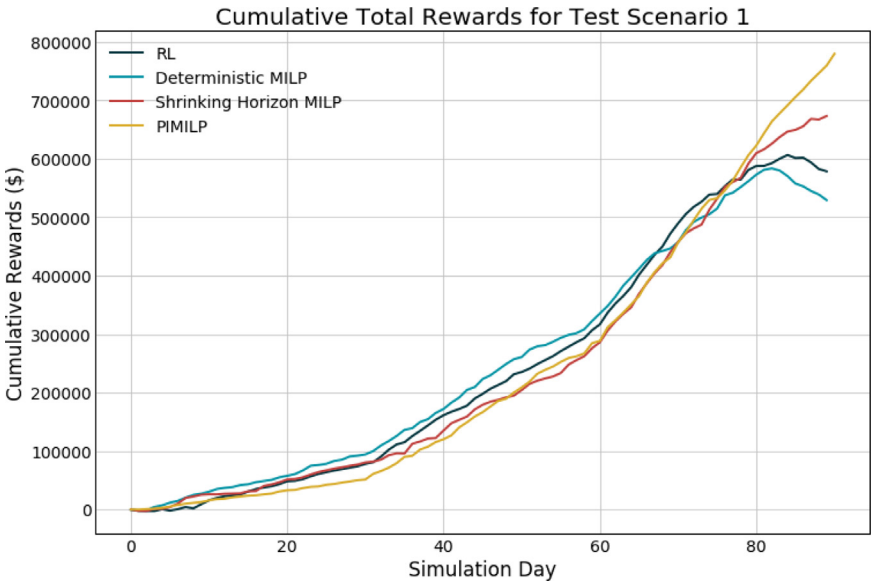

Fig. 20. Cumulative rewards for the RL agent, Deterministic MILP with Smoothed Demand Forecast, Deterministic MILP with Shrinking Horizon, and the Perfect Information Mixed-Integer Linear Program.

Table 12
Average model sizes and simulation times.

Model	Avg. Number of Variables	Avg. Number of Binary Variables	Avg. Sim Time (s)
Deterministic MILP with No Forecast	16,498	7931	83
Deterministic MILP with Uniform Forecast	33,048	15,746	197
Deterministic MILP with Smoothed Forecast	34,356	16,332	298
Shrinking Horizon MILP	272,028	133,985	4193
Stochastic MILP	918,069	432,913	21,913
Model	**Training Episodes**	**Avg. Train Time (s)**	**Avg. Sim Time (s)**
RL	50,000	40,248	0.021

ity gap and decrease from changing the environmental variables versus the training environment.

In Fig. 22, we show the original results for both the DRL model alongside one of the simpler and poorer performing models, DMILP with Smoothed Forecast, to demonstrate the difference in robustness between the DRL approach and the model based, MILP approach. With the new demand scenario, the gap versus the original demand environment was 19%, but now comes in at an 87% gap in the environment it has not seen. The deterministic MILP improves slightly between the two, rising from an average gap of 61% to 59% under this new demand scenario.

