Reinforcement learning approaches for the stochastic discrete lot-sizing problem on parallel machines

Reinforcement learning approaches for the stochastic discrete lot-sizing problem on parallel machines

Leonardo Kanashiro Felizardo[a], Edoardo Fadda[b], Emilio Del-Moral-Hernandez[a], Paolo Brandimarte[b]

[a]*Escola Politécnica da Universidade de São Paulo, Av. Prof. Luciano Gualberto, 380, São Paulo*
[b]*Dipartimento di Scienze Matematiche Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino*

Abstract

This paper addresses the stochastic discrete lot-sizing problem on parallel machines, which is a computationally challenging problem also for relatively small instances. We propose two heuristics to deal with it by leveraging reinforcement learning. In particular, we propose a technique based on approximate value iteration around post-decision state variables and one based on multi-agent reinforcement learning. We compare these two approaches with other reinforcement learning methods and more classical solution techniques, showing their effectiveness in addressing realistic size instances.

Keywords: Dynamic programming, Stochastic programming, Multi-agent systems, Machine learning, Reinforcement Learning

1. Introduction

Production planning aims to determine the best allocation of limited production resources to meet demand over a time horizon. In this context, lot-sizing problems are among the most important and most addressed decision problems. They have been studied under two settings: big-time buckets and small-time

Email addresses: leonardo.felizardo@usp.br (Leonardo Kanashiro Felizardo), edoardo.fadda@polito.it (Edoardo Fadda), emilio.delmoral@usp.br (Emilio Del-Moral-Hernandez), paolo.brandimarte@polito.it (Paolo Brandimarte), leonardo.felizardo@usp.br (Leonardo Kanashiro Felizardo)

buckets. In the first setting, the principal model is the *capacitated lot sizing problem*. It considers time steps whose length allows the production of different items on a machine; thus, it does not address the sequencing of production activities. In contrast, the principal model in the second setting is the *discrete lot sizing problem* (DLSP). It splits the (macro) periods of the capacitated lot sizing problem into several (micro) periods that do not allow for the production of different items on the same machine. Therefore, only one item may be produced per period, and if production starts, it uses the full capacity. This is the so-called *all-or-nothing* assumption. Usually, the time periods addressed in the discrete setting correspond to hours, shifts, or days.

This paper focuses on the small-time buckets setting, where production tries to satisfy a stochastic item demand, which we assume is independently and identically distributed over time. We consider a plant with parallel machines, as is usually the case, for example, in the semiconductor industry [1]. When a machine starts to produce one item, we incur both a *setup cost* and a *setup loss*. While *setup costs* accounts for the economic cost of starting production in terms of material and workforce, *setup losses* accounts for the decrease in the number of items produced due to the portion of the time bucket used for the setup operations. We assume both setup cost and setup losses to be sequence-independent. Therefore, the problem that we address is the *stochastic discrete lot sizing and scheduling problem with sequence-independent setup times and costs* ($SDLSP_{\text{setup}}$). This optimization model is challenging to solve in a realistic setting due to its dimension (number of products and machines) and the presence of binary variables (produce or do not produce an item on a machine). Therefore, heuristic methods are required.

Research works have been carried out to deal with stochastic DLSP employing a scenario tree, resulting in multi-stage stochastic mixed-integer programming. However, scenario trees are limited in capturing uncertainty while preserving computational tractability, which motivates the development of heuristics to obtain a good quality solution in a reasonable time [1].

Two interesting alternatives to multi-stage stochastic programming are ap-

proximate dynamic programming (ADP) and deep reinforcement learning (DRL). These techniques have been shown to be effective in solving complex optimization problems using statistical learning to approximate the optimal policy or value function. Nevertheless, their applications to the $SDLSP_{\text{setup}}$ setting have been limited. This paper fills this gap by proposing and comparing different dynamic programming-based heuristics for solving $SDLSP_{\text{setup}}$. Moreover, we evaluate the performance of these heuristics and provide insight into their relative strengths and limitations. More in detail, the contributions of this paper are:

- We provide an open-source environment model of inventory management simulation for RL application, which can support future researchers in reproducing and extending this work.
- We develop two innovative heuristics; one based on a value function approximation, and one based on cooperative multi-agent reinforcement learning.
- We compare the performance of these heuristics using out-of-sample simulations on different instances.

The paper is organized as follows: Section 2 presents the literature review about the problem and solution techniques, Section 3 specifies the mathematical model of the $SDLSP_{\text{setup}}$, Section 4 presents the proposed solution methods in detail, and Section 5 reports the results of the computational experiments. Finally, Section 6 presents the conclusions of the work.

2. Literature review

While lot sizing problems considering big time buckets have been extensively tackled in the literature [2, 3, 4, 5], problems considering small time buckets have been addressed much less [6, 7]. The basic problem type in the small time bucket setting is the DLSP, introduced in [8]. It models settings where decisions

must be made on a daily or even shift-wise basis. It combines both lot sizing (medium range planning) and job scheduling (short range planning) problems [9].

The DLSP has been formulated with both deterministic and stochastic models. Several papers have addressed deterministic DLSP [10, 9, 11, 12, 13, 14, 15, 16] on which all parameters (like demand, production costs, and capacities) are known in advance and remain constant throughout the planning horizon. The deterministic DLSP is often approached with heuristic methods to find good solutions efficiently, as exact methods cannot be used in real-size instances. Moreover, the deterministic DLSP has been applied in several industrial settings such as glass container production[17, 18], food and animal-feed industry [19, 20], furnace scheduling [21], soft drink production [22], textile and fiberglass industries [23], pharmaceutical manufacturing [24], amongst many others.

In a real setting, assuming that all parameters are deterministic is often unrealistic. For example, the demand or the machine production rate is subject to uncertainty and variability. When uncertainty is factored into the DLSP, the problem becomes computationally more challenging, and the deterministic solutions lead to bad results [1]. The first paper addressing the stochastic DLSP is [1]. The authors consider the variability in the machine production rate and present a solution by modeling the problem as multi-stage stochastic programming. They also suggest a fix-and-relax heuristic to achieve high-quality solutions within a reasonable time. While [1] is the only research paper that applies multi-stage methods to this problem, uncertainty has also been tackled with a chance-constraint approach in [25]. The authors address a multi-level lot-sizing and scheduling problem with sequence-dependent setup times, stochastic demands, and processing times. Instead, two-stage stochastic programming has been used to deal with a single-level version of the problem in [26]. The authors present a numerical study based on a few scenarios generated by moment matching and scenario reduction techniques.

Despite the significance of stochastic models and the integration of lot-sizing decisions within finite planning horizons, the literature about stochastic DLSP

is rather scarce [27, 28].

Various optimization techniques can be employed to tackle the stochastic DLSP effectively, with reinforcement learning (RL) emerging as a promising yet underexplored approach. Together with RL another important branch of research is deep reinforcement learning (DRL) , which, in contrast to RL, models policy or value function by means of artificial neural network (ANN). One of the first works to employ DRL is [29], where the authors use an ANN to approximate the value function of the $SDLSP_{\text{setup}}$ for the instances with three items and a single machine. Besides using a small plant, the authors do not consider setup loss nor compare the techniques with other stochastic policies. The same setting is used in [30], where the authors address uncertainty in demands and processing times. They proposed a Q-learning algorithm improved through heuristics, which modified the ϵ-greedy method to optimize the learning of the RL agent. Despite the good results, Q-learning algorithms do not scale well to large problem instances. This is proved also by [31], where the authors explore the use of Deep-Q-Network for single-machine problems within the constraints of a small-scale setting.

A competitive technique in this setting is Proximal Policy Optimization (`PPO`). In [32], the authors present results showing the superiority of `PPO` over other RL methods, as the ones in [29], although requiring more comprehensive training and hyperparameter optimization. Moreover, in [33], the authors investigate the scalability of `PPO` for large problem instances without set-up loss, bridging the gap between theoretical models and real-world industrial applications. Our work aims to build upon these developments, further advancing the field of stochastic DLSP employing ADP and DRL-related methods.

3. Mathematical models

We consider a production environment composed of a set $\mathcal{M} = \{1, \ldots, M\}$ of parallel machines. Each machine can be idle, or it may produce an item. We call the set of possible states of machine m, $\mathcal{I}(m)$, and we call the set of all the

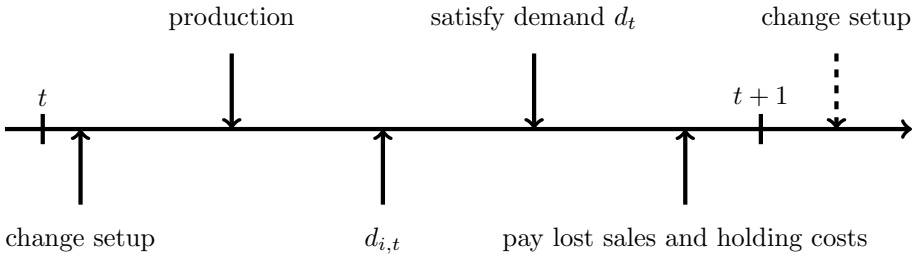


Figure 1: Flow of events.

items that can be produced $\mathcal{I} = \{1, \ldots, I\}$. For each item i, we call $\mathcal{M}(i)$ the set of machines that can produce it.

The problem consider set $\mathcal{T} = \{0, \ldots, T\}$ of time periods. Immediately before $t = 0$, each machine m has an initial state $i_0 \in \mathcal{I}(m)$, we call the set of all pairs (m, i_0), $\mathcal{C} \subseteq \mathcal{M} \times \mathcal{I}$.

At the beginning of each time period, the decision maker assigns to each machine a possible state (i.e., an element of $\mathcal{I}(m)$). If necessary, the machine configurations are changed and the setup cost $f_{i,m}$ for the production of item i is incurred. Then, each non-idle machine produces $p_{i,m}$ items if no setup occurs. Instead, if a setup occurs, the production will decrease by a number $c_{i,m}$ of items. After production, on-hand inventory is updated the demand $d_{i,t}$ becomes known, and the available items are used to satisfy it. Finally, both lost sales and holding costs are computed (in this paper, we rule out the possibility of backorder). We call h_i and l_i, the unit holding cost and lost sale penalty for item i, respectively. The flow of events is represented in Figure 1.

Using this notation, we present three mathematical models for the problem. First, in Subsection 3.1 we present the deterministic model, whose purpose is to clarify our assumed problem setting. Then, in Subsection 3.2 we model demand uncertainty using a scenario tree, defining a multistage stochastic programming model. Finally, we model stochasticity by means of a Markov Decision Process, in Subsection 3.3, which paves the way for the application of ADP/RL methods.

3.1. Deterministic Model

The presented model is similar to the one in [8], but it introduces setup losses. Let us define the sets:

- $\mathcal{I}_0(m)$ the set of items that can be produced by machine m, i.e., $\mathcal{I}_0(m)$ is equal to $\mathcal{I}(m)$ without the idle state.
- $\mathcal{T}^+ = \mathcal{T} \setminus \{0\}$.

