

Reinforcement Learning for Scheduling Processes Under Uncertainty in Chemical Engineering Facilities

by

Daniel Rangel-Martinez

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Chemical Engineering

Waterloo, Ontario, Canada, 2025

© Daniel Rangel-Martinez 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Biao Huang
 Professor
 Chemical and Materials Engineering
 University of Alberta

Supervisor: Luis Alberto Ricardez-Sandoval
 Professor
 Department of Chemical Engineering
 University of Waterloo

Internal Members: Hector Budman
 Professor
 Department of Chemical Engineering
 University of Waterloo

 Nasser Mohieddin Abukhdeir
 Associate Professor
 Department of Chemical Engineering
 University of Waterloo

Internal-External Member: William Melek
 Professor
 Department of Mechanical and Mechatronics Engineering
 University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Optimal scheduling of chemical systems has gained interest as it provides economic advantages. The development of methodologies for approaching this problem have expanded considerably bringing multiple options to the process optimization field. Real world applications deal with uncertainty in multiple stages of the process, from price fluctuations to processing times and material quality. Preventive and reactive scheduling techniques, which adapt to uncertainty realizations were developed to approach the scheduling optimization problem under uncertainty. Recently, the use of Deep Neural Networks (DNN) combined with Reinforcement Learning (RL) algorithms became an option to generate policies for decision-making processes. From the context of scheduling, a policy can be beneficial as it produces the schedule online, responding to the needs of the process and the realization of uncertainties in real time. In this thesis, a set of methodologies to approach the scheduling problem under uncertainty for batch systems with Deep Reinforcement Learning (DRL) methods is presented. The state-of-the-art methods available in this area are improved in this work with the development of different techniques that study the translation of the scheduling problem into the framework of Reinforcement Learning.

Contrary to multiple approaches in the literature where the process environment is assumed to be a Markov Decision Process (MDP), the methods presented in this work assume a partial observability of the process. This is called Partially Observable Markov Decision Process (POMDP) and it is useful to handle the uncertainty in the process as well to perceive the evolution over time of the process. To hold this assumption for the scheduling process, the use of Recurrent Neural Networks (RNN) is implemented in order to analyze their performance in the scheduling optimization problem. To the author's knowledge, these networks have not been used before with this purpose, i.e., their implementation in the literature is focused on other features of this type of network, for instance, their capacity to handle inputs of various lengths. This new perspective is compared with implementations framed as MDPs showing the advantage of approaching the scheduling problem in this way.

The decision space of the scheduling problem is usually discretized into a set of actions that can be taken in the online scheduling process with DRL. In this work we propose the use of hybrid agents that can make simultaneous decisions in the process at every decision-step. This allows to expand the applicability of the scheduling agent into more realistic scenarios where more than one decision is required. Moreover, this method extends the nature of the decisions into the continuous space by using DRL algorithms that can work on this domain. This method also allows the integration of the scheduling tasks with others levels in the hierarchical manufacturing systems, e.g., planning or control tasks. To the author's knowledge, attempts to extend the number of decisions, from the agent to these other levels, are not reported in the literature.

The use of DNNs for modeling policies in the scheduling process brings the issue of the "black box" model in which the model does not provide any descriptions of its heuristics for humans to understand the logics behind its decisions. This becomes an obstacle when ensuring that the setting of hyperparameters aligns with the current decisions of the agent. In other words, the agent does not provide any insights about its decisions and the only way to check this is through the final results of the agent. To provide the agent with interpretability, the use of attention mechanisms is implemented in this work. They allow to develop an attention matrix at every decision-step in the process which allows to gather information on the logics behind the decisions. Attention mechanisms have been used in the literature due to their outstanding capacity for building correlations between the elements of the process. To the author's knowledge, the use of this interpretability for inspection and correction of hyperparameters in the scheduling problem is not reported in the literature.

The algorithms from DRL that are used in the methods presented are: a) a variation of Deep Q-Learning and b) Proximal Policy Optimization (PPO). Deep Q-Learning is a well-known DRL method that specializes in problems with a discrete action space; on the other hand, PPO is applicable to discrete and continuous action spaces. These methods are of relatively simple implementation and showed good performance on the works presented in this thesis. The implementations of the methods developed were tested on job-shops, flow-shops and State Task Networks, following objective functions related to reduction of makespan and product maximization. Results showed that the trained agents with the presented methods can generate schedules considering the uncertain parameters of the system, thus making this online scheduling agent attractive for industrial-scale applications. The capacity to react of the agent is demonstrated with the experiments performed on each study. Moreover, the results were compared with different benchmarks including alternative DNNs and optimization solvers. The implementation of DRL methods to approach the scheduling problem under uncertainty demonstrated that this alternative has potential as a reactive online scheduler that can provide reliable responses in short turnaround times. The set of methods presented in this thesis illustrate the advantages and limitations of the incorporation of machine learning into the decision-making process in the context of chemical engineering.

Acknowledgements

The financial support awarded by the Consejo Nacional de Humanidades, Ciencia y Tecnología (CONAHCYT) and the University of Waterloo is greatly acknowledged.

Dedication

To my family and friends.

Table of Contents

List of Figures	xi
List of Tables.....	xiii
List of Abbreviations.....	xiv
Chapter 1 Introduction.....	1
1.1. Research Objectives	3
1.2. Contributions of this research.....	4
1.3. Structure of this thesis	5
Chapter 2 Background and Literature Review	8
2.1. Deep Reinforcement Learning	8
2.1.1 States, Actions, and Rewards.....	9
2.1.2 Markov Decision Processes	10
2.1.3 Partially Observable Markov Decision Process.....	11
2.1.4 The Scheduling Environment	13
2.2. Artificial Neural Networks (ANNs) for Reinforcement Learning	13
2.2.1 Feedforward Neural Networks for Reinforcement Learning	14
2.2.2 Recurrent Neural Networks for Reinforcement Learning.....	16
2.2.3 Self-Attention Modules for Reinforcement Learning.....	20
2.3. Deep Q-Learning.....	24
2.4. Proximal Policy Optimization (PPO).....	27
2.5. Deep Reinforcement Learning for Scheduling Applications	30
2.5.1 DRL Methods for Reactive Scheduling.....	34
2.6. Summary	38
Chapter 3 A Recurrent Reinforcement Learning Strategy for Optimal Scheduling of Partially Observable Job-Shop and Flow-Shop Batch Chemical Plants Under Uncertainty.....	41
3.1. Problem Statement	42
3.2. Methodology	46
3.2.1 Action Space.....	46
3.2.2 Observation Vector.....	48
3.2.3 Reward Function.....	49
3.2.4 Parametric Uncertainty	51
3.2.5 Agent.....	52
3.3. Case Studies	54
3.3.1 Case Study 1	55
3.3.2 Sub-Case 1: Deterministic problem.....	58
3.3.3 Sub-Case 2: Problem Under Uncertainty.....	59
3.4. Case Study 2.....	64
3.5. Summary	69
Chapter 4 A Recurrent Reinforcement Learning Strategy with a Parameterized Agent for Online Scheduling of a State Task Network Under Uncertainty	71
4.1. Problem Statement	72
4.2. Methodology	74
4.2.1 Deep Reinforcement Learning Hybrid Agent.....	75
4.2.2 Environment	78
4.2.2.1 Uncertainty	78
4.2.2.2 Observation Window.....	80
4.2.2.3 Rewards.....	81
4.2.2.4 Exploration methods.....	83

4.2.2.5 Masking Discrete Actions	84
4.3. Results	85
4.3.1 Case Study 1	85
4.3.1.1 Validation of the Method	89
4.3.1.2 Epistemic Uncertainty	91
4.3.1.3 Aleatoric Uncertainty	92
4.3.1.4 Constraint Violation	94
4.3.2 Case Study 2	95
4.4. Summary	99
Chapter 5 Hybrid Deep Reinforcement Learning Agent for Online Scheduling and Control for Chemical Batch Plants	101
5.1. Problem Statement	102
5.2. Methodology	104
5.2.1 Action Space	104
5.2.2 Observation Vector	105
5.2.3 Reward Function	106
5.2.4 Stochastic Parameters	107
5.2.5 Hybrid Agent	108
5.3. Case Study and Results	109
5.4. Summary	113
Chapter 6 Interpretable Online Scheduling for Chemical Batch Plants with Attention Augmented Reinforcement Learning Agents	115
6.1. Problem statement	116
6.2. Methodology	117
6.2.1 Graph Representation of the Environment	119
6.2.2 Uncertainty	120
6.2.3 Action Space and Rewards	121
6.2.4 Self-Attention Hybrid Agent	122
6.2.5 Attention Matrix for Interpretability	127
6.2.6 Attention for Uncertain Parameters	128
6.2.7 Bias Attention Matrix for Non-Stationary Environments	129
6.3. Results	131
6.3.1 Case Study	131
6.3.1.1 Interpretation of the Bias Attention Matrix	132
6.3.1.2 Attention Matrix for Case Under Uncertainty	137
6.4. Summary	140
Chapter 7 Conclusion and Future Work	142
7.1. Concluding Remarks	142
7.2. Future Work	146
References	147
Appendix A.	155
A.1. Collection of Sequential Information	155
A.1.1. Observation Vector	155
A.1.2. Input Sequence for the RNN	156
A.2. Equations and Tables	157
A.3. Flow-Shop Case Study	158
A.4. MDP Approach for Solving Case 1 with Uncertainty	161
A.5. MDP Approach for Solving Case 1 with a Reduced State	163
Appendix B.	165
B.1. Specifications of Case Study 1	165
B.2. Specifications of Case Study 2	167

Appendix C.....	169
C.1. Example for defining an attention matrix.....	169
C.2. Deterministic Case and Under Aleatoric Uncertainty.....	170
C.2.1. Environment	171
C.2.2. Graph Representation	171
C.2.3. Results and Comparison	172
C.3. Specification of Case Studies	175