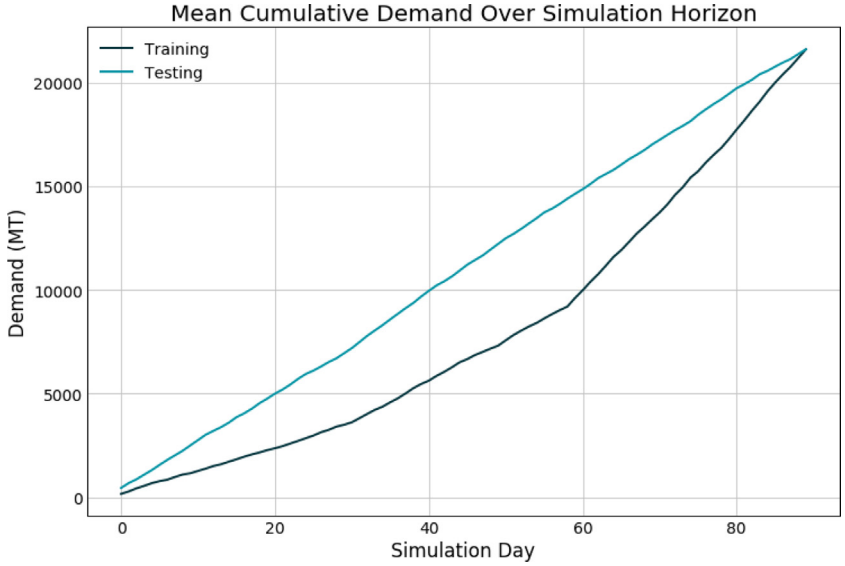

Fig. 21. Average cumulative demand profiles to test robustness of DRL solution. Note the cumulative testing demand increases at a constant rate while there are inflections in the training profile as demand increases to a seasonal peak.

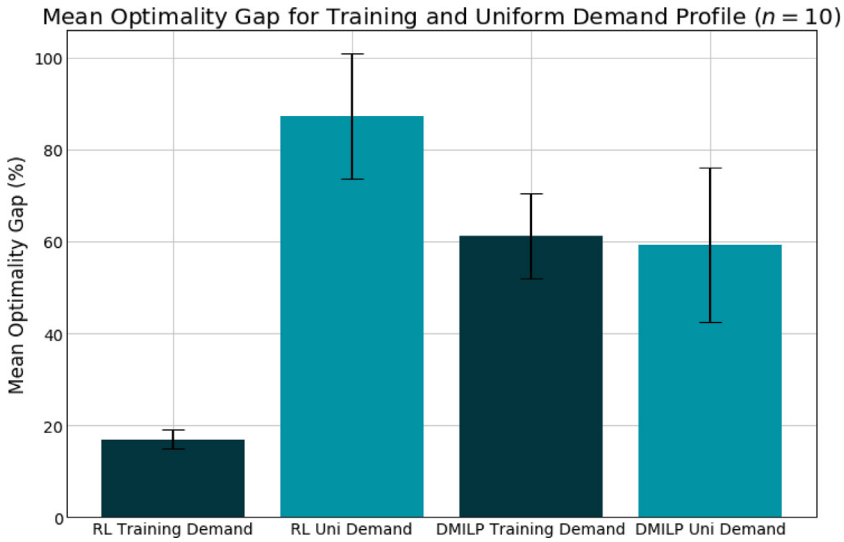

Fig. 22. Optimality gap for DRL and Deterministic MILP with Smoothed Forecast under training demand profile and under the uniform demand profile.