Moreover, we define the following variables:

- $I_{i,t}$ is the inventory of item i at the end of time period t.
- $x_{i,m,t}$, set to 1 if machine m is producing item i at time t.
- $\delta_{i,m,t}$: binary variable set to 1 if a setup on machine m is carried out between time $t-1$ and time t, in order to start production of item i.
- $z_{i,t}$: the lost sales of item i at time t.

Here, we have implicitly defined variables for compatible item–machine pairs. The deterministic version of the problem can be formulated as follows:

$$\min \sum_{i=1}^{I} \left[\sum_{m \in \mathcal{M}(i)} \sum_{t=0}^{T} f_{i,m}\delta_{i,m,t} + \sum_{t=1}^{T} (h_i I_{i,t} + l_i z_{i,t}) \right] \tag{1}$$

$$\text{s.t. } I_{i,t} - z_{i,t} = I_{i,t-1} + \sum_{m \in \mathcal{M}(i)} (p_{i,m} x_{i,m,t-1} - c_{i,m}\delta_{i,m,t-1}) - d_{i,t} \qquad \forall i \in \mathcal{I},\ t \in \mathcal{T}^+ \tag{2}$$

$$\sum_{i \in \mathcal{I}_0(m)} x_{i,m,t} \leq 1 \qquad \forall m \in \mathcal{M}, t \in \mathcal{T} \tag{3}$$

$$x_{i,m,t} = 0 \qquad \forall i \notin \mathcal{I}_0(m), t \in \mathcal{T} \tag{4}$$

$$\delta_{i,m,t} \geq x_{i,m,t} - x_{i,m,t-1} \qquad \forall\ m \in \mathcal{M},\ i \in \mathcal{I}_0(m)\ t \in \mathcal{T}^+ \tag{5}$$

$$x_{i,m,t-1} - x_{i,m,t} + \varepsilon \leq (1+\varepsilon)(1-\delta_{i,m,t}) \qquad \forall\, m \in \mathcal{M},\ i \in \mathcal{I}_0(m),\ t \in \mathcal{T}^+ \tag{6}$$

$$\delta_{i,m,0} \geq x_{i,m,0} - \mathbb{1}_{(m,i)\in\mathcal{C}} \qquad \forall\, m \in \mathcal{M},\ i \in \mathcal{I}_0(m) \tag{7}$$

$$\mathbb{1}_{(m,i)\in\mathcal{C}} - x_{i,m,0} + \varepsilon \leq (1+\varepsilon)(1-\delta_{i,m,0}) \qquad \forall\, m \in \mathcal{M},\ i \in \mathcal{I}_0(m) \tag{8}$$

$$I_{i,0} = \bar{I}_{i0} \qquad \forall i \in \mathcal{I} \tag{9}$$

$$I_{i,t} \in [0, I_{\max}], z_{i,t} \in \mathbb{R}^+ \qquad \forall i \in \mathcal{I},\ t \in \mathcal{T} \tag{10}$$

$$\delta_{i,m,t}, x_{i,m,t} \in \{0,1\} \qquad \forall\, i \in \mathcal{I}, m \in \mathcal{M}, t \in \mathcal{T}, \tag{11}$$

where ε is a small constant required for the logic constraints, and $\mathbb{1}_{(m,i)\in\mathcal{C}}$ is equal to 1 if $(m,i) \in \mathcal{C}$, 0 otherwise.

The objective function (1) is the sum of the setup, holding, and lost sales costs. This last term is considered even if the model is deterministic since for some instances, it may be convenient not to satisfy the demand in order not to incur high inventory costs. An example occurs when the productions $p_{i,m}$ are big while the demands are very low. Therefore, it may be better to lose some sales instead of producing the new batch and storing it. Constraints (2) force the inventory balance, and constraints (3) enforce the *all-or-nothing* assumption. Notice that this assumption prevents a machine from producing different items during the same time step, but it permits different machines to produce the same item. Constraints (4) forbid to produce items that a machine cannot produce. Constraints (5), and (6) impose that $\delta_{i,m,t}$ must be one if and only if there is a change in the machine setting. In particular, constraints (6) are needed to prevent the model from setting $\delta_{i,m,t} = 1$ to reduce the production and better match the demand even if a setup is not required. Finally, constraints (7), and (8) force the initial setup, and constraints (10), (11) impose the type

of variables. For the sake of simplicity, we assume that all the items have the same maximum inventory $I_{\max}$. It is worth noting that model (1) - (11) has full knowledge of the demand along all the time horizons. This knowledge is usually not available in practice. Despite this drawback, we will use this model as a benchmark. In the following, we refer to model (1) - (11) as *perfect information* (`PI`).

3.2. Multi-stage Stochastic Optimization

Model (1) - (11) can be modified to consider stochastic demand. A typical choice to model uncertainty is through a scenario tree. Following the notation in [34], we define:

- The set of nodes in the scenario tree is $\mathcal{N}$, and $\mathcal{N}^+ = \mathcal{N} \setminus \{0\}$.
- $p(n)$ the parent of node $n \in \mathcal{N}^+$.
- $\pi^{[n]}$ the unconditional probability of node n ($\pi^{[0]} = 1$).
- $d_i^{[n]}$ the demand for item i at node $n \in \mathcal{N}$.

We refer to the number of children at each node at a specific tree level as the *branching factor*. For instance, the branching factor $[2, 2, 2]$ identifies a binary tree over four-time instants, including the current time corresponding to the root node and eight scenarios.

The decision variable of the stochastic version of the model (1)-(11) have the superscript $\cdot^{[n]}$ instead of the subscript $\cdot_t$ to identify the node $n \in \mathcal{N}$ to which they refer. For example, the inventory of item i in node n will be $I_i^{[n]}$, etc. The multi-stage version of model (1) - (11) is:

$$\min \quad \sum_{i=1}^{I} \left[\sum_{n \in \mathcal{N}} \pi^{[n]} \Big(\sum_{m \in \mathcal{M}(i)} f_{i,m} \delta_{im}^{[n]} \Big) + \sum_{n \in \mathcal{N}^+} \pi^{[n]} (h_i I_i^{[n]} + l_i z_i^{[n]}) \right] \tag{12}$$

$$\text{s.t.} \quad I_i^{[n]} - z_i^{[n]} = I_i^{[p(n)]} + \sum_{m \in \mathcal{M}(i)} (p_{i,m} x_{i,m}^{[p(n)]} - c_{i,m} \delta_{i,m}^{[p(n)]}) - d_i^{[n]} \quad \forall i \in \mathcal{I},\ n \in \mathcal{N}^+ \tag{13}$$

$$\sum_{i \in \mathcal{I}_0(m)} x_{i,m}^{[n]} \leq 1 \qquad \forall m \in \mathcal{M},\ n \in \mathcal{N} \tag{14}$$

$$x_{i,m}^{[n]} = 0 \qquad \forall i \notin \mathcal{I}_0(m), n \in \mathcal{N} \tag{15}$$

$$\delta_{im}^{[n]} \geq x_{i,m}^{[(n)]} - x_{i,m}^{[p(n)]} \qquad \forall\ m \in \mathcal{M},\ i \in \mathcal{I}_0(m),\ n \in \mathcal{N}^+ \tag{16}$$

$$x_{i,m}^{[p(n)]} - x_{i,m}^{[n]} + \varepsilon \leq (1+\varepsilon)(1 - \delta_{i,m}^{[n]}) \qquad \forall\ m \in \mathcal{M},\ i \in \mathcal{I}_0(m),\ n \in \mathcal{N}^+ \tag{17}$$

$$\delta_{i,m}^{[0]} \geq x_{i,m}^{[0]} - \mathbb{1}_{(m,i) \in \mathcal{C}} \qquad \forall\ m \in \mathcal{M},\ i \in \mathcal{I}_0(m) \tag{18}$$

$$\mathbb{1}_{(m,i) \in \mathcal{C}} - x_{i,m}^{[0]} + \varepsilon \leq (1+\varepsilon)(1 - \delta_{i,m}^{[0]}) \qquad \forall\ m \in \mathcal{M},\ i \in \mathcal{I}_0(m), \tag{19}$$

$$I_i^{[0]} = \bar{I}_{i0} \qquad \forall i \in \mathcal{I} \tag{20}$$

$$I_i^{[n]} \in [0, I_{\max}], z_i^{[n]} \in \mathbb{R}^+ \qquad \forall i \in \mathcal{I},\ n \in \mathcal{N} \tag{21}$$

$$\delta_m^{[n]}, x_{i,m}^{[n]} \in \{0,1\} \qquad \forall m \in \mathcal{M}, i \in \mathcal{I}, n \in \mathcal{N}. \tag{22}$$

The objective function (12) is the expected value of the sum of all the costs (setup costs, lost sales, and holding costs). While constraints (13),(14), (15), (16), (17), (18), and (19) are the stochastic counterpart of constraints (2), (3), (4), (5), (6), (7), and (8), respectively. It is worth noting that given the solution of the model, the action taken by the agent at time t is $\mathbf{X}_t = [x_{i,m}^{[0]}]_{m \in \mathcal{M}, i \in \mathcal{I}}$. In the following, model (12) - (22) will be addressed as *multi-stage* (`MS`).

3.3. Markov Decision Process Model

Instead of using a scenario tree to model uncertainty, it is possible to use *Markov Decision Process*. This choice requires defining *states*, *actions*, *immedi-*

ate costs, and *state transitions*.

The *state* is described by the available inventory and by the state of the machines. Formally, we use the notation $\mathbf{S}_t = [\mathbf{I}_t, \mathbf{M}_t]$, where

- $\mathbf{I}_t = [I_{1,t}, \ldots, I_{I,t}]$ is the array of inventory levels.
- $\mathbf{M}_t = [m_{1,t}, \ldots, m_{M,t}]$ is the array of machines configurations, where $m_{m,t} \in \mathcal{I}(m)$ is the configuration of machine m.

Linked to the state, we define $\mathbf{N}_t = [n_{1,t}, \ldots, n_{I,t}]$ to be a vector containing the number of machines producing a given item.