List of Figures

Figure 1 Reinforcement Learning iterative process	9
Figure 2 Three different ways to move from A to B at three different episodes.	10
Figure 3 Difference between MDP and POMDP.....	12
Figure 4 An Artificial Neuron representation.	14
Figure 5 A Deep Neural Network is a set of layers of individual neurons.	14
Figure 6 Unrolling the network through time.	17
Figure 7 Different models built with RNNs: a) sequence-to-sequence model, b) sequence-to-vector model, and c)vector-to-sequence model.....	18
Figure 8: Long Short-Term Memory Cell.....	19
Figure 9 Graph representation.	21
Figure 10 Self-Attention Module.....	23
Figure 11 Stacked self-attention modules, equivalent to a multi-head self-attention module.	24
Figure 12 a) Flow-shop and b) job-shop configuration.	30
Figure 13 General plant diagram for implementing the framework presented in this work.	43
Figure 14 Method for integrating the uncertain parameters in the training.	52
Figure 15 a) Observations taken at time-interval 3, b) Architecture used for this work.....	53
Figure 16 Plant configuration for case study 1.	55
Figure 17 Final schedule from case study 1 with no uncertainty.....	58
Figure 18 Average violation of allocation constraint at each episode over multiple trainings.	60
Figure 19 Schedule for uncertainty in processing times and demands.	61
Figure 20 Demands and Constraint violation from different trainings with 10,000 episodes each one.	62
Figure 21 Average violation of allocation constraint at each episode over multiple trainings.	65
Figure 22 Schedule from the second case study.	66
Figure 23 State Task Network representation.	72
Figure 24 a) Architecture of the agent and b) architecture of the critic.	76
Figure 25. a) Hybrid agent with multiple discrete actions. b) Hybrid agent with discrete and continuous actions.	77
Figure 26: a) The reward model considered in this work. b) A bell-shaped reward with the maximum reward in the middle.....	82
Figure 27 State Task Network.	86
Figure 28 Deterministic solution for the scheduling problem of Case Study 1 developed with PYOMO.	90
Figure 29 Deterministic solution for Case Study 1 developed with the presented methodology.	90
Figure 30 Learning curves for case study 1 with uncertainty in production of Task 2.....	92
Figure 31 Learning curve for case with partial reduction on the observations.	94
Figure 32 STN for Case Study 2.....	96
Figure 33 Effect of Masking technique.....	97
Figure 34 Hybrid agent with correlational module.....	108
Figure 35 Batch Plant model.....	109
Figure 36 Comparison between policies.....	112
Figure 37 Schedule with three cycles initialized.....	113
Figure 38 Flowchart of the methods presented in this work.....	118
Figure 39 A complete graph with connections between nodes ζ which refer to the state and tasks from the environment. The edges show the existing connections between nodes, each arrow represents two edges and describe the attention from one node to the other.	119
Figure 40 Depiction of input matrix with temporal encoding aggregated. In the figure, three observation matrices $\mathcal{E}t$ are concatenated.	123
Figure 41 Diagram of the architecture from the actor.....	125

Figure 42 Plant for Case Study.	132
Figure 43 Evolution of the attention values to specific nodes in each of the three heads. Each of the lines in the plots show the attention to the node in one of the four time-intervals registered in the observation window, where T1 is the time-interval at t-3, T2 represents t-2, T3 represents t-1, and T4 is the present time-interval t.	136
Figure 44 Attention values for states F, G, and H from Head 1.	136
Figure A 1 Cumulative reward during 10,000 episodes of training.....	158
Figure A 2 Average reduction of allocation constraint violation during multiple trainings.	161
Figure A 3 Flow-shop schedule under uncertainty.	161
Figure A 4 Average violation of allocation constraints at each episode over multiple trainings.....	163
Figure A 5 Average violation of allocation constraints at each episode over multiple trainings.....	164
Figure B 1 Learning curve of the deterministic problem.....	166
Figure B 2 Learning curve for case with Gaussian noise in the observations.	167
Figure C 1 STN representation of reaction-filtering process.....	169
Figure C 2 Structure of the attention matrix	170
Figure C 3 Average learning curve of the agent in deterministic case	173
Figure C 4 Average learning curve for the agent with partial observation	174

List of Tables

Table 1 Adjacency Matrix.	21
Table 2 Job path configuration.....	47
Table 3 Reward decomposition in sub-rewards.....	51
Table 4 Available jobs/actions for case study 1.....	55
Table 5 Explicit reward shaping for Case Study 1.....	57
Table 6 Average number of constraint violations from one thousand evaluations of the MDP and POMDP agents.....	95
Table 7 Average of constraint violations performed in 1000 evaluations of the agent.....	98
Table 8. Flow-rates for the control tasks.....	110
Table 9. Rewards assigned to the actions.	111
Table 10 List of rewards for the experiments (first 5 rows) and production from each training (last 2 rows). Tasks 4 and 5 are progressively incremented in intervals of 20%.	133
Table 11 Highest mean-attention values from the three heads from the agent in a deterministic environment.	138
Table 12 Highest mean-attention values from the three heads from the agent in environment with uncertainty.....	139
Table B 1 Hyperparameters	165
Table B 2 Relations between tasks, units, and processing times.	165
Table B 3 Relations between tasks and states.....	165
Table B 4 Notation for problem statement P2.	166
Table B 5 Values for r_q for each task.	166
Table B 6 Relations between units and tasks in Case Study 2.....	167
Table B 7 Weights of rewards for case study 2.	168
Table C 1 Matrix representation of the nodes and features.	171
Table C 2 Weights of the reward shaping method used in Case Study 1	175
Table C 3 Matrix representation of the nodes and features for Case Study 2 with the addition of feed states A, B, and C.....	175

List of Abbreviations

A2C	Asynchronous Actor Critic
ANN	Artificial Neural Networks
BPTT	Backpropagation Through Time
CW	Cold Water Service
DQN	Deep Q-Learning
DRL	Deep Reinforcement Learning
DRQN	Deep Recurrent Q Network
ELU	Exponential Linear Unit
FNN	Feedforward Neural Networks
HPS	High-Pressure Service
LPS	Low-Pressure Service
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
MILP	Mixed Integer Linear Programming
ML	Machine Learning
MSE	Mean Squared Error
NLP	Natural Language Processing
PDF	Probability Density Functions
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RNN	Recurrent Neural Networks
SAC	Soft Actor Critic
SAM	Self-Attention Module

Latin letters, ordered alphabetically:

A	Attention matrix (Chapter 2)
A	Set of possible realizations of processing times (Chapter 3)
A^π	Advantage of a state-action pair following the policy π
$\mathbf{A}_{k,t}$	Attention matrix generated at time-interval t and training k
a	Action executed in the environment
a'	Next action corresponding to state s'
$a_{t,t',i,j}$	Attention value from the attention matrix at time t . It provides the attention given from the node positions in row i to the node in column j in the attention matrix.
$action_t$	Action a registered at time t and added to O_t
B	Batch with experiences from the environment (Chapter 2)
B	Set of sub rewards b (Chapter 3)
B_{ijt}	Capacity of the task i at unit j
B_t	Current amount of material from state B at a specific time-interval t
\mathbf{B}_k	Bias matrix generated at training k
b	Bias term in an Artificial Neuron
C	Cost information on utility services, raw materials, products, and by-products
C_s	Maximum storage capacity dedicated to state s
c	Number of time-intervals into the past that are registered in O_t
c_i	Value provided by the agent for continuous action i
c_t	LSTM output that keeps long term relations
ce_i	Maximum acceptable value for continuous action i
cte	Value for penalizing violation to discrete decisions
$D_{t \in \Omega}$	Specific realizations from $\mathbf{d}_{t \in \Omega}$ that can take place at the specific time interval t
$\mathbf{d}_{t \in \Omega}$	Vector with all realization of demands for the time-intervals in Ω
\mathbf{E}	Output product from the process
E_i	Total demand for each product i
e	Edges that connect nodes in a graph
F_t^c	Subset of continuous decision variables that belong to the set F_t
F_t^d	Subset of discrete decisions variables that belong to the set F_t
f	Features for describing each node in the graph G (Chapter 2)
f	Mechanistic dynamic model that describes the chemical process (Chapter 5)
f_t	Input to forget gate in LSTM
G	Economic function considering profits, operational times, costs and other services (Chapter 4, Chapter 5, and Chapter 6)
G	Graph representation (Chapter 2 and Chapter 6)
G_t	Gaussian distribution at a given time
g_t	Input to input gate in LSTM
g	Scaling factor for the bias matrix (Chapter 6)
H	Objective function of problem statement (Chapter 3)
H	Time horizon that starts at t_s and ends at t_f , divided into equal-length time-intervals t
h	Number of heads in the self-attention module (Chapter 6)
h	Inequality constraint function (Chapter 5)
h_t	LSTM output that keeps short term relations
I	Set of products to be processed in the plant
I_j	Set of tasks which can be performed by unit j
i_t	Input to input gate in LSTM
$input_t$	Input vector to the agent with a set of observations O_t