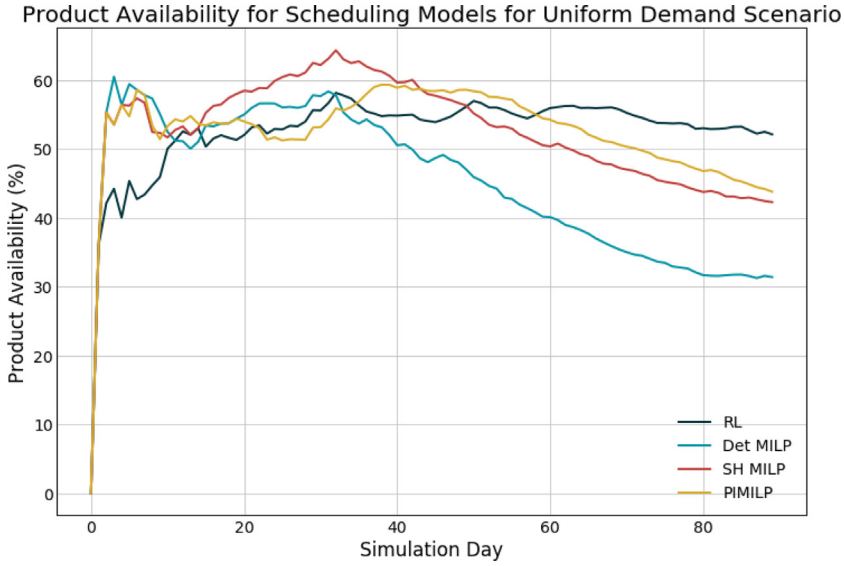

Fig. 23. Product availability for the RL agent, Deterministic MILP with Smoothed Demand Forecast and the Perfect Information Mixed-Integer Linear Program under the uniformly distributed demand scenarios.

Examining the details of a single run under this new scenario, we see that the DRL agent remains relatively consistent in its scheduling decisions exhibiting the same cyclic pattern as shown in Fig. 17, albeit with a shorter mean cycle time (8.3 days per cycle compared to 12.6 days). This reduction in cycle time, as well as poorer transition decisions as the agent finds itself facing states outside of its previous experience, increases the off-grade production by a factor of two. The loss in prime product further keeps the DRL agent from meeting its goals yielding poorer performance over the course of the simulation as it falls farther behind over time, despite maintaining relatively high product availability (see Figs. 23 and 24). Despite these limitations, the DRL model still outperforms the MILP without forecast disaggregation on the new demand profile by 29%.

The fragility of the RL policy to changes in the demand distribution is not uncommon - all of its training data was selected from a different distribution, thus by shifting it, we see a dramatic drop in performance. However, it is important to understand the limitations of RL algorithms. Being model-free, they have fewer capabilities when responding to large changes in the environment. These limitations can be addressed by re-training the system on the new distribution before deployment, or perhaps through development of hybrid models which can leverage the best of both worlds from the machine learning and MILP frameworks. Additionally, providing a system of trained agents that are more responsive to different regions of the state space or special situations outside of the assumed demand distribution, may prove fruitful as well.

While numerous other DRL algorithms exist in the literature (e.g. A3C, TRPO, PPO, Rainbow, and so forth) (see Mnih et al. (2016), Schulman et al. (2015), Schulman et al. (2017), Hessel et al. (2018) respectively), A2C was selected for its relative ease of implementation and the existence of fewer hyperparameters to tune. Moreover, good results were relatively easy to obtain. However, exploration of these other algorithms may yield more robust policies.

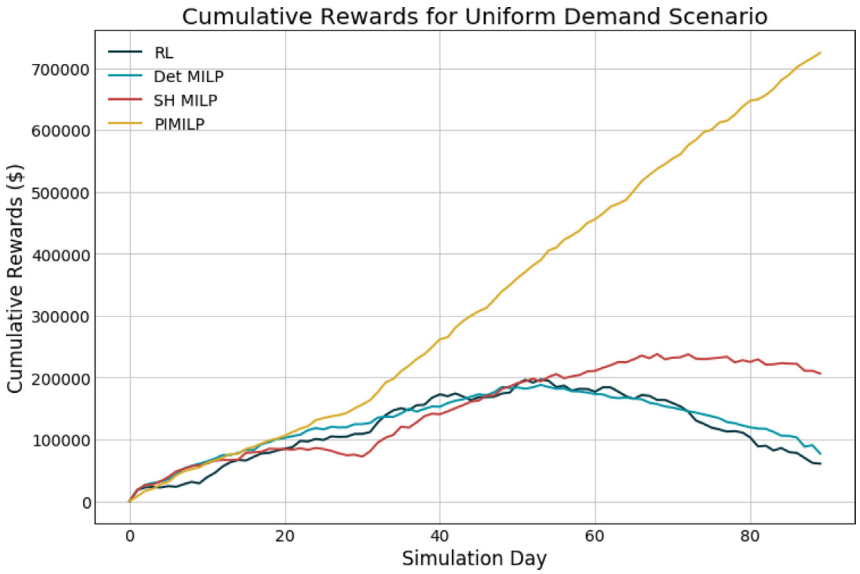

Fig. 24. Cumulative rewards for the RL agent, Deterministic MILP with Smoothed Demand Forecast and the Perfect Information Mixed-Integer Linear Program under the uniformly distributed demand scenarios.