The *action* is described by $\mathbf{X}_t = [x_{i,m,t}]_{m \in \mathcal{M}, i \in \mathcal{I}}$, where $x_{i,m,t} = 0, \forall\ i \notin \mathcal{I}_0(M)$

The *immediate costs* is:

$$C_t\left(\mathbf{X}_t, \mathbf{S}_t\right) = \sum_{m \in \mathcal{M}(i)} \sum_{i=1}^{I} f_i \delta_{i,m,t} + \sum_{i=1}^{I} h_i \left[I_{i,t} + \sum_{m \in \mathcal{M}(i)} \left(p_{i,m} x_{i,m,t} - c_{i,m} \delta_{i,m,t}\right) - d_{i,t} \right]^+ + \sum_{i=1}^{I} l_i \left[d_{i,t} - I_{i,t} + \sum_{m \in \mathcal{M}(i)} \left(p_{i,m} x_{i,m,t} - c_{i,m} \delta_{i,m,t}\right) \right]^+, \tag{23}$$

where the first term accounts for the setup cost, the second for the inventory, the third for possible lost sales, and where $[y]^+ \doteq \max\{y, 0\}$. Note that the first term of the immediate costs is deterministic, while the second and third ones are stochastic (since, at time t, $d_{i,t}$ is unknown).

Finally, the *state transition* equation for inventory, under our lost sales hypothesis, is

$$I_{i,t+1} = \left[I_{i,t} + \sum_{m \in \mathcal{M}(i)} \left(p_{i,m} x_{i,m,t} - c_{i,m} \delta_{i,m,t}\right) - d_{i,t} \right]^+, \tag{24}$$

while the *state transition* equation for the setup state is straightforward.

4. Solution methods

This section presents the solution methods used to tackle the $SDLSP_{\text{setup}}$ problem. In Subsection 4.1 we describe a decision rule which will be used as

benchmark as well as a component for other heuristics. In Subsection 4.2, we present a new approximate dynamic programming method. Finally, in Subsection 4.3, we describe actor-critic techniques used as a benchmark, and in Subsection 4.4, we present a new model-free reinforcement learning technique leveraging a multi-agent approach.

4.1. Decision Rule

To develop a decision rule for $SDLSP_{\text{setup}}$, we consider the expected run-out times (which depend on the inventory state and can be easily estimated if the demands are i.i.d.) and the current machine setup states [35].

The main procedure is divided into three steps: First, we select all items with an expected run-out time below a threshold. Then, we compute a priority indicator. Finally, we assign to each machine a state. The algorithm is shown in Pseudo-code 1.

The initial step of the heuristic is to compute the average demand $\bar{d}_i$ of each item employing Monte Carlo methods. Then, each item's expected run-out time is computed by dividing the available inventory by the average demand.

All items with run-out time smaller than α_1 are considered eligible for production. The priority for each of these items is computed as a combination of the following:

- The ratio between the lost sales costs and the expected run-out time: It prioritizes big lost sales costs and small expected run-out-time.
- The number of machines producing that item: the higher this number, the smaller the priority.
- The ratio between the average demand and the maximum possible production: If this ratio is high, we need more machines to satisfy the average demand. Therefore, the priority is higher.

We call the weights α_2, α_3, α_4, and the weighted sum *priority*. Without loss of generality, we set $\alpha_2 = 1$.

1 Pseudo-code: Decision Rule

eligible_items = []

compute average demand $\bar{d}$ by using Monte Carlo method

expected_runout $\leftarrow [I_{i,t}/\bar{d}_i]_{i\in\mathcal{I}}$

for $i \in \mathcal{I}$ **do**

  if *expected_runout[i]* $\leq \alpha_1$ **then**

    priority $\leftarrow \alpha_2 \frac{l_i}{\text{expected\_runout[i]}} + \alpha_3 N_i + \alpha_4 \frac{\bar{d}_i}{\max_m p_{im}}$

    eligible_items.add((i, priority))

sort eligible_item with respect to priority

$\mathbf{x} \leftarrow [0, \ldots, 0]$ # initialize the action by setting all machine to idle

for $i \in$ *eligible_items* **do**

  $M \leftarrow$ set of machine producing i

  if $M \neq \emptyset$ **then**

    $\hat{m} \leftarrow \max_{m\in M} f_{i,m}$

  else

    $\hat{m} \leftarrow$ free machine with smallest $\frac{f_{i,m}}{p_{i,m}-c_{i,m}}$

  $\mathbf{x}[\hat{m}] \leftarrow i$

for $m \in \mathcal{M}$ **do**

  if $\mathbf{x}[\hat{m}] = 0$ *and* $\boldsymbol{M}_t[m] \neq 0$ **then**

    $I_i^x \leftarrow I_i + p_{i,m}$

    if $f_{\boldsymbol{M}_t[m],m} \geq \alpha_4 h \left(\left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor + 1 \right) \left(I_i^x - \frac{\bar{d}_i}{2} \left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor \right)$ **then**

      $\mathbf{x}[m] \leftarrow \mathbf{M}_t[m]$

The list of eligible items is then sorted according to their priority.

We initialize the action by setting all the machines to idle. Then, starting with the item with higher priority and moving to the one with less priority, we check if a machine is producing that item. If more machines produce the item, we keep producing it with the machine with the highest setup costs. Instead,

if no machine produces the item, we start producing it with the free machine with the smallest ratio between production and setup costs.

Once we end the list of eligible items, we focus on setting activated machines to idle. We leave them to produce if the setup cost is greater than a threshold α_5 multiplied by an estimation of the expected inventory cost computed as:

$$\sum_{t=0}^{\lfloor \frac{I_i^x}{\bar{d}_i} \rfloor} h(I_i + p_{i,m} - \bar{d}_i t) \tag{25}$$

where, $I_i^x = I_i + p_{i,m}$. This expression sums the holding costs considering a depletion equal to the average demand (i.e., $\bar{d}_i$) and considering that production happens (i.e., the initial inventory is $I_i + p_{i,m}$). We can rewrite Eq. (25) as

$$\begin{aligned}
&\sum_{t=0}^{\lfloor \frac{I_i^x}{\bar{d}_i} \rfloor} h(I_i + p_{i,m} - \bar{d}_i t) = hI_i \left(\left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor + 1 \right) - h\bar{d}_i \sum_{t=0}^{\lfloor \frac{I_i^x}{\bar{d}_i} \rfloor} t \\
&hI_i \left(\left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor + 1 \right) - h\bar{d} \frac{\lfloor \frac{I_i^x}{\bar{d}_i} \rfloor \left(\lfloor \frac{I_i^x}{\bar{d}_i} \rfloor + 1 \right)}{2} = h \left(\left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor + 1 \right) \left(I_i^x - \frac{\bar{d}_i}{2} \left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor \right).
\end{aligned} \tag{26}$$

Where, $\left(\left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor + 1 \right)$ represents the total number of time periods required to deplete the inventory level I_i^x based on the average demand $\bar{d}_i$ for item i, and $\left(I_i^x - \frac{\bar{d}_i}{2} \left\lfloor \frac{I_i^x}{\bar{d}_i} \right\rfloor \right)$ is the average inventory level over the time periods until depletion.

Comparing the Eq. (26) with α_5 enables us to avoid stopping the production of machines whose setup costs are much greater than their inventory costs.

In the following, we refer to this algorithm as *decision rule* (`DR`).

4.2. Approximate Dynamic Programming

Dynamic programming can be used to compute the optimal solution of the $SDLSP_{\text{setup}}$ for small instances, assuming that the demand distribution is known. Using the Markov Decision Process setting described in Section 3, we define the state *value function* $V_t(\mathbf{S}_t)$ as:

$$\begin{aligned}
V_t(\mathbf{S}_t) &= \min_{\mathbf{X}_t \in \mathcal{X}} \mathbb{E}\left[C(\mathbf{X}_t, \mathbf{S}_t) + \gamma V_{t+1}(\mathbf{S}_{t+1}) \right] = \\
&= \min_{\mathbf{X}_t \in \mathcal{X}} \mathbb{E}\left[C(\mathbf{X}_t, \mathbf{S}_t) \right] + \gamma \mathbb{E}\left[V_{t+1}(\mathbf{S}_{t+1}) \right],
\end{aligned} \tag{27}$$

where γ is a discount factor [36].

Unfortunately, Eq. (27) includes a difficult stochastic optimization problem and suffers from the curse of state dimensionality. Therefore, it can only be solved for really small instances. One way to simplify the problem, allowing for statistical learning-based ADP, is to introduce post-decision states [37]. The post-decision state variables refer to the change in the state after the decision has been implemented but before the risk factor realization. In the $SDLSP_{\text{setup}}$ setting, we define the inventory after replenishing but before serving demand ($\mathbf{I_t}^x$) and the setup state at the end of the time bucket ($\mathbf{M_t^x}$). These post-decision states are related to the next pre-decision states by

$$\begin{aligned} \mathbf{I}_{t+1} &= \left[\mathbf{I_t}^x - \mathbf{d}_t\right]^+ \\ \mathbf{M}_{t+1} &= \mathbf{M}_t^x. \end{aligned} \tag{28}$$

Using these variables, we may define a value function around post-decision states, i.e., $V_t^x\left(\mathbf{I}_t^x, \mathbf{M}_t^x\right)$. The standard dynamic programming recursion around pre-decision states is:

$$V_t\left(\mathbf{S}_t\right) = \min_{\mathbf{X}_t \in \mathcal{X}} \left\{\mathbb{E}\left[C\left(\mathbf{X}_t, \mathbf{S}_t\right) \mid \mathbf{X}_t, \mathbf{S}_t\right] + \gamma \mathbb{E}\left[V_{t+1}\left(\mathbf{S}_{t+1}\right) \mid \mathbf{S}_t, \mathbf{X}_t\right]\right\}. \tag{29}$$

The value function around post-decision state is defined as

$$V_t^x\left(\mathbf{S}_t^x\right) = \mathbb{E}\left[V_{t+1}\left(\mathbf{S}_{t+1}\right) \mid \mathbf{S}_t^x\right]. \tag{30}$$

Plugging Eq. (30) in (29), we obtain:

$$V_t\left(\mathbf{S}_t\right) = \min_{\mathbf{X}_t \in \mathcal{X}} \left\{\mathbb{E}[C\left(\mathbf{X}_t, \mathbf{S}_t\right) | \mathbf{X}_t, \mathbf{S}_t] + \gamma V_t^x\left(\mathbf{S}_t^x\right)\right\}. \tag{31}$$

Taking expectations at $t-1$ we get:

$$V_{t-1}^x\left(\mathbf{S}_{t-1}^x\right) = \mathbb{E}\left[V_t\left(\mathbf{S}_t\right) \mid \mathbf{S}_{t-1}^x\right] = \mathbb{E}\left[\min_{\mathbf{X}_t \in \mathcal{X}} \left\{\mathbb{E}[C\left(\mathbf{X}_t, \mathbf{S}_t\right) | \mathbf{X}_t, \mathbf{S}_t] + \gamma V_t^x\left(\mathbf{S}_t^x\right)\right\}\right]. \tag{32}$$

Eq. (32) allows swapping optimization and expectations with respect to Eq. (29). We split the expected value of the immediate costs as:

$$\mathbb{E}[C\left(\mathbf{X}_t, \mathbf{S}_t\right)] = D\left(\mathbf{X}_t, \mathbf{S}_t\right) + G\left(\mathbf{X}_t, \mathbf{S}_t\right), \tag{33}$$

where $D(\mathbf{X}_t, \mathbf{S}_t) = \sum_{m=1}^{M} \sum_{i=1}^{I} f_i \delta_{i,m,t}$ is deterministic component linked to the setup costs, and

$$G(\mathbf{X}_t, \mathbf{S}_t) = \mathbb{E}\left[\sum_{i=1}^{I} h_i \left[I_{i,t} + \sum_{m \in \mathcal{M}(i)} (p_{i,m} x_{i,m,t} - c_{i,m}\delta_{i,m,t}) - d_{i,t}\right]^+ + \sum_{i=1}^{I} l_i \left[d_{i,t} - I_{i,t} + \sum_{m \in \mathcal{M}(i)} (p_{i,m} x_{i,m,t} - c_{i,m}\delta_{i,m,t})\right]^+\right] \tag{34}$$

is the expected value of the stochastic component linked to the holding and lost sales costs that must be learn. The first term of the Eq. (34) calculates the inventory costs as the sum of the inventory at the end of the time, and the total production across all machines, minus the demand. Instead, the second term of Eq. (34) accounts for the lost sales costs considering the demand exceeding the available inventory plus the total production.