J	Set of jobs that can be executed in the plant
\hat{J}_i	Set of job(s) used for producing product i
K	Number of linear layers in the ANN architecture
K_i	Set of units capable of performing task i
K_j	Paths of machines (recipe) that the jobs need to follow
k	Number of observations registered in the state in POMDPs
\mathcal{L}	Loss function based on temporal difference
$L_{p_{m,s},t}$	Binary variable set to 1 if the machine $p_{m,s}$ at the time interval t is processing a product, otherwise set to 0
l_n	Logits (outputs) from the n Neural Network from the actor
M	A small value that substitutes the masked values from the outputs of a decision n
M_s	Set of machines at each stage
N	Input node matrix to the SAM
\hat{N}	Output node matrix from the SAM
N_i	Set of tasks performed in the batch-plant
N_j	Set of available units for processing the tasks
N_k	Set of tasks in the process
N_p	Possible states of the system
N_q	Set of constraints in the system
N_s	Set of states for storage available
n	Nodes identified in a graph (Chapter 2 and Chapter 6)
n	Number of neural networks dedicated to the number of variables (Chapter 4)
n^c	Number of continuous decision variables
n^d	Number of discrete decision variables
n_i	Number of products in the production
n_j	Number of jobs existing in the plant
n_s	Number of stages in which the plant is divided
\mathbf{O}_t	Vector with individual observations o_t from a sequence of time intervals (Chapter 3)
\mathbf{O}_t	Observation window that contains multiple observation vectors \mathbf{o}_t (Chapter 4)
$o_{s,t}$	Time that a machine has been in use from the stage s in the job recorded in $action_t$
o_t	Scalar value in the observation \mathbf{O}_t (Chapter 3)
\mathbf{o}_t	Observation vector with information of the environment at time-interval t (Chapter 4)
o_t	Input to output gate in LSTM (Chapter 2)
Pd_t	Calculated penalty for discrete actions at time t
Pc_t	Calculated penalty for continuous actions at time t
p	Probability of arriving to state s' after taking action a at state s
p_{a_n}	Categorical distribution provided by the agent for every decision n
$p_{m,s}^j$	Path containing the list of machines m for job j
p_i	Processing time for task i
p_m	Probability for action m in decision n
Q	Q-value corresponding to an action-state pair (Chapter 2)
Q	Projected queries from the input matrix (Chapter 6)
Q	Set of sub rewards q (Chapter 4)
Q_val_j	Q-value for job j
q_j	Amount of product that job j produces
R_t	Sum of rewards from the present time t until the end of the horizon T
r	Reward signal
r_m	Sub reward from the environment

$r_{b,t}$	Individual sub reward b at time-interval t
r_t	Reward provided to the agent at time interval t
rew_t	Sum of set of sub rewards at time-interval t
S	Set of stages in batch-plant
S_{st}	Amount of material stored in state s , at the beginning of t
s	Present state of the environment
s'	Next state of the environment
$S_{k,j,e}$	Binary decision variable that specifies the scheduling process for the batch plant
T	Time horizon of the scheduling problem
T	Training time horizon which represents the total number of time-intervals in the training, not in an episode (Chapter 4)
T	Set of processing times $\tau_{k,j}$ that units in N_j takes to complete the tasks in N_k (Chapter 5)
\bar{T}_s	Set of tasks producing material from state s
T_s	Set of tasks receiving material from state s
t	Time-interval from an episode
t_f	Final time of the horizon
t_s	Starting time of the horizon
U	Set of discretized control actions that include $u_{k,j}(t)$
U_u^{max}	Maximum available capacity of the service u
$u_{j,s}$	Variable that defines the starting time of the job j at the stage s
$u_{k,j}(t)$	Control decision variable involved in task k at unit j
V	Projected values from the input matrix
V_i	Designated capacity in the warehouse for product i
V_{ij}^{max}	Maximum capacity of unit j when used to perform task i
V_{ij}^{min}	Minimum capacity of unit j when used to perform task i
V^π	Value function of a state when following policy π
v	Value function hold by the agent
$W_{j,t}$	Binary decision variable that is set to 1 when a job is started at time t , and 0 otherwise
w	Weights from an Artificial Neuron
X_a	Matrix with the processing times of realization a
$x_{k,j}(t)$	State variable from the system for task k taking place at unit j
$\dot{x}_{k,j}(t)$	Derivative of the variable $x_{k,j}(t)$
Y	Gaussian distribution with mean μ and standard deviation σ
Z	Number of linear layers in each of the n neural networks
z	Output from the neuron to be evaluated in the activation function
z_i	Amount of product i that still needs to be produced

Greek Letters, ordered alphabetically:

α	Learning rate hyperparameter
$\beta_{ui\theta}$	Required amount of the utility service u per kg in the task i in the related batch B_{ijt}
δ	Temporal difference value
ε	Capping hyperparameter for policy update
γ	Discount factor for rewards
η	Number of features used to describe each node
ι	Feature dimension after projection
λ	Dimension of outputs from each recurrent cell

μ_t	Mean value from the distribution dependent on B_t
$\xi_{\varsigma,\eta}$	Element from the observation matrix Ξ
π	Policy that the agent holds.
π^*	Final policy obtained after completing the training.
π_f	Policy trained under fixed conditions, i.e., with no uncertain parameters
π_v	Policy trained under uncertain conditions
$\bar{\rho}_{is}$	Proportion of output of task i to state s
ρ_{is}	Proportion of input of task i from state s
σ	Logistic activation function
φ	Vector with Q-values
ϕ	Hyperbolic tangent activation function
ϱ	Features in the input matrix dedicated for temporal encoding
Ψ	Set of uncertain parameters that include fixed (ψ_{fix}) and stochastic (ψ_{var}) parameters
$\Psi_{j,k}$	Mean attention value from the bias matrix.
Ψ_k	Mean attention matrix from the bias matrix.
θ	Function that describes the aleatoric sources of uncertainty w.r.t. time
ω	Function that describes the epistemic sources of uncertainty w.r.t. time
$\Upsilon_{t,k}$	Mean attention value from the attention matrix at time t
$\mathbf{Y}_{t,j,k}$	Mean attention matrix at time t from the attention matrix
Ξ_t	Observation matrix ($m \times l$) that concatenates the information from Γ time-intervals
Γ	Length of O_t in terms of number of time intervals
ς	Set of nodes from a graph representation

Chapter 1 Introduction

The process of scheduling optimization in the chemical engineering industry has gained increasing relevance in recent years, as it holds significant potential to enhance the economics and sustainability of chemical and manufacturing plants. This field aims to build feasible and near-optimal schedules that allow the accomplishment of objectives related to economics, production, stability, and sustainability. Scheduling can be defined as a decision-making process where resources (i.e., equipment and raw material) and processing times are considered to organize and execute tasks over the short term [1]. These tasks consist of unit operations required for the chemical and physical transformation of raw materials into valuable products (e.g., filtration, reaction, and distillation). There are many factors that need to be considered for the optimal functioning of the scheduling process which include the characteristics of the material, prices and requirements of the process, i.e., number of machines, capacities, and service requirements (like cleaning and maintenance). Other factors include the time availability, the demand of products from the clients, and the parameters that have an impact on the scheduling, for example, the kinetics of the reaction or the environmental conditions. Scheduling optimization problem aims to specify feasible and optimal schedules that consider the demands of the plant and the available resources.

As any other real-world planning process, the scheduling of tasks and resources is susceptible to unexpected events that impact the schedule. These realizations may have a direct effect on multiple parts of the production process, for instance, the quality of the product, the delivery time, the final price, or the economic revenue. Thus, it is important to consider and describe these unexpected events as uncertainties in the scheduling optimization problem. Current methodologies addressing scheduling processes under uncertainty are focused on schedule generation and re-scheduling that guarantee an efficient response to uncertainties while maintaining an efficient computation of the solution, [2], [3]. For scheduling generation, a robust optimization approach that considers multiple scenarios can be used to produce a schedule that

remains fixed during the process. Other techniques for scheduling optimization under uncertainty include stochastic optimization, and fuzzy programming methods; thorough reviews for this subject can be found elsewhere [1], [4], [5]. Reactive scheduling, on the other hand, aims to re-schedule during operation once the uncertainty has been realized. This implies a computational effort every time an uncertainty realization takes place. The challenges that these optimization methods face in handling uncertainties can be summarized as follows: a) Schedules need to be flexible to approach multiple scenarios, requiring a trade-off between solution quality and computational demand; b) Computational costs associated with online scheduling, with complexity growing with problem size; c) Most scheduling problems under uncertainty are NP-complete, posing challenges for optimization [1].

In recent years, the application of data-driven methods have been increasing in decision-making processes for chemical process optimization [6], [7], [8], [9], [10], [11], [12], [13], [14]. Reinforcement Learning (RL) methods, which are used to develop policies for making decisions in stochastic processes, have become attractive due to their fast response and generalization capabilities. The policy is embedded in an agent which trains in a simulation (referred to as the environment) to learn how to perform a specific task. The time horizon of the process is divided into equal-length time-intervals; at which the agent makes one decision. The agent receives rewards and penalties according to the alignment of these decisions to the objectives of the task. An advantage of RL methods is that they can be combined with deep learning techniques and use Artificial Neural Networks (ANNs) to model the policy. This approach is referred to as Deep Reinforcement Learning (DRL) and is useful for complex environments, i.e., where temporal operations cannot be readily described with a few equations.

During the training, the agent develops a policy for making its decisions, based on the conditions of the environment. This assimilates a reactive scheduling process with the advantage that, once trained, the agent provides immediate responses rather than requiring a computational resource to solve an optimization problem with the current conditions of the plant. The agent's policy becomes a key feature for approaching the online scheduling problem under uncertainty because it can handle multiple outcomes from uncertain parameters. To achieve this goal, there are multiple considerations when developing the model that will

represent the agent in the process, specifically: the architecture of the agent, the DRL training method, and the exploration methods to learn from the environment. The policy becomes an alternative to current optimization methods as it provides immediate response to multiple possible scenarios.

A common approach to the scheduling problem with DRL consists of using a discrete action space to contain the actions that the agent can make in the environment. This limits the influence of the agent in the environment to only one decision per time-interval. Cases where the agent needs to make more than one decision, for instance in the integration of scheduling and control tasks, are not reported in the literature for chemical engineering facilities. Another relevant aspect of the DRL approach on scheduling problems is that, in literature, the decision-making process is described by a Markov Decision Process (MDP). This implies that a transient model for getting the probabilities to pass from one state s to s' exist and are available. The Markovian property from the MDP implies that the only information required to know the next state s' of the environment after an action a is executed is that one that state s provides. Nevertheless, real world scenarios present multiple situations where the information is spread out in a sequence of states rather than in the immediate state. For instance, information that is provided in an intermittent fashion or a particular process that takes multiple time-intervals to take place, as in scheduling and control problems. In these cases, where the states can only provide a partial observation of the environment because the information is spread in time, a Partially Observable Markov Decision Process (POMDP) results in a more adequate approach for the transient modelling of the environment.