6. Conclusions

A reinforcement learning model has been developed and proposed for dynamic scheduling of a single-stage multi-product reactor. The proposed approach provides a natural representation for capturing the uncertainty in a system and outperforms the MILP schedulers operating with a short receding time horizon for this reason. The incorporation of a forecast and changing to a longer lookahead by switching to the shrinking horizon MILP model enables this approach to perform better than all others.

DRL provides a viable and promising approach for chemical production scheduling. It is often easier to incorporate uncertain elements in simulation versus in a mathematical program. This uncertainty can be represented by the DRL agent such that, once the DRL agent is trained, it can produce schedules online, that are superior to more computationally intensive methods. The schedule can be generated almost instantly via a sequence of forward-passes through a deep neural network. This makes DRL for use-cases with regular and rapid rescheduling a valuable approach, given a system that can be simulated.

DRL has obvious advantages, however there are drawbacks as well. There is no guarantee of optimality with policy gradient methods apart from the REINFORCE algorithm as shown in Sutton et al. (1999), which can be shown to converge to local optimality in the limit. Additionally, DRL requires a large number of samples in order to learn a good policy, samples that typically must come through interaction with a simulator. This simulator must be efficient and can be difficult to build. Finally, the DRL approach lacks robustness to different, out of sample demand distributions than seen during training. Thus, if possible to combine DRL and model-based optimization methods. it may be possible to leverage the advantages of both approaches, leading to powerful scheduling algorithms.

One possible path towards integration may be to use a MILP model as an “oracle” during training, which the DRL agent can query when there is no clear action probability that dominates. In this respect, the DRL agent can invoke an expensive solver only when needed and learn from the solver to take those actions in the future without having to consult the oracle. Another possibility is using the DRL agent to restrict the search space in a stochastic programming algorithm. The agent, once trained, could assign low probability of receiving a high reward to certain actions in order to remove those branches and accelerate the search of the optimization algorithm.

Future research will explore possibilities for integrating DRL and optimization methods, examining DRL in a continuous time representation, and extending DRL to multi-stage and multi-agent systems for network optimization approaches. Additional sources of uncertainty may be considered as well such as maintenance and equipment reliability, changes in the prime rates of transitions, price fluctuations, and so forth to more accurately mirror the actual system in question.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Christian D. Hubbs: Conceptualization, Methodology, Software, Writing - original draft. **Can Li:** Conceptualization, Methodology. **Nikolaos V. Sahinidis:** Conceptualization, Methodology, Writing - review & editing, Supervision. **Ignacio E. Grossmann:** Conceptualization, Methodology, Writing - review & editing, Supervision. **John M. Wassick:** Conceptualization, Methodology, Supervision.

Acknowledgment

Funding from Dow Chemical is gratefully acknowledged.

References

Badgwell, T.A., Lee, J.H., Liu, K.-h., 2018. Reinforcement learning overview of recent progress and implications for process control. In: International Symposium on Process Systems Engineering, pp. 71–85.

Balasubramanian, J., Grossmann, I., 2003. Scheduling optimization under uncertainty an alternative approach. Comput. Chem. Eng. 27, 469–490. doi:10.1016/S0098-1354(02)00221-1.

Balasubramanian, J., Grossmann, I.E., 2004. Approximation to multistage stochastic optimization in multiperiod batch plant scheduling under demand uncertainty. Ind. Eng. Chem. Res. 43, 3695–3713. doi:10.1021/ie030308+.

Bellman, R., 1957. A Markovian decision process. 10.1007/BF02935461.