It is worth noting that it is not possible to have both terms in Eq. (34) positive since there is either excess inventory (leading to holding costs) or unmet demand (leading to lost sales costs).

Using post-decision state variables, we write $G(\mathbf{X}_t, \mathbf{S}_t)$ as $G^x(\mathbf{I}_t^x)$. It is easy to see that $G^x(\mathbf{I}_t^x)$ is additively separable with respect to the items:

$$G(\mathbf{I}_t^x) = \sum_{i=1}^{I} G(I_i^x), \text{ where } G_i(I_i^x) = \mathbb{E}\left[h_i[I_i^x - d]^+ + l_i[d - I_i^x]^+\right]. \tag{35}$$

Since each $G_i(I_i^x)$ is the expected value of piecewise linear convex functions, it is still convex (see Section 3.2.1 of [38]). Moreover, also $G(\mathbf{I}_t^x)$ is convex being the sum of convex functions. We approximate each $G_i(I_i^x)$ by using a piecewise linear approximation based on regression tree, and we call it $\hat{G}(I_i^x)$.

Since V_t^x still suffers from the curse of dimensionality, we approximate it with a sum of two components: one related to the inventory and one related to the machine configuration. In formula,

$$\hat{V}^x(\mathbf{I}_t^x, \mathbf{M}_t^x) = \sum_{i=1}^{I} \hat{V}_i^{(I)}(I_{it}^x) + \sum_{i=1}^{I} \hat{V}_i^{(M)}(n_{it}^x), \tag{36}$$

where:

- $\hat{V}_t^x(\mathbf{I}_t^x, \mathbf{M}_t^x)$ is the post-decision value function approximation,
- n_{it}^x is the number of machines that are producing item i at time t computed after that the decision is made,
- $\hat{V}_i^{(I)}(\cdot)$ is the function accounting for the costs of the inventory.
- $\hat{V}_i^{(M)}(\cdot)$ is the function accounting for the machine states.

Since each inventory may have $I_{\max}$ maximum value and the maximum number of machines producing an item is M, we consider all the $\hat{V}_i^{(I)}(\cdot)$ and $\hat{V}_i^{(M)}(\cdot)$ in a tabular representation.

It is worth noting that the post-decision value function approximation in Eq. (36) does not consider the time index. With this choice, we are approximating the finite time horizon problem with an infinite time horizon one. This approximation is usually done [33].

The algorithm used to learn $\hat{G}(\cdot)$, $\hat{V}_i^{(I)}(\cdot)$, and $\hat{V}_i^{(M)}(\cdot)$ is described in Pseudocode 2.

Problem (37) is the core of the ADP algorithm as it provides the data for the update of the estimation of the value function as well as deciding the optimal action to take. There, the first term is the deterministic component of the immediate cost, the second one is the expected cost of inventory and lost sales while the third one estimates the future value based on the post-decision states of inventory and machine setup.

Due to its importance, we solve it in an exact way. Nevertheless, since model (37) is non-linear due to $\widehat{V}^x(\cdot)$, we need a procedure that searches exhaustively all the possible solutions. This is equivalent to explore $\prod_{m=1}^{M} |I_m|$ possible solutions, which is clearly out of the question. Therefore, we develop a wise exploration, adapting the *branch and bound* algorithm.

The branch and bound tree has a starting root node, and several levels (each level is associated with a machine). Each node of the level represents a potential state of the corresponding machine (i.e., a possible item to produce or the idle state). A path from the root node to a leaf is a possible solution to model (37),

2 Pseudo-code: ADP algorithm.

Initialize $\hat{V}_i^{(I)}(I), \hat{V}_i^{(M)}(I), \hat{G}_i(I)$ to zero, $\forall I = 0, \ldots, I_{\max}, \forall i$.

Set the total number of iterations N and $k \leftarrow 1$.

while $k \leq N$ **do**

  Set the initial state $\mathbf{S}_0$.

  Generate a sample path for the demand $[d]_{t=0,1,\ldots,T}$.

  for $t \in [0, 1, 2, ..., T-1]$ **do**

$$\widetilde{V}_t \leftarrow \min_x D(x, \mathbf{S}_t) + \sum_{i=1}^{I} \left[\hat{G}_i(I_t^x)\right] + \gamma \hat{V}^x(\mathbf{I}_t^x, \mathbf{M}_t^x) \qquad (37)$$

    Use $\widetilde{V}_t$ to update $\widehat{V}^x$

    Let $\mathbf{X}_t^*$ be the optimal solution of Eq. (37).

    Generate the next post-decision state $\mathbf{S}_t^x$, based on the current state $\mathbf{s}_t$ and the optimal decision $\mathbf{X}_t^*$.

    Generate the next pre-decision state $\mathbf{S}_{t+1}$, based on $\mathbf{S}_t^x$ and d_{t+1}.

    Use the observation of $G(\mathbf{X}_t, \mathbf{S}_t)$ to update $\hat{G}(I^x)$

  Increment the iteration counter $k \leftarrow k + 1$.

i.e. a vector assigning to each machine a state. We call the path from the root node to any intermediate node *partial solution*. We show a part of a general tree in Figure 2. To reduce the search space, we use two pruning strategies, namely *feasibility pruning* and *optimality pruning*.

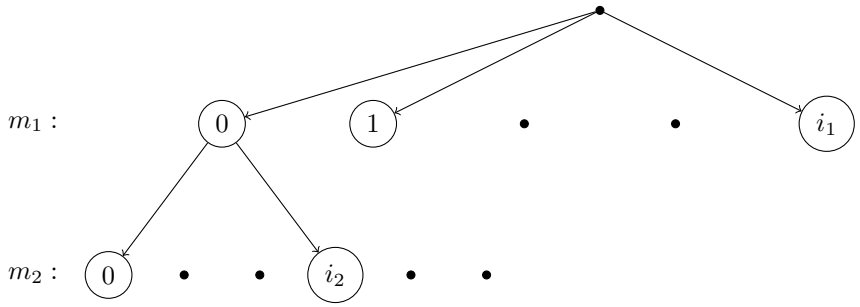


Figure 2: Part of the general branch and bound tree.

We apply *feasibility pruning* when the quantity of items produced by a partial solution leads to an inventory greater than the maximum allowed. For example, if the maximum inventory level for item i is equal to 10, the initial inventory is equal to 5 and we are considering a partial solution in which the production of item i is 6 there is no point in continuing the exploration of the successor of that node since they will violate the maximum inventory constraint.

Instead, we apply *optimality pruning* if all the successors of one nodes lead to a sub-optimal solution. We can detect this condition by looking to the sum of the set up cost and the expected inventory cost since it is a lower bound of the cost of the final solution. If this value is greater than the value of an incumbent solution, there is no point in continue the exploration. To summarize, value function approximation offers several advantages over the traditional vanilla value iteration method commonly used in dynamic programming. While the vanilla method provides an exact solution for smaller instances, it is heavily constrained by the curse of dimensionality, especially when dealing with the complex stochastic nature of the $SDLSP_{\text{setup}}$ problem. Our ADP approach, leveraging post-decision states, addresses these limitations effectively. In fact, it allows for a more effective description of the system's dynamics, providing

a more accurate estimation of the value function by capturing the immediate effects of decisions. This refinement is particularly useful in managing the uncertainty inherent in the $SDLSP_{\text{setup}}$.

Moreover, the use of branch and bound algorithms significantly enhances the efficiency of the solution process enabling us to optimally solve the decision problem (37).

Furthermore, while other studies, such as [39], have applied similar branch and bound heuristics in conjunction with approximated dynamic programming, the proposed heuristic is specifically applied to the lot-sizing problem. In the following, we refer to this technique as `ADP`.

4.3. Actor-critic

Actor-critic architectures are composed of two main components: the *actor* and the *critic*. The *actor* is the policy which is approximated by an ANN and is denoted $\mu_\theta(\mathbf{S})$, where θ is the vector of parameters of the ANN. The *critic* is the value function approximated by an ANN. It can either estimate the state value function (represented as $\hat{V}_\omega(\mathbf{S})$, where ω is the vectors of parameters of the ANN), or the state-action value function (represented as $\hat{Q}_{\omega'}(\mathbf{X}, \mathbf{S})$). Notice that the state-action value function can be expressed in terms of the state-value function: $\hat{Q}_{\omega'}(\mathbf{X}_t, \mathbf{S}_t) = \mathbb{E}\left[C(\mathbf{X}_t, \mathbf{S}_t) + \gamma V_\omega(\mathbf{S}_{t+1})\right]$, where $\mathbf{X}_t = \mu_\theta(\mathbf{S}_t)$, therefore we use the parameters ω' also for the state-action value function [40]. Actor and critic work in tandem to improve both the policy and value estimation. It is important to mention that employing ANN for both the actor and the critic may give a good capacity to handle large action and state spaces (see, e.g., the neural networks used to deal with images [41]).

The update of the actor network is done by using batch stochastic gradient:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}[(\log \mu_\theta(\mathbf{S}))\hat{Q}_\omega(\mathbf{X}, \mathbf{S})]. \tag{38}$$

Note that, we do not employ the t subscript notation as we are working with batches of $\mathbf{X}$ and $\mathbf{S}$.