1.1. Research Objectives

The aim of this thesis is to explore and advance the applications of DRL techniques in scheduling optimization process under uncertainty for chemical engineering facilities. The translation of the scheduling optimization problem into a DRL model is the central topic of this thesis for which a set of methodologies were developed and tested. The current PhD study focuses on the following specific objectives.

- Explore the implementation of DRL methods in partially observable environments representing batch chemical plants under uncertainty such as job-shops, flow-shops, and State Task-Networks.

- Evaluate and compare multiple neural network architectures to model the DRL agent, including Recurrent Neural Networks (RNN), Feedforward Neural Networks (FNN), and Self-attention modules. Insights into the capacities of multiple architectures for decision-making in environments with uncertainty will be illustrated throughout the thesis.
- Define the exploration and exploitation techniques required for training DRL agents and hybrid agents in environments of batch chemical plants under uncertainty.
- Approach the integration of the scheduling and control tasks using a hybrid DRL agent, based on a discretized decision space.
- Explore the interpretability of the decision-making process from a DRL agent using attention models. Use these models to confirm the alignment of the rewards with the objectives and constraints from the scheduling optimization problem.

1.2. Contributions of this research

The research conducted in this PhD thesis is expected to provide the following contributions:

- A methodology for implementing a Deep Recurrent Q-Learning algorithm into a general representation of a flow-shop and job-shop which can be scalable to different sizes. The method will describe the translation from the scheduling optimization problem into the elements that conform the DRL implementation (i.e., actions, rewards, environment, and agent). The proposed scheme will consider uncertainty in the demands and processing times of the plant.
- A methodology that takes into consideration a hybrid agent into the scheduling process that allows making multiple decisions that could be discrete or continuous. The hybrid agent expands the action space and allows the implementation in different scheduling processes.
- Based on the same approach with a hybrid agent, an integration of scheduling and control for a flow-shop model is presented. The problem assumes uncertainty in the composition of the raw material before entering the process, and the policy should provide response to multiple possible compositions.

- A method for gathering information that allows the interpretation of the decision-making process from the agent through the implementation of a self-attention module in the DRL agent. To do this, the method uses a graph representation of the environment (plant instance) to build correlations between the elements; under this representation, the focal point of the agent in the plant can be observed.
- Multiple environments used to represent batch plants from the chemical engineering industry that have uncertainty incorporated at some level. These environments are useful to train and assess the performance of multiple DRL agents from literature.

The contributions from this PhD thesis present novel methodologies that apply DRL algorithms to scheduling problems under uncertainty. Overall, these approaches aim to increase the information provided to the agent to make decisions and expand the visibility of the agent. For instance, the agent can learn about the effects of the realization of uncertainties in the process and react to these realizations in short turnaround times. The approaches listed above have not been reported in the literature as is presented in this PhD thesis and represent an advance in the field of scheduling of chemical processes.

1.3. Structure of this thesis

A description of the structure of this thesis is presented next:

Chapter 2 presents a background section and a literature review of the subjects that are relevant for this work. In the Background, a conceptual description of the elements that compose the DRL formulation is presented; namely architectures of neural networks and DRL algorithms. The reader already familiarized with these subjects is invited to skip this section. In the Literature Review, different approaches using DRL for solving the scheduling optimization problem are presented, the gaps in the literature that motivate this research are shown in this chapter.

Chapter 3 presents a methodology that makes use of Deep Recurrent Q-Learning to develop an agent that acts as an online scheduler for flow-shop or job-shop batch plants with zero-wait restriction under uncertainty. The methodology is used for translating the optimization problem into a DRL framework where a discrete set of decisions corresponding to the initialization of tasks in the process. This methodology

explores the capacity of incorporating RNNs into the agent to gain insight from the sequential data from scheduling processes. The methodology specifically approaches uncertainty in the demands and processing times. The implementation of this method is illustrated for the online scheduling of three variations of shops. The work presented in this chapter has been published in the 9th CODIT International Conference 2023 conference [15] and in *Computers & Chemical Engineering* [16].

Chapter 4 presents a framework for approaching the State Task Network representation of a batch plant under uncertainty. The framework allows the expansion of the action space of DRL into a parameterized action space which allows the agent to make multiple decisions (which could be discrete or continuous) at every time interval in the process. The methodology avoids the need for a hierarchical approach, reducing the computational burden required for training multiple agents. The hybrid agent shown in this work uses RNNs to gather sequential information from multiple time-intervals and track the evolution of the processes which provide insight into the uncertain parameters. The framework is tested in two STNs to demonstrate their capacities and limitations. The results from this chapter were presented in the 2024 AIChE Annual Meeting [17] and published in *Industrial & Engineering Chemistry Research* [18].

Chapter 5 presents a methodology which is similar to that presented in Chapter 4 but applied for the integration of scheduling and control tasks. The methodology uses a hybrid agent which takes decisions over the two tasks at different time scales, demonstrating the capacity of the hybrid agent to work simultaneously on tasks that are correlated. The results from this chapter has been accepted for publication on the DYCOPS 2025 [19].

Chapter 6 presents a strategy that uses an attention model for state representation. The attention matrices produced with this module provide interpretability about the decisions made by the agent at every time-interval. This method allows to gain insight into the reasoning of the agent and its alignment with the objective function. Moreover, this method allows to track changes in the attention of the agent due to changes in the reward function, providing a clear insight into how the agent responds in non-stationary environments. Thus, the methodology can provide information about the attention that the agent aims toward relevant conditions in the environment like safety and allocation related constraint violations.

Moreover, the attention to the uncertain parameters can be measured through the self-attention module. The framework is tested on two State Task Networks with a parameterized action space. This work was submitted to a special issue in the journal *Computers & Chemical Engineering*, and is currently under review [20].

Chapter 7 presents the concluding remarks achieved by the present research work. This chapter also provides recommendations for potential future work.

Chapter 2 Background and Literature Review

This chapter is divided into two main sections: the first section presents general background information on DRL methods; the second section is a literature review that presents the state of the art with regards of the approach of the scheduling optimization problem under uncertainty. The first section presents a description of the fundamental parts of DRL followed by a brief introduction to ANNs. This introduction is centered on describing the main features and functionality of the architectures used in this work: FNNs, RNNs, and Self-attention modules. Furthermore, the Deep Q-Learning (DQN) and Proximal Policy Optimization (PPO) methods are explained in general terms to provide the reader with the needed background to follow the methodologies explained in Chapters 3-6. In the second section of this chapter, the approaches that were found in the literature for the scheduling optimization problem under uncertainty are described. First, a description of current methodologies for approaching the scheduling problem under uncertainty is provided. Following, the works from the literature that use DRL methods are presented, specifically those focused on the use of FNNs, RNNs, and Self-attention modules.

This chapter is organized as follows: Section 2.1 provides a background on Deep Reinforcement Learning. Section 2.2 provides insight into the ANNs utilized in the presented methodologies. Section 2.3 provides an overview of DQN, followed by an overview on PPO on Section 2.4. Section 2.5 presents the current state of the art in the approaches for scheduling optimization problems under uncertainty, including applications that integrate Machine Learning (ML) methods. Finally, Section 2.6 provides a summary of the gaps in the literature that are used to motivate the development of the methods presented in this thesis.

2.1. Deep Reinforcement Learning

Reinforcement Learning (RL) is a Machine Learning method used to train an intelligent agent on a specific task in an environment. The environment is a training simulation where the agent develops its knowledge through an iterative process of trial and error and a reward-based system. The training happens

in multiple episodes, which are divided into equal length time-intervals. At each time-interval, the agent inputs the current conditions from the environment, which are referred to as state s , and outputs an action a to that state. Such action is chosen from an action set that is established a priori in the agent which could be discrete or continuous. The agent uses its policy, which is modelled through an ANN, to choose the action that best applies for the state of the environment. Then, the action is applied in the environment which produces a new state s' , which will be sent again to the agent to produce a new action (a'). At the same time, the environment provides a signal r that reinforces (reward) or weakens (penalty) the use of such action in such state. Fig. 1 shows the training loop for the agent's policy, which stops once the episode reached its maximum number of time-intervals. After the training, the agent holds a policy with the knowledge from the environment; this agent can be implemented to work online in the environment to complete the task which was trained for.

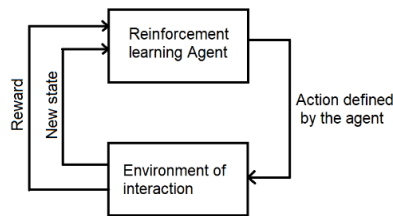


Figure 1 Reinforcement Learning iterative process

2.1.1 States, Actions, and Rewards

An example of a task for the agent is to go from point A to point B in an environment with several obstacles that obstruct the direct way between A and B, with the agent starting at A. Assume that the configuration of the obstacles changes at every episode at which the environment is started, forcing the agent to learn how to navigate in the environment to reach point B. Fig. 2 illustrates this environment with the agent in it, note that there is more than one way to go from A to B, and the agent could take any of these paths.