Bertsimas, D., Brown, D.B., Caramanis, C., 2011. Theory and applications of robust optimization. SIAM Rev. 53 (3), 464–501. doi:10.1137/080734510.

Birge, J., Louveaux, F., 1997. Introduction to Stochastic Programming. Springer, New York.
Bishop, C.M., 2006. Pattern Recognition and Machine Learning. Springer, Heidelberg.
Grossmann, I.E., Apap, R.M., Calfa, B.A., García-Herreros, P., Zhang, Q., 2016. Recent advances in mathematical programming techniques for the optimization of process systems under uncertainty. Comput. Chem. Eng. 91, 3–14. doi:10.1016/j.compchemeng.2016.03.002.
Gupta, D., Maravelias, C.T., 2016. On deterministic online scheduling: major considerations, paradoxes and remedies. Comput. Chem. Eng. 94, 312–330. doi:10.1016/j.compchemeng.2016.08.006.
Gurobi Optimization LLC, 2020. Gurobi optimizer reference manual.
Harjunkoski, I., Maravelias, C.T., Bongers, P., Castro, P.M., Engell, S., Grossmann, I.E., Hooker, J., Méndez, C., Sand, G., Wassick, J., 2014. Scope for industrial applications of production scheduling models and solution methods. Comput. Chem. Eng. 62, 161–193. doi:10.1016/j.compchemeng.2013.12.001.
Harjunkoski, I., Nyström, R., Horch, A., 2009. Integration of scheduling and control-theory or practice? Comput. Chem. Eng. 33 (12), 1909–1918. doi:10.1016/j.compchemeng.2009.06.016.
Hessel, M., Modayil, J., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Silver, D., 2018. Rainbow: combining improvements in deep reinforcement learning. AAAI.
Huang, K., Ahmed, S., 2009. The value of multistage stochastic programming in capacity planning under uncertainty. Oper. Res. 57 (4), 893–904. doi:10.1287/opre.1080.0623.
Jung, J.Y., Blau, G., Pekny, J.F., Reklaitis, G.V., Eversdyk, D., 2004. A simulation based optimization approach to supply chain management under demand uncertainty. Comput. Chem. Eng. 28 (10), 2087–2106. doi:10.1016/j.compchemeng.2004.06.006.
Lee, J.H., Shin, J., Realff, M.J., 2018. Machine learning: overview of the recent progresses and implications for the process systems engineering field. Comput. Chem. Eng. 114, 111–121. doi:10.1016/j.compchemeng.2017.10.008.
Lewis, F.L., Liu, D., 2013. Reinforcement Learning and Approximate Dynamic Programming for Feedback Control. Wiley, Hoboken, New Jersey.
Li, Y., 2017. Deep reinforcement learning: an overview, 1–70. 10.1007/978-3-319-56991-8_32
Li, Z., Ierapetritou, M., 2008. Process scheduling under uncertainty: review and challenges. Comput. Chem. Eng. 32 (4–5), 715–727. doi:10.1016/j.compchemeng.2007.03.001.
Lin, X., Janak, S.L., Floudas, C.A., 2004. A new robust optimization approach for scheduling under uncertainty: I. Bounded uncertainty. Comput. Chem. Eng. 28 (6–7), 1069–1085. doi:10.1016/j.compchemeng.2003.09.020.
Mao, H., Alizadeh, M., Menache, I., Kandula, S., 2016. Resource management with deep reinforcement learning. In: HotNets, pp. 50–56. doi:10.1145/3005745.3005750.
Martinez, Y., Nowe, A., Suarez, J., Bello, R., 2011. A reinforcement learning approach for the flexible job shop scheduling problem. In: Coello, C.A. (Ed.), Learning and Intelligent Optimization. Springer Verlag, pp. 253–262.
Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E., 1953. Equation of state calculations by fast computing machines. J. Chem. Phys. 21 (6), 1087–1092. doi:10.1063/1.1699114.
Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning, 48 arXiv:1602.01783doi:10.1177/0956797613514093.