The update of the critic is done using the stochastic gradient descent applied to the following loss function:

$$L(\mathbf{X}, \mathbf{S}) = \left(\hat{Q}_\omega(\mathbf{X}_t, \mathbf{S}_t) - C(\mathbf{X}_t, \mathbf{S}_t) - \gamma \min_{\mathbf{X}_{t+1}} \hat{Q}_\omega(\mathbf{X}_{t+1}, \mathbf{S}_{t+1}) \right)^2. \tag{39}$$

Notice that in Eq. (39), a temporal difference is used to calculate the loss, $L(\mathbf{X}, \mathbf{S})$ [42, 43]. This loss function is designed to minimize the difference between the actual estimation, $\hat{Q}_\omega(\mathbf{X}_t, \mathbf{S}_t)$, and the temporal difference target estimation, $C(\mathbf{X}_t, \mathbf{S}_t) - \gamma \min_{\mathbf{X}_{t+1}} \hat{Q}_\omega(\mathbf{X}_{t+1}, \mathbf{S}_{t+1})$, associated with the actions taken by the agent in a given state. This enables actor-critic methods to estimate the state-action value without knowing the transition functions. The lack of dependence on the transition function is one of the key characteristics of *model-free* methods.

While the discrete state space of the $SDLSP_{\text{setup}}$ can be easily addressed, the discrete action space must be considered with care since actor-critic agents handle large continuous action spaces describing the probability of taking action in a given state. Therefore, we transform the probability vector output by the actor into action through a function called *embedding*. In summary, the output of the artificial neural network that approximates the policy is a probability matrix of dimensions $(I + 1) \times M$ which is then embedded into the actual decision (a vector of size M that specifies for each machine the action to pick).

Note that an underlying assumption in this approach is that the choices are independent. For instance, if we have two identical machines, the probability of allocating Item A to Machine 2 could be influenced by allocating the same item to Machine 1. However, this potential issue is mitigated by the ANN which accounts for a machine's current setup status. This simplification further reduces the likelihood of such dependencies.

Throughout the training phase, the conversion from continuous to discrete space is accomplished by sampling based on the probability vector. This method enables the policy to explore the action space. Instead, during the testing phase, the action selected is the one corresponding to the maximum probability. The model effectively incorporates the constraints from the mathematical environ-

ment not by fixing them but by allowing the network to learn and understand them with the environment interaction.

In the following, we will use two actor-critic techniques: advantage actor-critic (`A2C`), and proximal policy optimization (`PPO`) [40].

4.4. Lot-sizing cooperative multi-agent adjustment

In this section, we present a multi-agent reinforcement learning method that employs cooperative concepts to address the $SDLSP_{\text{setup}}$. We call the proposed method *Lot-sizing cooperative multi-agent adjustment* (`LSCMA`). It used a *baseline agent* (a pre-trained RL algorithm or another technique) that provides a first action (called *baseline recommendations*, $\mathbf{X}^b$) and M other RL agents, called *sub-agents*, one for each machine, which try to improve the baseline recommendation. Different from other implementations employing multi-agent RL [44], the reward function is not shared across the different sub-agents but we have a different reward function for each sub-agent which is defined as:

$$C_m(\mathbf{X}_t, \mathbf{S}_t,) = \sum_{i \in \mathcal{I}_0(m)} \left[f_{i,m}\delta_{i,m,t} + (h_i I_{i,t} + l_i z_{i,t}) \right]. \tag{40}$$

In contrast to [44, 45], where policies are developed entirely from scratch through environmental interactions, we use a trained baseline agent to generate the initial policy. Therefore, the state that each sub-agent considers is $(\mathbf{S}_t, \mathbf{X}^b)$, where $\mathbf{S}_t$ is the state of the system, and $\mathbf{X}^b$ is the recommendation.

Each sub-agent decides if it should use the recommendations of the baseline agent or the previous recommendation. In formula,

$$\mathbf{X}_{m,t} = \begin{cases} \mathbf{X}^b_{m,t-1} \text{ if } x^-_{m,t} = 1 \\ \mathbf{X}^b_{m,t} \text{ if } x^-_{m,t} = 0 \end{cases}, \tag{41}$$

where $x^-_{m,t}$ is the action of the sub-agent associated with machine m at time t. Therefore, if the action is $x^-_m = 1$, the sub-agent uses the action of the previous setup, and if the action is 0, the agent uses the new baseline setup. Thus, each sub-agent has a reduced set of two possible actions.

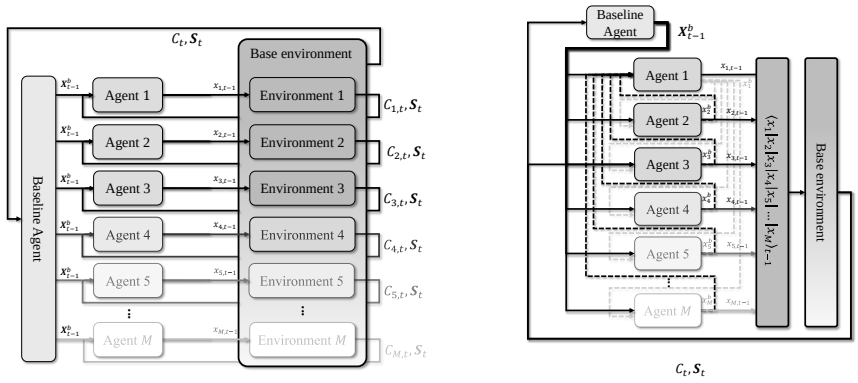
Figure 3 depicts the architecture of the multi-agent system during both the training and testing phases for the `LSCMA`. The training phase for `LSCMA` is detailed in Pseudo-code 3 and graphically represented in Figure 3a. Since each agent works independently, we use a set of adapted environments for each agent, which runs with the same parameter of the standard environment but returns the cost specifically for the sub-agent (considering the set-up cost of the machine that the sub-agents can act on). The baseline agent's decision $\mathbf{X}_{b,t-1}$ and the current state $\mathbf{S}_t$ are given to each sub-agent. Therefore, each sub-agent is aware of all the baseline actions x^b and it decides to use the new recommendation x_t^b or the previous recommendation x_{t-1}^b as shown in Eq (41).

The test method presents important differences from the training method. In fact, during the test each sub-agent uses the baseline agent recommended action, x_t^b, and the environment state, $\mathbf{S}_t$, observed to return a new adjusted decision $x_{m,t}$. This adjusted decision is then informed to all the other sub-agents (represented by the transparent dashed line in Figure 3b) as a new baseline action x_m^b. Then, each sub-agent returns its final decision that is concatenated together with the other baseline decisions to form the final action, i.e., $X_t = \langle x_1 \,|x_2|\, x_3 \,|x_4|\, x_5| \ldots |x_M\rangle_{t-1}$. A graphical representation is depicted in Figure 3b. Notice that we perform this testing procedure to guarantee that the actions of each sub-agent are working cooperatively.

5. Computational Experiments

To compare the proposed techniques, we consider different instances varying the number of machines, items, time steps, max inventory level, and demand distribution (we discuss the instance generation procedure in Subsection 5.1). In detail, we conduct three sets of computational experiments.

In the first set of experiments, reported in Subsection 5.2, we address a problem with one machine and two items. This setting (detailed in Appendix A, Table A.4) allows us to compute the optimal solution with value iteration and to compare `DR`, `A2C`, `PPO`, `MS`, `ADP` against it. Here, we measure the performance



(a) `LSCMA` training

(b) `LSCMA` test

Figure 3: Training and testing architectures for the `LSCMA`.

3 Pseudo-code: Lot-sizing multi-agent adjustment

Generate and train the baseline agent.
Initialize the parameters of each sub-agents.
Select the total number of iterations N
$k \leftarrow 1$.
while $k \leq N$ **do**
    Set an initial random state $\mathbf{S}_0^j$.
    Generate a sample path for the demand $[d]_{t=0,1,\ldots,T}$.
    for $t \in (0, 1, 2, ..., T-1)$ **do**
        Compute $\mathbf{X}_{t+1}^b$ from the baseline agent
        for $m \in \mathcal{M}$ **do**
            Compute $x_{m,t+1}$ from each sub-agent
        $\mathbf{X}_t \leftarrow (x_{1,t}, x_{2,t}, \ldots, x_{m,t})$
        Generate the next state $s_{m,t+1}$, based on $s_{m,t}$, d_{t+1}, and $\mathbf{X}_t$.
    Updates the sub-agents using the new information;
    Increment the iteration counter $k \leftarrow k + 1$.

of each method against the optimal solution computed by value iteration. We do not consider LSCMA since, with one machine, we would have just one sub-agent, making the multi-agent approach useless.

In the second set of experiments, reported in Subsection 5.3, we tackle instances with up to 15 items and 5 machines. This setting (detailed in Appendix A, Table A.4) prevents the usage of exact techniques such as value iteration. Therefore, we compare the performance of DR, A2C, PPO, MS, ADP, and LSCMA using the perfect information agent as a benchmark.

Finally, in the third set of experiments, reported in Subsection 5.4, we examine large instances with up to 25 items and 10 machines. In this setting (detailed in Appendix A, Table A.5), the perfect information agent is no longer a viable baseline comparison due to the size of the mathematical model. Therefore, we compare the absolute costs of DR, A2C, PPO, MS, ADP, and LSCMA.

All the computational experiments were run on an AMD Ryzen 5 5600X 6-Core Processor 3.70 GHz, an RTX 3060 (12GB), and 32GB of RAM. The code (available upon request) has been developed in Python 3.6; the libraries used for the DLR algorithms are Pytorch [46] and Stable baselines3 [47]. Gurobi v9.5.0 solves all the models via its Python3 APIs.

The following settings are considered in the experiments (unless otherwise stated):

- **Multi-stage configuration:** For the MS model, we implement a branching factor of $[4, 4, 2, 2]$. Recognizing the limitations of rough Monte Carlo scenario generation, as highlighted in [48], we intentionally reduce scenarios to enhance result accuracy. Due to the potentially time-consuming nature of solving the model for some instance configurations, we set a time limit of 5 minutes per solution for each decision. This time limit is crucial, especially considering that our experiments encompass 100 simulations, each containing 10 to 100 steps. A time frame exceeding 5 minutes per step would render the MS approach impractical for our experimental scope. Additionally, as the complexity of the problem increases with more

machines and items, the time required for computation scales accordingly, eventually surpassing the feasibility threshold for available computing resources.

- **Decision Rule parameters:** Parameters $\alpha_1, \alpha_3, \alpha_4$, and α_5 for the `DR` model are optimized using *particle swarm optimization*.
- **Reinforcement learning configuration:** The `LSCMA` model employs `DR` as its baseline policy, supplemented by `PPO` for each sub-agent. As a comparison, we also use single-agent RL methods, both `PPO` and `A2C` methods configurated to train for 50,000 epochs, with the number of iterated steps tailored to the specific environment configuration. A detailed list of hyperparameters for the single agent compared methods for both `PPO` and `A2C` is available in the supplementary material (see Appendix A, Table A.6).

Since the demand is stochastic, the performances of the methods are averaged over 100 different episodes. Despite the considerable error range implied by the standard deviation, the statistical robustness of our experiment—stemming from the substantial number of tests—ensures reliable average performance metrics across various initial conditions.