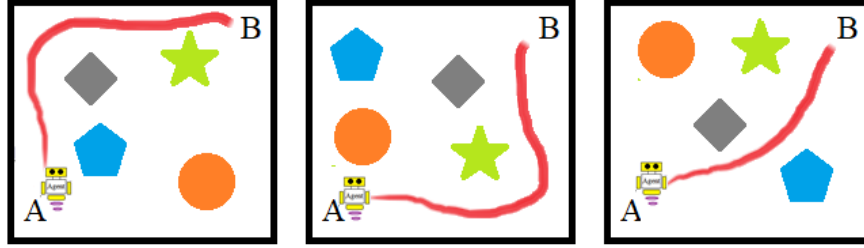


Figure 2 Three different ways to move from A to B at three different episodes.

During an episode from the training of the agent, there is a limited number of time-intervals (also called time-steps) to complete the task. At each time-step, the agent sees the state of the environment and makes the decision on what action to take, which for the case from Fig. 2, is to take one step in a specific direction (up, down, left, or right). The environment then executes this action, provides the corresponding reward r which describes how good or bad that action was, and sends the new state s' to the agent for the next action.

At the beginning of the training, the agent takes random decisions to explore the environment and situations like hitting obstacles, hitting the walls, and not reaching point B will occur. These negative experiences will receive a low reward, while experiences like avoiding obstacles and reaching point B will receive high rewards. These experiences, and their respective rewards and penalties, are gathered by the agent and used to constantly update its policy. At the end of the training, the agent will have developed a policy that recognizes the way to navigate within the environment without hitting any obstacle and reaching point B within the available time-steps.

2.1.2 Markov Decision Processes

The decision-making cycle illustrated in Fig. 1 can be described as an MDP. The ideal MDP process states that the information from the present state is enough to take the next action; in other words, events from the past do not influence the decision at the present time step because only the present state matters to take the next action. RL methods are mostly applied on environments described as MDPs, which are sequential decision-making processes that can be described using a probability function between actions, states, and rewards (Eq. 1) [21]. Recall that a state is a description of the conditions of the environment. The function $p(s', r | s, a)$ expresses the probability Pr which can be described as the likelihood that the

environment at time t will be at the state-vector s' and the reward will be r given the fact that the previous state of the environment was described by s and the action taken by the agent in that time ($t - 1$) was a .

$$p(s', r | s, a) = \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\} \quad (1)$$

It is desired that during the training, the agent acquires a large number of experiences that will help it to learn the probability distribution from Eq. 1. It can be said that by developing the probability distribution from the environment, the agent will learn its dynamics. From these dynamics the agent will be able to predict the state in which the environment will evolve after an action is carried out. Moreover, the rewards can also be predicted for every possible state-action combination. Ultimately, the agent can take the decision path that will provide most rewards.

2.1.3 Partially Observable Markov Decision Process

Fitting an industrial problem as an MDP has been one of the main challenges for the implementation of DRL methods in different industries [22]. Most of the time, the agent cannot rely only on information from the present state because this is not enough for making the next decision, and it must complete this partiality with the information collected in recent history. For instance, suppose that the agent from the previous example has a limited vision which is restricted to what is in front, so its left, right, and back vision is null. Then, the agent needs to complement its spatial information with observations taken previously. This type of environment, where the Markovian property is not entirely satisfied, is referred to as Partially Observable Markov Decision Process (POMDPs). In this process, the agent has access to an observation o_t of the environment instead of a state s_t . The POMDP fits better the representation of real-world case studies because it considers the limitation of the agent to access all the information required to make a decision. There are multiple ways to approach the lack of information in the present observation [21], a simple strategy consists on defining the state s_t as a sequence of observations collected from the history of the process up to some time-interval defined through the variable k (see Eq. 2). Ideally, the complete history of the process can provide all the information available to the agent, but to limit computational expenses, the number of observations should be fixed to a certain number in the past.

$$s_t = o_t, o_{t-1}, \dots, o_{t-k} \quad , k \geq 1 \quad (2)$$

Fig. 3 shows the conceptual difference between the definition of a state on an MDP and a POMDP. The MDP registers the information from the environment at every time-step which is later used for making the next decision. POMDP collects several observations into the past and use them to produce the next action. The processing and correlation of these observations to generate a state representation can be done through different neural network architectures, which will be presented in Section 2.2.

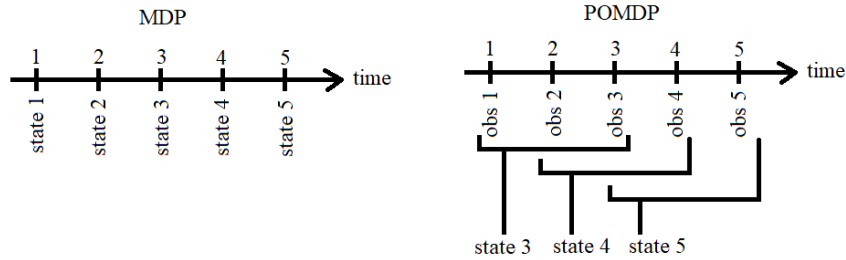


Figure 3 Difference between MDP and POMDP.

The scheduling optimization problem can be approached as a sequential problem where events from the past have a constant influence on future steps. For instance, after a task is initialized, it will remain in the process for several time-steps. Thus, the effects of an action will develop through history and not only in the next time-interval. Then, the approach of the scheduling problem as a POMDP is adequate due to the nature of the problem. This approach has not been reported in the literature for scheduling problems and the usual approach consists of manipulating the environment to become an MDP.

In summary, an environment that follows the Markov property and is partially observable does not provide a clear state at every time interval but instead, it provides pieces of the entire state of the process as it takes place. In the scheduling process this happens due to the gradual development over time of the effects from the actions executed. In the POMDP, the agent gathers observations through time and defines the state of the model. This *hidden state* integrates and encodes the information from the environment so the policy can use it to process the information and then suggest an action out of it.

2.1.4 The Scheduling Environment

The dynamics of the environment are the desired information that is meant to be acquired by the agent through interaction during the training. The collection of the information required to develop the transition model (Eq. 1) is restricted by the information that the environment is capable to provide to the agent. In a batch-plant, for example, the information collected from the process is restricted to the quantity and quality of sensors distributed in the operations. Moreover, the generation of information in a plant is mostly asynchronous due to the variety in length and nature of the processes. For instance, consider two chemical reactions with different conversion rates, one in the order of seconds and the other in the order of hours, the collection of information would conveniently be different for both. This disparity in generation fits in the POMDP outline and demands the approach of the scheduling process as a problem where events from past time intervals influence the decision of the present.

2.2. Artificial Neural Networks (ANNs) for Reinforcement Learning

An agent is an entity that performs multiple functions centered around the interaction with the environment. These functions include the collection of information from the environment, the learning algorithm for updating the policy, the implementation of the exploration methods, and the decision-making process done with the policy. In the learning algorithms called *actor-critic* methods a critic model is also part of the agent and oversees scoring the decisions made through a value function. The agent inputs the state and the reward from every interaction with the environment (see Fig. 1). The policy is a mapping function $\pi(s|a)$ that inputs the states, or observations and outputs the probabilities of selecting each available action in the action set. For actor-critic methods, the value function $v(s)$ is also a mapping function that inputs states and outputs a value related to the quality of that state. This section is focused on the conceptual explanation of the ANNs that will be used for modelling the policy and the value function, namely the FNNs, the RNNs and a special configuration called Self-attention modules. Familiar readers with these architectures and their general functionality can skip this section.

2.2.1 Feedforward Neural Networks for Reinforcement Learning

ANNs were created based on biological neural networks, their fundamental unit is called artificial neuron and the way it is connected to other neurons and how it processes information is very similar (and in some cases better) to the organic counterpart. As shown in Fig. 4, the neuron receives different inputs which are multiplied by parameters called weights (w_1 and w_2 in the figure) which are used to control the influence of an input over the output. Most of the time, a bias term b is also added to the neuron for expanding the expressiveness of the neural network. Then, a summation of the weighted parameters is performed in the neuron to integrate the information. Moreover, an activation function (in this case a step function) is executed to return a bounded output which gives stabilization to the network.

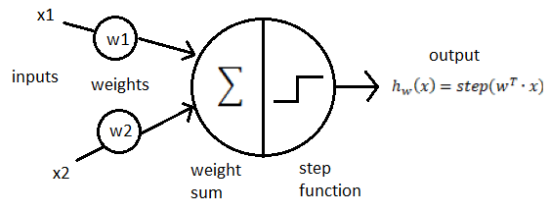


Figure 4 An Artificial Neuron representation.

A layer is an organized set of neurons that receive and share weighted inputs to be processed. When an ANN contains two or more layers then it is referred to as Deep Neural Network (DNN), Fig. 5 shows this representation.

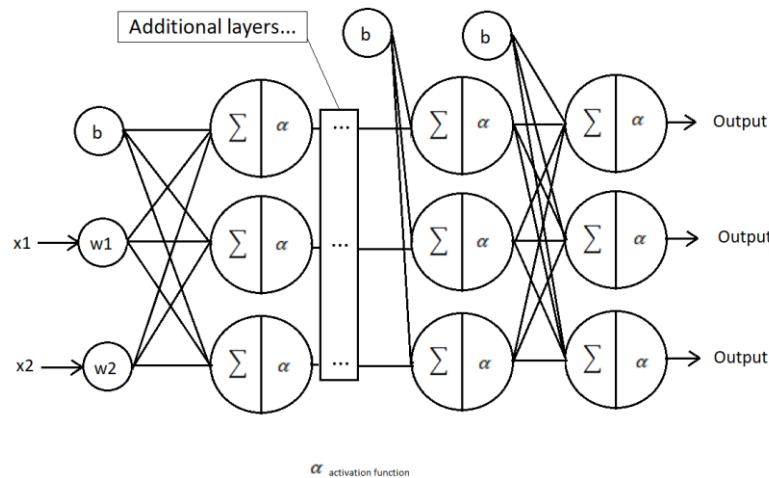


Figure 5 A Deep Neural Network is a set of layers of individual neurons.