Morinelly, J.E., Ydstie, B.E., 2016. Dual MPC with reinforcement learning. IFAC-PapersOnLine 49 (7), 266–271. doi:10.1016/j.ifacol.2016.07.276.
Mortazavi, A., Arshadi Khamseh, A., Azimi, P., 2015. Designing of an intelligent self-adaptive model for supply chain ordering management system. Eng. Appl. Artif. Intell. 37, 207–220. doi:10.1016/j.engappai.2014.09.004.
Oroojlooyjadid, A., Nazari, M., Snyder, L., Takáč, M., 2017. A deep Q-network for the beer game: a reinforcement learning algorithm to solve inventory optimization problems1–38. arXiv:1708.05924
Palombarini, J., Barsce, J. C., Martinez, E., 2018. Generating rescheduling knowledge using reinforcement learning in a cognitive architecture 2 real-time rescheduling in soar cognitive architecture. arXiv preprint.
Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A., 2017. Automatic differentiation in PyTorch. NIPS Autodiff Workshop.
Riedmiller, S., Riedmiller, M., 1999. A neural reinforcement learning approach to learn local dispatching policies in production scheduling. In: IJCAI International Joint Conference on Artificial Intelligence, vol. 2, pp. 764–769.
Sahinidis, N.V., 2004. Optimization under uncertainty: State-of-the-art and opportunities. In: Computers and Chemical Engineering, vol. 28, pp. 971–983. doi:10.1016/j.compchemeng.2003.09.017.
Sand, G., Engell, S., 2004. Modeling and solving real-time scheduling problems by stochastic integer programming. Comput. Chem. Eng. 28 (6–7), 1087–1103. doi:10.1016/j.compchemeng.2003.09.009.
Schneider, J.G., Boyan, J.A., Moore, A.W., 1998. Value function based production scheduling. In: International Conference on Machine Learning, Madison, Wisconsin, USA, vol. 15, pp. 522–530.
Schulman, J., Levine, S., Moritz, P., Jordan, M. I., Abbeel, P., 2015. Trust region policy optimization. arXiv:10.3917/rai.067.0031.
Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms John. arXiv:10.1016/j.conb.2007.07.004.
Shobrys, D.E., White, D.C., 2002. Planning, scheduling and control systems: why cannot they work together. Comput. Chem. Eng. 26 (2), 149–160. doi:10.1016/S0098-1354(01)00737-2.
Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Driessche, G.V.D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529 (7585), 484–489. doi:10.1038/nature16961.
Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., Hassabis, D., 2017. Mastering the game of Go without human knowledge. Nature 550 (7676), 354–359. doi:10.1038/nature24270.
Singh, S. P., Sutton, R. S., 1996. Reinforcement learning with replacing elibility traces.
Stockheim, T., Schwind, M., Wolfgang, K., 2003. A reinforcement learning approach for supply chain management. 1st European Workshop on Multi-Agent Systems.
Sutton, R., Barto, A., 2018. Reinforcement Learning: An Introduction, second ed. MIT Press, Cambridge, Massachusetts.
Sutton, R.S., Mcallester, D., Singh, S., Mansour, Y., 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. Adv. Neural Inf. Process. Syst. 12, 1057–1063. doi:10.1.1.37.9714
Sutton Richard, S., 1988. Learning to predict by the method of temporal differences. Mach. Learn. 3 (1), 9–44. doi:10.1023/A:1018056104778.
Watkins, C.J.C.H., 1989. Learning from Delayed Rewards. University of Cambridge Ph.D. thesis.
Williams, R.J., 1992. Simple statistical gradient following algorithms for connectionist reinforcement learning. Mach. Learn. 8 (3–4), 229–256.
Zhang, W., Dietterich, T., 1995. A reinforcement learning approach to job-shop scheduling. In: International Joint Conference on Artificial Intelligence, pp. 1114–1120.