5.1. Instance generation

Given a number of items I, of machines M, a number of time steps T, and a maximum inventory size $I_{\max}$, we generate the initial inventory for each item a random number between $[0, I_{\max}]$. Then, following what is done in [1], we generate h_i, l_i, and $f_{i,m}$ from a uniform random variable in $[0, 1]$, $[5, 10]$, and $[1, 5]$, respectively.

The production matrix (i.e., $p_{i,m}$) is generated by randomly selecting $2|\mathcal{I}|/|\mathcal{M}|$ items for each machine; for these items, the production is drawn from a uniform distribution between 10 and 20, while for the remaining ones, $p_{i,m} = 0$. This setting ensures that several machines produce the same item, thus avoiding a problem that can be decomposed. Then, we check that each row of $p_{i,m}$ has at

least one non-zero element (i.e., there is at least a machine that can produce it). If not, we set two random components to a value drawn from a uniform distribution between 10 and 20.

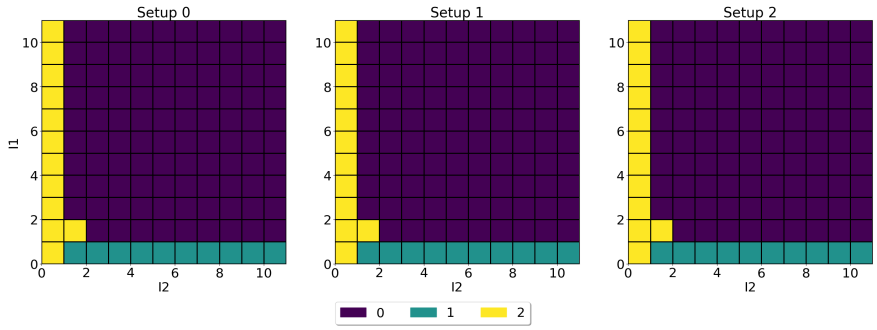
In this study, demand is modeled using different binomial distributions whose parameters adapt to the instance type (e.g. the instances with more machines will be characterized by higher demand). Since the problem is characterized by several parameters, we present in this paper only the most interesting results that we have obtained. Nevertheless, the open-source code is available at `https://github.com/leokan92/discrete_lot_sizing_rl_agents`, to properly guarantee reproducibility and enable interested readers to carry out further experiments.

5.2. Small size instance

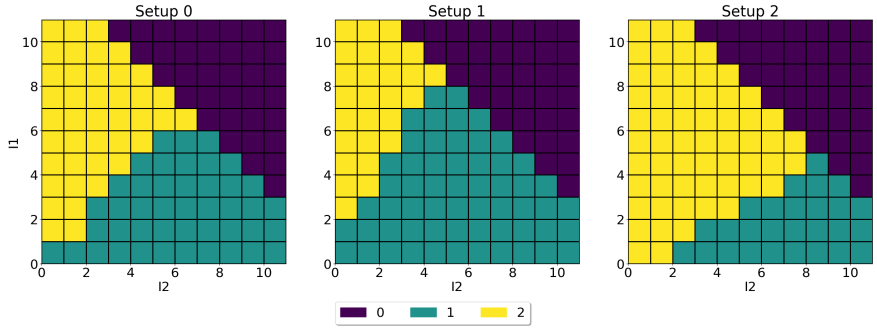
This section presents the computational results for the small instance ($I = 2$, $M = 1$, and $T = 10$). We consider that production, setup costs, setup losses, and inventory costs are the same for the two items, while the lost sale costs of item 2 are twice the lost sales of item 1. These characteristics enable us to compute the exact solution using value iteration and to visualize the policy function.

In Figure 4, we show the optimal policy (computed using `VI`) for different demand distributions. In particular, we use a binomial distribution with $n = 3$ and $p = \frac{1}{3}$ for Figure 4a while we use $n = 5$, $p = 0.4$ for Figure 4b.

We represent the policy employing three tables for each demand scenario, one for each present machine setup. Each table contains $I_{\max} \times I_{\max}$ cells, one cell for each possible inventory level, and each cell is shaded with a color corresponding to the optimal action. The green color (in the legend, the color 1) corresponds to the production of item 1 (in the legend, the color 2), the yellow color is the production of item 2 (in the legend, the color 0), and the blue color is the idle state. As the reader can notice, the optimal policy has a similar pattern for each of the three tables: if the inventory of one item is small, its production starts, and if both the inventories are small, item 2 is preferred due to its greater



(a) `VI` policy visualization



(b) `VI` policy visualization - higher demands

Figure 4: Comparison of `VI` policy visualizations under different demands - lower and higher demand

lost sales cost. Moreover, in Figure 4b (higher demands), it is possible to notice that setup deeply affects the optimal policy. When the machine produces item 1 (second table), the number of cells in which the production is maintained is greater than in the other two tables. The same holds, almost symmetrical when the machine produces item 2 (see the third table).

These representations help to obtain insights concerning the pattern of the optimal policy. In the following, we compare the methods using only the lower demand scenario with a binomial distribution parameterized with $n = 3$ and $p = \frac{1}{3}$. We report the results of the experiments in Table 1. Table 1 defines the experiment configuration in the column "Experiment". This column indicates the number of items by the number followed by the letter "I", the number of machines with the number followed by the letter "M," the number of time

Table 1: Average total costs, holding, lost sales and setup costs percentage with respect to value iteration for the one machine and two items setting.

Experiment	**Algorithm**	**Total Costs %**	**Holding Costs %**	**Lost Sales Costs %**	**Setup Costs %**
I2 M1 T20 $I_{\max}10$	DR	6±71	-12±25	100±479	0±94
	ADP	2±12	-2±5	3±2	0±4
	A2C	114±172	-31±31	992±1181	-54±120
	PPO	1±46	-3±21	18±301	8±67
	MS	**0±8**	0±4	0±49	0±17

steps in an episode next to "T," and the maximum number of each item in the inventory after the notation "$I_{\max}$".

As the reader can notice, MS, PPO, and ADP have performance close to VI, but PPO has a far much bigger standard deviation than the other three methods. DR performs quite well, achieving an average gap of 6%, while A2C achieves the worst performance being distant from the other methods. Its bad performance is due to its underproduction, which leads to huge lost sales costs. The high standard deviation observed in the results is due to the stochastic demand and to the different initial conditions. To cope with this inherent limitation we run 100 episodes of the test.

5.3. Medium size instances

This section presents the computational results for the medium instances ($I = 4M = 2, T = 10$, $I = 10M = 5, T = 10$, and $I = 15M = 5, T = 10$). For those instances, VI cannot be applied due to dimensionality. Therefore, we use the PI agent as benchmark. The distribution of item demand is given by the binomial distribution with $p = 0.4$ and $n = 4$ for the three instances. The average results are shown in Table 2.

In the $I4$ $M2$ instances, PPO performs best, followed by LSCMA and by ADP. MS performs poorly due to the time limit that prevents reaching the optimal solution. It is worth noting that while its computation is not finished after

Table 2: Average total costs, holding, lost sales, and setup costs percentage with respect to the perfect information agent.

Experiment	Algorithm	Total Costs %	Holding Costs %	Lost Sales Costs %	Setup Costs %
I4 M2 T10 $I_{\max}10$	DR	159±159	-59±39	465±493	104±213
	ADP	64±102	-38±41	231±274	16±193
	A2C	208±131	9±41	730±396	-79±162
	PPO	**47±87**	3±44	115±187	29±197
	MS	131±140	-50±39	310±369	161±221
	LSCMA	59±95	-7±43	207±253	-13±182
I10 M5 T10 $I_{\max}10$	DR	94±100	-29±29	425±453	67±126
	ADP	102±97	-23±33	704±458	-56±107
	A2C	138±89	44±35	732±440	-51±140
	PPO	145±108	3±43	840±509	-42±119
	MS	89±84	-29±31	226±253	155±151
	LSCMA	**83±90**	4±36	421±394	3±115
I15 M5 T10 $I_{\max}10$	DR	86±73	-41±26	155±130	67±99
	ADP	**37±48**	-13±27	85±76	-19±104
	A2C	93±57	25±38	193±103	-62±91
	PPO	67±46	23±31	95±70	51±103
	MS	61±53	-33±24	82±77	115±111
	LSCMA	65±60	3±34	132±102	-19±95

the time limit of 5 minutes, all the other methods require a computational time smaller than a minute. Moreover, while DR has poor results, LSCMA is the second best method, proving that the multi-agent strategy can deeply improve the starting policy of the baseline agent.

The best policy in the setting with 10 items and 5 machines is LSCMA followed by DR and MS. Considering the costs, it is possible to notice that ADP strongly reduces setup costs at the expense of increased lost sales costs. Instead, LSCMA maintains setup costs lower than DR while achieving a relatively low lost sales cost, which is crucial for its overall superior performance. Moreover, the holding

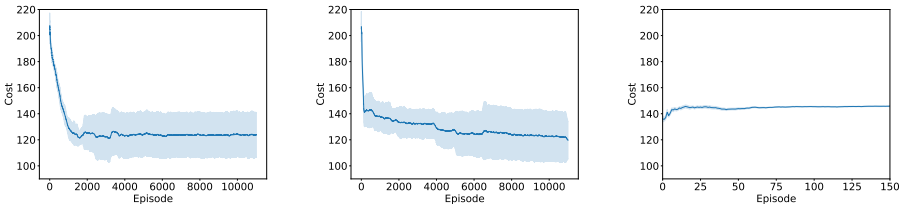
costs for all three methods are quite similar, with the LSCMA being higher.

Finally, in the setting with 15 items and 5 machines, the lost sales cost takes on greater significance, rendering the ADP policy the best choice, surpassing the LSCMA policy.

More in detail, ADP policy incurs a slightly lower setup cost yet maintains the lowest lost sales cost. Instead, DR while being the best model in terms of holding cost, it generates a lot of lost sales and setup costs, which means that it stops production too often. Finally, LSCMA deals with DR deficiency and improves over it, having a better result than DR in all three costs.

When comparing the LSCMA multi-agent approach against the PPO and A2C methods, we observe that the training times of LSCMA are much faster than the ones of both PPO and A2C. To better investigate convergence, we report the cost in the different episodes of the training set in Figure 5. As the reader can see, the cost of LSCMA slightly increases in the first episodes and quickly stabilizes. This increment is due to the exploration phase of the method that terminates after 50 episodes. Then the cost becomes stable. This really fast convergence is due to the small set of actions that each sub-agent considers. Moreover, due to a good starting solution also standard deviation of the cost is reduced.

It is worth noting that, despite the costs achieved by LSCMA in the training set are greater than the ones achieved by PPO and A2C, in the test the result is the opposite, due to the change in the LSCMA structure.



(a) PPO (b) A2C (c) LSCMA

Figure 5: Moving average of 100 of the cost curve with standard deviation of the distribution taking 5 different seeds for the instance I10 M5 T10 $I_{\max}10$.