A general classification of ANNs is based on how the information is distributed in the network. In the FNN the information flows from inputs to outputs, just as in the network presented in Fig. 5. These networks are used as nonlinear function approximators to model complex processes that may not be easy to represent with a linear model. FNNs are used to model the relation between inputs and outputs from a data set by tuning its parameters (weights) through optimization algorithms. In the general training process, the data is fed into the FNN with initial random weights and bias terms. During the training process, a loss function is used to compare the predicted outputs $\hat{y}_j(x)$ with the real expected values $y_j(x)$ and then reduce this loss using the optimization method. For instance, Eq. 3 describes the Mean Squared Error (MSE), which is minimized by changing the weights and the bias terms in the neurons, i.e., w and b . In Eq. 3, m denotes the number of instances in the data set. Variations of the gradient descent optimization algorithm are commonly used to find the weights that would lead to the minimum in the loss function, currently the most common algorithm for optimization is the Adam optimizer [23].

$$J(w, b) = \frac{1}{2m} \sum_x \|\hat{y}_j(x) - y_j(x)\|^2 \quad (3)$$

For every update in the FNN, the optimizer calculates the gradients for reducing the error from the network. To ensure a constant non-zero derivative at every point of the FNN, the use of activation functions is implemented. Activation functions for neural networks evolved from simple step functions (Eq. 4) to more sophisticated equations, like the logistic function, the hyperbolic tangent function, and the rectified linear unit (ReLU) function, as shown in Eq. 5-7 respectively. Note that z represents the result of the weighted inputs in the neuron.

$$\text{Step function}; (z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \quad (4)$$

$$\text{Logistic function}; \sigma(z) = \frac{1}{1 + \exp(-z)} \quad (5)$$

$$\text{Hyperbolic tangent function}; \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (6)$$

$$\text{ReLU function}; \text{ReLU}(z) = \max(0, z) \quad (7)$$

FNNs are widely used in DRL as policy models, because they are relatively simple networks that can be used to approximate the policy or the value function. On DRL applications, the FNN is trained as the policy using the experiences that are generated online from the environment. During the iterative process, the parameters from the network are updated to perform progressively better in the environment. This architecture can be used in POMDPs environment but would not build any correlation between the observations as the next architecture in this section does (RNNs). Instead, the FNN inputs all the information from the state representation and builds the model with it, without any time consideration.

2.2.2 Recurrent Neural Networks for Reinforcement Learning

As mentioned before, environments that fit on the definition of POMDP as the scheduling process are required to provide more information to the agent to build the state representation used in the policy. The information sent from the environment to the agent corresponds to a sequence of recent events, which can extend as long as needed considering the computational demands that this mean (Eq. 2). The agent could use an FNN as a policy or value network, and input the sequence of events (observations) but this approach would define the inputs as uncorrelated objects in the temporal context. In other words, the FNNs do not treat the inputs as a temporal sequence which is a feature from RNNs; in fact, to do this the entire history of events would be needed at every time interval [24]. RNNs were developed to work with dynamic systems, as their characteristic counter-flow of information helps as a memory for optimizing the use of information collected by the agent.

RNNs are structured so they can receive sequences of inputs and process them in sequence as well. The network takes the first input from the sequence, builds an output and then this output is input again into the network along with the second input of the sequence. This process is called unrolling the network through time and is depicted in Fig. 6. The figure represents a layer of basic RNN cells that input a sequence of inputs and are interconnected in such a way that each input in the sequence will generate an output which will have an influence in the next element. In this way, a temporal influence is exerted along the sequence. As can be seen, the influence can be propagated into the past and the future (see arrows in the extremes of

the figure). This feature counts as a memory in the network which brings the influence from past events into the present sequence. The RNN has different elements to consider: the length of the input sequence, the dimension of each input in the sequence, the number of layers, and the dimension of the output vector.

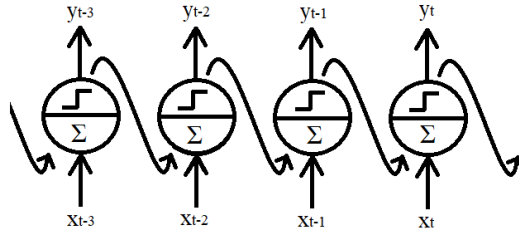


Figure 6 Unrolling the network through time.

The layer of RNNs outputs one output per value in the input sequence (as shown in Fig. 6), but this can be modified depending on the needs of the model. The mathematical formulation of the output is described in Eq. 8, where the activation function ϕ inputs the addition of the weighted inputs plus the weighted outputs from the previous recurrent cell plus a bias term. The activation function ϕ , usually corresponds to a hyperbolic tangent function which outputs values between -1 and +1. This function brings stability while backpropagation takes place every time the network is updated.

$$y(t) = \phi(W_x^T x_t + W_y^T y_{t-1} + b) \quad (8)$$

The RNN can be restructured depending on the need of the outputs. Some configurations are illustrated in Fig. 7, namely, a) A *sequence-to-sequence* model, used for time-series predictions and correlation, for instance, in speech translation where a sequence of words is translated into another sequence of words, b) A *sequence-to-vector* model, used to summarize a sequence in the final output and ensure the influence of previous time intervals, for instance, in prediction in the demand of a product based on information of previous days, and c) The *vector-to-sequence* model, used to generate a time dependent sequence out of the information in one vector, for instance, providing the weather information and make a forecast for the coming days. The size of the RNN can be modified either by increasing the number of neurons in each layer or by increasing the number of layers in of the network. Similar to the enlargement of FNNs, the addition of layers increases the level of abstraction of the network, in other words it searches for relations between relations, which is useful for finding deep correlations between the inputs. On the other hand, adding more

neurons per layer adds capacity to better approximate the function, similar to the addition of parameters in a regression function [25].

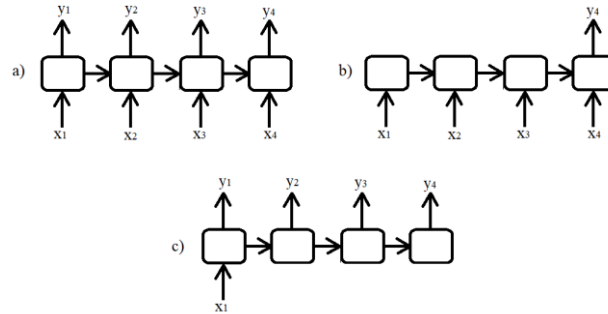


Figure 7 Different models built with RNNs: a) sequence-to-sequence model, b) sequence-to-vector model, and c) vector-to-sequence model.

Basic RNN cells (as those showed in Fig. 6) have a good performance with short sequences that are no longer than 10 elements [26]. Longer sequences approached with basic RNN cells are not capable of being well generalized due to memory issues. In other words, the influence from distant elements from the sequence fades because the cell does not hold relevant information. The Long Short-Term Memory (LSTM) cell allows to maintain relevant features that appear in the long and short term in the sequence. Fig. 8 shows the schematic representation of the LSTM with its three characteristic gates. Each gate regulates the flow of information into the next LSTM cell. The LSTM passes the information through two different outputs which are depicted as c_t and h_t ; the former, passes and maintain information from long term events and is a type of long-term memory, while the latter is for short-term information and has the same function as the output y_t in the basic RNN cell. The function of the forget gate is to discard the information that is not useful anymore and maintain the one that still is. The input gate includes the new relevant information into the long-term memory of the LSTM. The output gate filters the information that will go to the state h_t .

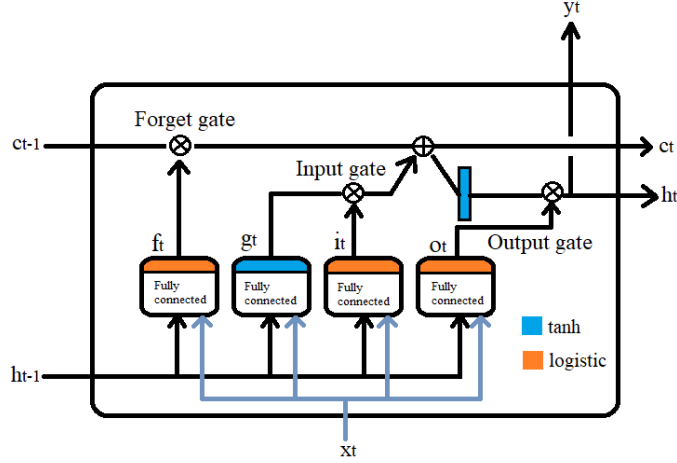


Figure 8: Long Short-Term Memory Cell.

There are four fully connected layers integrated into the LSTM which process the information from the x_t, h_{t-1} and c_{t-1} . The outputs of these layers are f_t, g_t, o_t , and i_t . The output f_t from the first layer controls the parts of the information that will be erased in the long-term memory. The output g_t highlights the most relevant parts from the inputs; this output is intervened by i_t which specifically defines which parts should be maintained in the long-term memory. The output o_t identifies the information that should be dropped out in the h_t state. Eq. 9-14 shows the definition of each layer and the calculation of the outputs c_t and h_t .

$$c_t = f_t \otimes c_{t-1} + i_t \otimes g_t \quad (9)$$

$$h_t = o_t \otimes \tanh(c_t) \quad (10)$$

$$f_t = \sigma(W_{xf}^T x_t + W_{hf}^T h_{t-1} + b_f) \quad (11)$$

$$g_t = \tanh(W_{xg}^T x_t + W_{hg}^T h_{t-1} + b_g) \quad (12)$$

$$o_t = \sigma(W_{xo}^T x_t + W_{ho}^T h_{t-1} + b_o) \quad (13)$$

$$i_t = \sigma(W_{xi}^T x_t + W_{hi}^T h_{t-1} + b_i) \quad (14)$$

where the matrices with the weights and bias terms are referred as W and b with the subscript indicating the corresponding inputs (x and h). The activation function σ indicates a logistic function and the operations \otimes and \oplus are an element-wise multiplication and an addition operation respectively.

The updates in RNNs are performed using a special type of backpropagation called backpropagation through time (BPTT) where the adjustment of the weights take place sequentially to consider the temporal

dependency of the network. One disadvantage of RNNs in general is that during the BPTT they might suffer from gradient vanishing or gradient explosion. These gradient instabilities happen due to the dependency between steps in the sequence which can become unstable as the BPTT goes deeper into the network. When the update reaches the first steps of the sequence, the instability becomes larger and thus it has a large (explosion) or no effect (vanishment) in this part of the network. The memory gates from LSTMs help in reducing this effect by making the training more stable, other techniques include the reduction of the learning rate, or using saturating activation functions as the hyperbolic tangent function [27].

The use of RNNs for sequential information is key for the application of DRL methods in environments described as POMDPs. Their capacity for building correlations between sequential elements through the interconnection of the RNN cells allows them to build levels of abstraction on the collected sequences. Moreover, the LSTM cells permit the influence of events from the past into the present by using the long and short-term information channels (h_t and c_t). These characteristics can be found in other types of architecture that aims for building and highlighting connections between elements in the environment, for instance, the self-attention modules.

2.2.3 Self-Attention Modules for Reinforcement Learning

The use of RNNs for Natural Language Processing (NLP) provided important background for the development on Attention modules. The main problems of RNNs in NLP were related to their memory which was developed with LSTMs but was not enough for large NLP problems and the scattered focus across the sequence [28]. These issues were both approached through a new feature that expresses the relative importance from each input in the sequence and was called attention score. The attention score would be given to every element in the sequence which prevented the disappearance of the elements as the sequence progresses in time, additionally, the attention brought focus to the parts of the sequence that were more relevant. The development on the attention mechanism carried a considerable advantage to the field of NLP because it was believed that RNNs were the only way to treat the sequential dependency on language structures. In this section, a form of attention mechanism called Self-Attention Module (SAM) is

discussed for DRL applications. The prefix *self* refers to an attention module from an environment that pays attention to itself, aiming to explain how multiple elements pay attention to each other. This type of attention describes the correlation between existing elements just like the RNNs does, between the inputs from a sequence. The application of attention modules for DRL that is applied further in this work is based on the work from Zambaldi et al. [29].

An attention model requires the translation of the environment into a graph representation. A graph consists of a structure with nodes and edges $G(n, e)$, in which the edges express the relation between nodes. Fig. 9 shows a graph representation of 4 nodes with edges that display how they interact with each other. Each node in the graph represents an element from the environment with specific features. For instance, the nodes can represent the operation units from a batch-plant, where each node is described by multiple features, e.g., machine capacity, on-off status, processing time, jobs scheduled, etc.

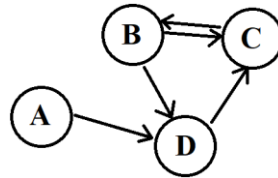


Figure 9 Graph representation.

The edges can be related to the flow path of the process, i.e., the connections between sequential processes. The edges e from this graph can be displayed in a matrix called adjacency matrix that correlates with ones and zeros the existing edges in this graph. Table 1 shows the adjacency matrix for the graph displayed in Fig. 9.

Table 1 Adjacency Matrix.

	A	B	C	D
A	1	0	0	1
B	0	1	1	1
C	0	1	1	0
D	0	0	1	1

The attention mechanism is used to build an attention matrix which provides insight about the relational level that exists between the nodes of the graph $G = (n, e)$. An attention matrix is a squared matrix with

dimensions corresponding to the number of nodes. The attention model serves for analyzing the interactions between nodes and provides an interpretation of data before being processed by other modules, for instance, a set of linear layers. In other words, the attention model transforms the data into a more expressive output. In DRL, the self-attention module pre-treats the input (just as the RNNs) and provides an analyzed version of the state (equivalent to the hidden state) to the agent which is used by the policy to make decisions.

The input to the self-attention module is the node matrix $N: \mathbb{R}^{n \times f}$ where for each node n there is a corresponding list of f features which is common for every node. The expected output from the self-attention module is an updated version of the node matrix, namely $\hat{N}: \mathbb{R}^{n \times d}$ where d is a new dimension of features which could be the same as f or different. The \hat{N} matrix highlights the relevant features from the nodes by using an attention matrix to identify their weight in N .

To do the transformation from N to \hat{N} , the input node matrix is projected into three different matrices referred as *keys*, *queries*, and *values*. In the attention modules (not the self-attention modules), these three correspond to an input x (*query*), a data set with input values k (*keys*) and output values y (*value*). The attention module produces the output y that is similar to the weighted sum of multiple keys k that correspond to the input query x . To find the key values that apply to x , an evaluation of the similarity between x and all the keys k must be executed using a compatibility function. After the similarity is calculated, the value of y can be defined based on the similarity to the keys. The final value of y that corresponds to x is calculated with multiple keys that share a similarity with x [30]. In the self-attention modules, these three elements (*queries, keys and values*) are projections of the same value which is the matrix N . For more information, the reader is referred to [31].

In the self-attention modules, the keys, queries and values are representations of the same matrix, and the module will find the attention matrix by first correlating the representation of keys and queries through a compatibility function. The attention matrix $A: \mathbb{R}^{n \times n}$ provides a map of the similarities between the nodes, and then this matrix is used to highlight the features from the value projection of the N matrix. A high-level

diagram of an attention-based module is depicted in Fig. 10. As can be seen, first the input N matrix is projected through three linear layers with parameters λ , ξ , and ϖ .

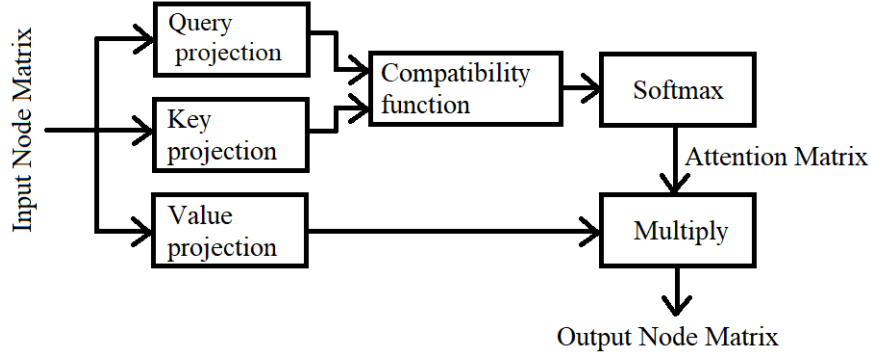


Figure 10 Self-Attention Module

The output matrices from the query and the key projection are compared through a compatibility function called additive attention. Eq. 15 shows this expression that inputs the query and key matrices and outputs an attention matrix. The operation consists on adding the two matrices passed through two independent layers, then apply a nonlinear activation function, which for this case is the Exponential Linear Unit (elu) [32] and pass this result through a third linear layer that transforms the output to obtain a matrix with dimension $|n| \times |n|$.

$$A = SoftMax(Linear_{\vartheta} (elu(Linear_l(Q) + Linear_{\kappa}(K)))) \quad (15)$$

The attention matrix that results from this computation provides values that are not bounded, and which can be very large or very small, which depends on the values from the first projections. To alleviate this, a SoftMax activation function is used to normalize the values from each row by scaling the addition of them to one [32]. The attention matrix is finally multiplied by the value matrix to generate the updated node matrix \hat{N} .

As mentioned, the SoftMax activation function applied in Eq. 15 normalizes the rows from the attention matrix to become values between 0 and 1 and to add up to 1. Applying this transformation also eliminates the potential differences between the values at each row and also helps to highlight where the attention is focused at each node. Nevertheless, this transformation also could dissipate relevant information if the

number of nodes is too large. In other words, the attention from each node could have different focuses, then it is necessary to extend the scope of the self-attention module by increasing the number of *heads* to attend. A set of heads is a set of self-attention modules in which each one produces one different output matrix \hat{N} . Fig. 11 illustrates the stack of self-attention modules to generate multiple attention heads. With this increment in the number of heads, the self-attention module is capable to attend multiple focal points in the nodes which increases the interpretability of the graph.

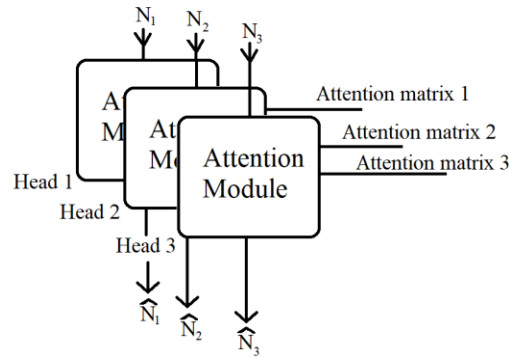


Figure 11 Stacked self-attention modules, equivalent to a multi-head self-attention module.

2.3. Deep Q-Learning

There are multiple DRL methodologies to train an agent, their application depends on the characteristics of the environment, the type of action space (discrete or continuous) and the approximation model desired. Policy-based, Value-based, and Actor-Critic methods are the three classes in which a DRL methodology can be classified. The policy-based method consists in developing a policy that directly provides the action for the state of the environment, an example of this is the REINFORCE algorithm [33]. Value-based DRL models do not provide the action but a set of scores (or values) to the set of actions at every time interval, one value for each action. The action with the highest score is chosen to be implemented in the environment. DQN is an example of value-based models and is a widely used algorithm due to its efficiency for discrete action spaces. Lastly, Actor-Critic methods combine features from policy-based and value-based methods; PPO is an example of this class and will be explained in the next section.