In conclusion, we can claim that while for small instances, PPO provides

good solutions, both LSCMA and ADP have better results in medium-sized ones. Moreover, it is worth noting that ADP and LSCMA deliver better outcomes, particularly in reducing setup and inventory costs and they are also able to reduce or maintain the lost sales while reducing the other costs.

5.4. Large size instances

This section considers instances with $I = 15, M = 5$, $I = 25$, $M = 10$, and a time horizon of 100 time steps. For the large size instances, MS becomes too time-consuming even with the time limits: in the $I15M5$ instances (the smaller ones), the computation rarely stops before the 5 minutes, thus requiring a huge amount of time for the 100-time steps repeated 100 times. Therefore, we just compare DR, ADP, A2C, PPO, and LSCMA. Due to the branch and bound procedure implemented, ADP becomes too time-consuming for the $I25M10$ setting, leaving DR, A2C, PPO, and LSCMA as the only candidate method for these instances.

For the $I15M5T100I_{\max}10$, the demand is distributed according to a binomial distribution with the same parameters of previous experiments $p = 0.4$ and $n = 4$ while for the $I15M5T100I_{\max}100$ and $I25M10$ ones, the parameter n assumes a value of 20, thereby increasing the demand. We consider greater demand values to maintain the proportion of production and demand capacity while increasing both T and $I_{\max}$. We show the average cost in Table 3. ADP performs better than the other methods achieving the best results for all the instances with 15 items.

Comparing the results with $I = 15$, with $I_{\max} = 10$, and $I_{\max} = 100$, it is possible to notice how the methods behave concerning a change in the dimension of the state space. As the reader can notice, the results are almost the same, meaning all the methods can deal effectively with the larger state space. Notice that the increment in dimensionality affect ADP computational time since the number of feasibility cut decreases.

Finally, in the setting with $I = 25$, the best method is LSCMA which improves the performance of DR and reduces its variance. Interestingly, DR surpasses both A2C and PPO, meaning that for high dimensional space, the learning methods

Table 3: Average total costs, holding, lost sales and setup costs. Testing in settings on which perfect information agent it is not able to provide optimal solution

Experiment	**Algorithm**	**Total Costs**	**Holding Costs**	**Lost Sales Costs**	**Setup Costs**
I15 M5 T100 $I_{\max}10$	DR	2062±443	44±135	1966±647	52±88
	ADP	**1393±163**	112±110	888±217	393±59
	A2C	2000±250	327±72	1555±319	118±76
	PPO	2066±269	322±77	1737±367	7±70
	LSCMA	1988±319	243±97	1715±436	30±28
I15 M5 T100 $I_{\max}100$	DR	1982±384	58±122	1689±515	235±34
	ADP	**1472±169**	109±101	1137±253	226±26
	A2C	2129±270	320±70	1801±364	8±81
	PPO	1918±230	103±104	1705±337	110±79
	LSCMA	1981±343	229±95	1655±459	97±36
I25 M10 T100 $I_{\max}100$	DR	3290±922	129±253	2962±1397	199±272
	A2C	3609±470	436±130	3158±642	15±152
	PPO	3612±468	435±127	3158±641	19±191
	LSCMA	**3249±590**	451±166	2682±825	117±101

of these two algorithms start to fail. More in detail, LSCMA mostly reduces lost sales costs using the production capacity.

6. Conclusions

This paper addresses the stochastic discrete lot sizing problem on parallel machines. The problem arises when the time step considered does not allow for the production of simultaneous different items on the same machine. This characteristic leads to a difficult integer programming problem that requires heuristics to be solved. We propose two heuristics: one based on approximate dynamic programming and leveraging a branch and bound technique to solve the nonlinear optimization problem for selecting the action (ADP), and one based on multi-agent reinforcement learning applied to an initial action generated by a simple decision rule (LSCMA).

Through computational experiments, we provide shreds of evidence of the effectiveness of these techniques on a set of instances, and we compare their performance against state-of-the-art deep reinforcement learning techniques (namely, proximal policy optimization and advantage actor-critic). The results show that the proposed techniques behave on average better than the benchmark techniques. Future studies will address more complex versions of the stochastic discrete lot sizing problem, e.g., considering non-parallel machines and/or uncertainty in both demand and production rates. Moreover, an interesting topic to consider is heteroscedastic demand. In fact, due to the orders the demand in closer timeframes is characterized by less noise compared to that in more distant timeframes.

Lastly, another promising field of research is the integration of model-free methods together with more standard operation research techniques. In fact, one of the results of this paper shows that up to medium-size instances, these techniques achieve the best results. These mixed strategies promise to advance the field further, offering more versatile and powerful solutions to complex and high-dimensional lot-sizing problems and other problems.

CRediT author statement

Leonardo Kanashiro Felizardo: Conceptualization, Methodology, Formal analysis, Data Curation, Investigation, Writing - Original Draft, Writing - Review & Editing, Project administration **Edoardo Fadda**: Conceptualization, Methodology, Formal analysis, Software, Investigation Writing - Original Draft, Writing - Review & Editing **Emilio Del-Moral-Hernandez**: Validation, Writing - Review & Editing, Supervision. **Paolo Brandimarte**: Conceptualization, Validation, Writing - Review & Editing, Supervision.

Acknowledgment

This research was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES - Coordination for the Improvement of

Higher Education Personnel, Finance Code 001, grant 88882.333380/2019-01), Brazil.

References

[1] P. Beraldi, G. Ghiani, A. Grieco, E. Guerriero, Fix and relax heuristic for a stochastic lot-sizing problem, Computational Optimization and Applications 33 (2-3) (2005) 303–318. doi:10.1007/s10589-005-3055-2.
URL https://doi.org/10.1007/s10589-005-3055-2

[2] L. A. W. Yves Pochete, Production Planning by Mixed Integer Programming, Springer New York, 2006. doi:10.1007/0-387-33477-7.
URL https://doi.org/10.1007/0-387-33477-7

[3] N. Brahimi, S. Dauzere-Peres, N. M. Najid, A. Nordli, Single item lot sizing problems, European Journal of Operational Research 168 (1) (2006) 1–16. doi:https://doi.org/10.1016/j.ejor.2004.01.054.
URL https://www.sciencedirect.com/science/article/pii/S0377221704003923

[4] D. Taş, M. Gendreau, O. Jabali, R. Jans, A capacitated lot sizing problem with stochastic setup times and overtime, European Journal of Operational Research 273 (1) (2019) 146–159. doi:https://doi.org/10.1016/j.ejor.2018.07.032.
URL https://www.sciencedirect.com/science/article/pii/S0377221718306374

[5] N. Sereshti, Y. Adulyasak, R. Jans, The value of aggregate service levels in stochastic lot sizing problems, Omega 102 (2021) 102335. doi:https://doi.org/10.1016/j.omega.2020.102335.
URL https://www.sciencedirect.com/science/article/pii/S0305048320306897

[6] B. Karimi, S. F. Ghomi, J. Wilson, The capacitated lot sizing problem: a review of models and algorithms, Omega 31 (5) (2003) 365–378. doi:

10.1016/s0305-0483(03)00059-8.
URL https://doi.org/10.1016/s0305-0483(03)00059-8

[7] H. Stadtler, M. Meistering, Model formulations for the capacitated lot-sizing problem with service-level constraints, OR Spectrum 41 (4) (2019) 1025–1056.

[8] B. Fleischmann, The discrete lot-sizing and scheduling problem, European Journal of Operational Research 44 (3) (1990) 337–348. doi:10.1016/0377-2217(90)90245-7.
URL https://doi.org/10.1016/0377-2217(90)90245-7

[9] M. Salomon, M. M. Solomon, L. N. V. Wassenhove, Y. Dumas, S. Dauzère-Pérès, Solving the discrete lotsizing and scheduling problem with sequence dependent set-up costs and set-up times using the travelling salesman problem with time windows, European Journal of Operational Research 100 (3) (1997) 494–513. doi:10.1016/s0377-2217(96)00020-3.
URL https://doi.org/10.1016/s0377-2217(96)00020-3

[10] M. Salomon, L. G. Kroon, R. Kuik, L. N. V. Wassenhove, Some extensions of the discrete lotsizing and scheduling problem, Management Science 37 (7) (1991) 801–812. doi:10.1287/mnsc.37.7.801.
URL https://doi.org/10.1287/mnsc.37.7.801

[11] W. Bruggemann, H. Jahnke, Remarks on: "some extensions of the discrete lotsizing and scheduling problem", Management Science 43 (1) (1997) 122–122. doi:10.1287/mnsc.43.1.122.
URL https://doi.org/10.1287/mnsc.43.1.122

[12] R. JANS, Z. DEGRAEVE, An industrial extension of the discrete lot-sizing and scheduling problem, IIE Transactions 36 (1) (2004) 47–58. doi:10.1080/07408170490247296.
URL https://doi.org/10.1080/07408170490247296

[13] S. G. Dastidar, R. Nagi, Scheduling injection molding operations with multiple resource constraints and sequence dependent setup times and costs, Computers & Operations Research 32 (11) (2005) 2987–3005. doi:10.1016/j.cor.2004.04.012.
URL https://doi.org/10.1016/j.cor.2004.04.012

[14] C. Silva, J. M. Magalhaes, Heuristic lot size scheduling on unrelated parallel machines with applications in the textile industry, Computers & Industrial Engineering 50 (1-2) (2006) 76–89. doi:10.1016/j.cie.2006.01.001.
URL https://doi.org/10.1016/j.cie.2006.01.001

[15] B. Almada-Lobo, D. Klabjan, M. A. Carravilla, J. F. Oliveira, Multiple machine continuous setup lotsizing with sequence-dependent setups, Computational Optimization and Applications 47 (3) (2009) 529–552. doi:10.1007/s10589-009-9235-8.
URL https://doi.org/10.1007/s10589-009-9235-8

[16] C. Gicquel, M. Minoux, Y. Dallery, Exact solution approaches for the discrete lot-sizing and scheduling problem with parallel resources, International Journal of Production Research 49 (9) (2011) 2587–2603. doi:10.1080/00207543.2010.532927.
URL https://doi.org/10.1080/00207543.2010.532927

[17] R. F. Fachini, K. F. Esposto, V. C. B. Camargo, Glass container production planning with warm-ups and furnace extraction variation losses, The International Journal of Advanced Manufacturing Technology 90 (1-4) (2016) 527–543. doi:10.1007/s00170-016-9369-7.
URL https://doi.org/10.1007/s00170-016-9369-7

[18] C. F. M. Toledo, M. da Silva Arantes, M. Y. B. Hossomi, B. Almada-Lobo, Mathematical programming-based approaches for multi-facility glass container production planning, Computers & Operations Research 74 (2016) 92–107. doi:10.1016/j.cor.2016.02.019.
URL https://doi.org/10.1016/j.cor.2016.02.019

[19] G. Claassen, J. Gerdessen, E. Hendrix, J. van der Vorst, On production planning and scheduling in food processing industry:modelling non-triangular setups andproduct decay, Computers & Operations Research 76 (2016) 147–154. doi:10.1016/j.cor.2016.06.017.
URL https://doi.org/10.1016/j.cor.2016.06.017

[20] E. A. Toso, R. Morabito, A. R. Clark, Lot sizing and sequencing optimisation at an animal-feed plant, Computers & Industrial Engineering 57 (3) (2009) 813–821. doi:10.1016/j.cie.2009.02.011.
URL https://doi.org/10.1016/j.cie.2009.02.011

[21] S. A. de Araujo, M. N. Arenales, A. R. Clark, Lot sizing and furnace scheduling in small foundries, Computers & Operations Research 35 (3) (2008) 916–932. doi:10.1016/j.cor.2006.05.010.
URL https://doi.org/10.1016/j.cor.2006.05.010

[22] D. Ferreira, R. Morabito, S. Rangel, Relax and fix heuristics to solve one-stage one-machine lot-scheduling models for small-scale soft drink plants, Computers & Operations Research 37 (4) (2010) 684–691. doi:10.1016/j.cor.2009.06.007.
URL https://doi.org/10.1016/j.cor.2009.06.007

[23] V. C. Camargo, F. M. Toledo, B. Almada-Lobo, HOPS – hamming-oriented partition search for production planning in the spinning industry, European Journal of Operational Research 234 (1) (2014) 266–277. doi:10.1016/j.ejor.2013.10.017.
URL https://doi.org/10.1016/j.ejor.2013.10.017

[24] C. Ge, Z. Yuan, Production scheduling for the reconfigurable modular pharmaceutical manufacturing processes, Computers & Chemical Engineering 151 (2021) 107346. doi:https://doi.org/10.1016/j.compchemeng.2021.107346.
URL https://www.sciencedirect.com/science/article/pii/S0098135421001241

[25] R. Ramezanian, M. Saidi-Mehrabad, Hybrid simulated annealing and MIP-based heuristics for stochastic lot-sizing and scheduling problem in capacitated multi-stage production system, Applied Mathematical Modelling 37 (7) (2013) 5134–5147. doi:10.1016/j.apm.2012.10.024.
URL https://doi.org/10.1016/j.apm.2012.10.024

[26] Z. Hu, G. Hu, A two-stage stochastic programming model for lot-sizing and scheduling under uncertainty, International Journal of Production Economics 180 (2016) 198–207. doi:10.1016/j.ijpe.2016.07.027.
URL https://doi.org/10.1016/j.ijpe.2016.07.027

[27] X. Zhu, W. E. Wilhelm, Scheduling and lot sizing with sequence-dependent setup: A literature review, IIE Transactions 38 (11) (2006) 987–1007. doi:10.1080/07408170600559706.
URL https://doi.org/10.1080/07408170600559706

[28] S.-I. Chen, D. Su, A multi-stage stochastic programming model of lot-sizing and scheduling problems with machine eligibilities and sequence-dependent setups, Annals of Operations Research 311 (1) (2019) 35–50. doi:10.1007/s10479-019-03462-1.
URL https://doi.org/10.1007/s10479-019-03462-1

[29] C. D. Paternina-Arboleda, T. K. Das, A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem, Simulation Modelling Practice and Theory 13 (5) (2005) 389–406. doi:https://doi.org/10.1016/j.simpat.2004.12.003.
URL https://www.sciencedirect.com/science/article/pii/S1569190X04001406

[30] J. Wang, X. Li, X. Zhu, Intelligent dynamic control of stochastic economic lot scheduling by agent-based reinforcement learning, International Journal of Production Research 50 (16) (2012) 4381–4395. arXiv:https://doi.org/10.1080/00207543.2011.592158, doi:

10.1080/00207543.2011.592158.
URL https://doi.org/10.1080/00207543.2011.592158

[31] T. J. Park, Y. J. Jang, Discrete lot-sizing problem of single machine based on reinforcement learning approach, in: Proceedings of the International Symposium on Semiconductor Manufacturing Intelligence (ISMI2022), 2022, p. 6.

[32] H. Rummukainen, J. K. Nurminen, Practical reinforcement learning -experiences in lot scheduling application, IFAC-PapersOnLine 52 (13) (2019) 1415–1420, 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019. doi:https://doi.org/10.1016/j.ifacol.2019.11.397.
URL https://www.sciencedirect.com/science/article/pii/S2405896319313783

[33] L. van Hezewijk, N. Dellaert, T. V. Woensel, N. Gademann, Using the proximal policy optimisation algorithm for solving the stochastic capacitated lot sizing problem, International Journal of Production Research 0 (0) (2022) 1–24. arXiv:https://doi.org/10.1080/00207543.2022.2056540, doi:10.1080/00207543.2022.2056540.
URL https://doi.org/10.1080/00207543.2022.2056540

[34] P. Brandimarte, Multi-item capacitated lot-sizing with demand uncertainty, International Journal of Production Research 44 (2006) 2997–3022.

[35] U. S. Karmarkar, Equalization of runout times, Operations Research 29 (4) (1981) 757–762.
URL http://www.jstor.org/stable/170389

[36] P. Brandimarte, From Shortest Paths to Reinforcement Learning: A MATLAB-based Tutorial on Dynamic Programming, Springer, 2021.

[37] W. B. Powell, Approximate Dynamic Programming: Solving the Curses

of Dimensionality, 2nd Edition, Wiley Series in Probability and Statistics, Wiley, Hoboken, NJ, USA, 2011.

[38] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.

[39] S. Tanaka, K. Tierney, C. Parreño-Torres, R. Alvarez-Valdes, R. Ruiz, A branch and bound approach for large pre-marshalling problems, European Journal of Operational Research 278 (1) (2019) 211–225. doi:https://doi.org/10.1016/j.ejor.2019.04.005.
URL https://www.sciencedirect.com/science/article/pii/S037722171930325X

[40] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, 2nd Edition, A Bradford Book, Cambridge, MA, USA, 2018.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533. doi:10.1038/nature14236.
URL https://doi.org/10.1038/nature14236

[42] A. L. Samuel, Some studies in machine learning using the game of checkers, IBM Journal of Research and Development 3 (3) (1959) 210–229. doi:10.1147/rd.33.0210.

[43] R. S. Sutton, Learning to predict by the methods of temporal differences, Machine Learning 3 (1) (1988) 9–44. doi:10.1007/BF00115009.
URL https://doi.org/10.1007/BF00115009

[44] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. M. Bayen, Y. Wu, The surprising effectiveness of MAPPO in cooperative, multi-agent games, CoRR abs/2103.01955. arXiv:2103.01955.
URL https://arxiv.org/abs/2103.01955

[45] M. Wen, J. G. Kuba, R. Lin, W. Zhang, Y. Wen, J. Wang, Y. Yang, Multi-agent reinforcement learning is a sequence modeling problem (2022). arXiv:2205.14953.

[46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.

[47] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann, Stable-baselines3: Reliable reinforcement learning implementations, Journal of Machine Learning Research 22 (268) (2021) 1–8.
URL http://jmlr.org/papers/v22/20-1364.html

[48] H. Heitsch, W. Römisch, Scenario reduction algorithms in stochastic programming, Computational Optimization and Applications 24 (2) (2003) 187–206. doi:10.1023/A:1021805924152.
URL https://doi.org/10.1023/A:1021805924152

Appendix A.

Here we show a summary of the scenario configuration settings. Table A.4 refers to the results from the experiment in subsection 5.3. In Table A.5 we list the environment configuration setting employed in the experiments in presented in Subsections 5.2, 5.3, 5.4

We also provide the hyperparameters employed for the PPO and A2C algorithms in Table A.6.

Table A.4: Scenario configuration settings for **lower** number of steps setting. Since we generate random numbers for some of the environment settings, we only present the interval and the matrix shape.

Parameter	I2 M1 T20	I4 M2 T20	I10 M5 T10	I15 M5 T10
Time horizon	20	20	10	10
Number of items	2	4	10	15
Number of machines	1	2	5	5
Initial setup	0	$1 \times m$ array $\in [0, n]$	$1 \times m$ array $\in [0, n]$	$1 \times m$ array $\in [0, n]$
Machine production	3	$m \times n$ array $\in [0, n]$	$m \times n$ array $\in [0, n]$	$m \times n$ array $\in [0, n]$
Max inventory level	10	10	10	10
Initial inventory	0	$1 \times n$ array $\in [0, 10]$	$1 \times n$ array $\in [0, 10]$	$1 \times n$ array $\in [0, 10]$
Holding costs	0.01	0.1	0.1	0.1
Lost sales costs	1	$1 \times n$ array $\in [1, 3]$	$1 \times n$ array $\in [1, 3]$	$1 \times n$ array $\in [1, 3]$
Demand distribution	Binomial, $d_{i,t,2}$	Binomial $d_{i,t,4}$	Binomial $d_{i,t,4}$	Binomial $d_{i,t,4}$
Setup costs	1	2	2	2
Setup loss	1	1	1	1

Table A.5: Scenario configuration settings for **higher** number of steps setting. Since we generate random numbers for some of the environment settings, we only present the interval and the matrix shape.

Parameter	I15 M5 T100 M10	I15 M5 T100 M100	I25 M10 T100 M100
Time horizon	100	100	100
Number of items	15	15	15
Number of machines	5	5	5
Initial setup	$1 \times m$ array $\in [0, n]$	$1 \times m$ array $\in [0, n]$	$1 \times m$ array $\in [0, n]$
Machine production	$m \times n$ array $\in [0, n]$	$m \times n$ array $\in [0, n]$	$m \times n$ array $\in [0, n]$
Max inventory level	10	10	10
Initial inventory	$1 \times n$ array $\in [0, 10]$	$1 \times n$ array $\in [0, 100]$	$1 \times n$ array $\in [0, 100]$
Holding costs	0.1	0.1	0.1
Lost sales costs	$1 \times n$ array $\in [1, 3]$	$1 \times n$ array $\in [1, 3]$	$1 \times n$ array $\in [1, 3]$
Demand distribution	Binomial $d_{i,t,4}$	Binomial $d_{i,t,20}$	Binomial $d_{i,t,20}$
Setup loss	1	1	1
Setup costs	2	2	2

Table A.6: Hyperparameters for `PPO` and `A2C` models

Hyperparameter	`PPO`	`A2C`
Batch size	256	-
Number of steps	256	100
Gamma (γ)	0.96	0.95
GAE lambda	0.9	-
Update delay (epochs)	20	-
Entropy coefficient	0.0	-
Max gradient norm	0.5	-
Value function coef.	0.5	0.7
Learning rate α	5e-3	0.002
Use SDE	False	-
Clip range ϵ	0.4	-
Policy net architecture	[300, 300]	[300, 300]