The DQN method scores the actions with a Quality value (Q-value) which is an estimation of the sum of rewards that the agent will obtain from the present state until the end of the episode. The set of Q-values

has a dimension which is equal to the number of actions. A policy π that maps states into actions can be described in terms of the Q-values, by choosing the action with the highest value (see Eq. 16). A definition of the function $Q(s, a)$ is provided in Eq. 17, which states that the Q-value for a given action a in a state s is equal to the reward obtained in that state plus the Q-value calculated for the next state assuming that the best action is chosen in the following state s' ; this action is $\pi(s')$. The γ value is known as the discount factor, which is defined between 0 and 1 and reduces the estimated Q-value for the future.

$$\pi(s) = \operatorname{argmax}_a Q(s, a) \quad (16)$$

$$Q(s, a) = r + \gamma Q(s', \pi(s')) \quad (17)$$

At every time interval, the Q-values are calculated for every action for a given state, from which the best action is chosen. As shown in Eq. 17, the calculation of Q considers the Q-value of the next state (s'), in the expression $Q(s', \pi(s'))$. At the same time, this value implicitly considers the calculation of the next state s'' . Thus, it can be concluded that the value of Q is an estimation of the rewards into the future and until the end of an episode. The discount factor γ is used to fade the consideration of the future rewards, when this value is close to 1, the policy will tend to increase the rewards from the future states, when this value is closer to 0, then the policy will focus on increasing the present reward.

Since the Q-value function cannot be exactly calculated from the environment, it is approximated with an ANN. This approximation is performed using the experience that is being generated from training in the environment. The loss function used to update the parameters of the approximation model is called temporal difference δ and is described in Eq. 18. The temporal difference is derived from Eq. 17 and should be equal to 0 after an infinite training process of the agent. The minimization of the temporal difference can be set as an optimization problem where the updated parameters are those from the ANN that models the Q-value function.

$$\delta = Q(s, a) - r - \gamma Q(s', \pi(s')) \quad (18)$$

The temporal difference uses tuples of elements $\langle s, a, r, s' \rangle$ which refer to as experiences that were collected from the environment at a given time step. Recall that, after the agent realizes the state s , it

generates the action a which is executed in the environment where the reward r is produced and the new state s' is computed. The mean squared error method can be used to compute the error from the temporal difference equation and then use it to compute the gradients. Eq. 19 shows the loss calculation considering a batch B of tuples for computing the loss \mathcal{L} .

$$\mathcal{L} = \frac{1}{|B|} \sum_{s,a,r,s'} \frac{1}{2} \delta^2 \quad (19)$$

The pseudo code for executing the DQN algorithm is presented below (Algorithm 1: Deep Q Learning). The function approximator is an ANN where the architecture could be FNNs, RNNs, or any other architecture. The use of batches for computing the gradients instead of using individual samples helps with the stability of the training. For this reason, a memory buffer called Replay Buffer for saving experiences is needed for training. When the weights are updated, a learning rate α is used to control the changes in the process based on the gradient. Additional methods for accelerating convergence and enhance learning include the use of the epsilon-greedy method and the use of target networks, not mentioned in the pseudocode for the sake of brevity but available elsewhere [34].

Algorithm 1: Deep Q Learning

Initialize weights θ from the approximation function Q_θ

Observe current state s

Loop:

 Select action a and execute it in the environment

 Receive immediate reward r

 Observe the new state s'

 Save the tuple $\langle s, a, r, s' \rangle$

 Create a batch and compute the gradient: $\frac{\partial \mathcal{L}}{\partial \theta} = \frac{1}{|B|} \sum_{s,a,r,s'} \delta \frac{\partial Q_\theta(s,a)}{\partial \theta}$

 Update weights: $\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}}{\partial \theta}$

 Update state: $s \leftarrow s'$

DQN is a value-based method used to train agents in environments with a set of discrete actions. A Q-value function is approximated for learning the dynamics of the system and defines the expected return of rewards during an episode. Challenges of this method include its instability in the learning process which can be mitigated using multiple techniques. Rainbow DQN is a variation of this method which can achieve great performance but with a trade off with the number of hyperparameters that need to be tuned [35]. The

current development of new algorithms is aimed to increase sample efficiency and the stability of the learning process. The PPO method is a policy-based model that incorporated techniques to ensure a more stable learning.

2.4. Proximal Policy Optimization (PPO)

The use of ANNs in DRL supposes a challenge of instability in the training process when the environment is highly complex as in the scheduling process. When the agent is experimenting with the environment it might get in front of situations that drastically change the return of rewards. For instance, assume an agent is learning the way to go through a maze, when it suddenly finds that a promising path is actually a no-end way. This strategy starts to give bad rewards to the agent as it can no longer get closer to the end of the maze, then the agent becomes confused, and it experiences something called policy collapse. This situation also takes place when the number of states in the environment is too large, becoming too difficult for the ANN to generalize such complexity [36]. The solution to this issue is to work with techniques that avoid drastic changes in the parameters of the approximation function. Trust region methods were an option to approach this problem by containing the update of the policy's parameters into a region that is close to the current parameters. Trust Region Policy Optimization is a DRL algorithm that integrates this feature [37]; nevertheless, its implementation is considered complicated as it requires procedures as the calculation of the Kullback-Leibler (KL) divergence from one policy to another, and the use of second-order optimization methods.

The PPO algorithm is classified as an actor-critic algorithm which has two main components modeled with ANNs: the actor which maps the states to actions directly ($\pi(s) = a$) and a critic which estimates how good or bad an action is [38]. The critic computes a metric simply called value, which estimates the average return of rewards from the present state into the future, assuming the policy π is followed. The value function V^π is an approximation function that provides an estimated of the return of rewards (R_t) from the present time t until the end of the episode at time T . The exact value of R_t is described in Eq. 20, where the discount factor γ exponentially decreases with time.

$$R_t = \sum_{t=0}^T \gamma^t r_t \quad (20)$$

The value function is used to compute an advantage function A^π which is described in Eq. 21 which computes a measured advantage of a specific action in a specific state. According to this equation, the advantage is defined as the difference between the Q-values and the value V^π of a state s . The value for $Q^\pi(s, a)$ is calculated from the experience collected from the environment and is not approximated from an ANN. If the value of the advantage is positive for an action, this will mean that the reward of this action (Q-value) has an advantage over the average reward (V^π value) expected for that state. If this value is negative, then the action will bring a lower reward than the average. This is how the critic provides insight into the actor about advantage (or disadvantage) of the actions.

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \quad (21)$$

The integration of the advantage into the actor is given in the objective function from the PPO algorithm which aims to maximize the advantage value. This means that the agent will always aim to choose the action at every state with the largest reward with respect to the average (V^π value). The pseudo code of PPO is shown below in Algorithm 2: Proximal Policy Optimization. First, a set of experiences generated with the current policy are collected from the environment. This is contrary to the replay buffer memory from DQN, where the experiences generated with old versions of the policy are mixed with those from the present policy. PPO uses fresh batches of data for every update made to the agent. Then, the advantage values for the actions from the experience are computed through the return values (approximated with the value function V^π) and the Q-values. The update of the policy aims for maximizing the advantage value obtained at each time interval which in other words is to search for the actions that provide the maximum returns. To avoid policy collapse, the update is made considering the minimum deviation between the parameters from the new and the old policy $(\frac{\pi_\theta(s_t, a_t)}{\pi_{\theta_k}(s_t, a_t)})$, this is done by choosing the minimum argument between the calculated value and a clipped version of it: $g(\varepsilon, A^\pi(s_t, a_t))$, where the value ε is a hyperparameter for capping the ratio between policies. An update to the parameters φ_{k+1} from the value

function approximation is then performed using a mean squared error, similar to the update for the Q-value function approximation from DQN.

Algorithm 2: Proximal Policy Optimization

Initialize weights θ from the policy and weights φ from the value function approximation.

For $k = 1, 2, 3, \dots$ do:

Collect a set of trajectories $D_k = \{\tau_i\}$ using the policy π_θ

Compute the return of rewards R_t

Compute the advantage estimates, based on the current value function V_φ

Update the policy with the objective function θ_{k+1}

$$\theta_{k+1} = \operatorname{argmax}_\theta \frac{1}{|D_k|T} \sum_{\tau \in D} \sum_{t=0}^T \min \left(\frac{\pi_\theta(s_t, a_t)}{\pi_{\theta_k}(s_t, a_t)} A^\pi(s_t, a_t), g(\varepsilon, A^\pi(s_t, a_t)) \right)$$

where

$$g(\varepsilon, A^\pi(s_t, a_t)) = \operatorname{clip} \left(\frac{\pi_\theta(s_t, a_t)}{\pi_{\theta_k}(s_t, a_t)}, 1 - \varepsilon, 1 + \varepsilon \right) \times A^\pi(s_t, a_t)$$

Fit value function by regression:

$$\varphi_{k+1} = \frac{1}{|D_k|T} \sum_{\tau \in D} \sum_{t=0}^T (V^\pi(s_t) - R_t)^2$$

The PPO algorithms is considered a state-of-the-art algorithm for training agents mainly because of its efficiency in sampling, becoming a computationally attractive option. Meanwhile, capping the change of parameters at every update produces stable learning without drastic changes in the decisions of the agent that would derive in a catastrophic learning. For these reasons and its relative simplicity, this algorithm has been widely utilized for the application of DRL methods in multiple environments. Moreover, techniques as mini-batching and the Generalized Based Estimate have been added to Algorithm 2 to enhance performance and computational efficiency [39], [40]. Variations of this and other methods have surged as adaptations for different fields. Scheduling applications is an example of this and recently has emerged as a promising research area because of the features that the policy brings, including online implementation and fast response. In the following section, a review from the literature is presented to show the state of the art of process scheduling.

2.5. Deep Reinforcement Learning for Scheduling Applications

Scheduling problems have been in the literature since the early 1950's, but it was until the 1990's when they became relevant for chemical engineering applications [41]. At this time, the scheduling problem was expanded to consider a wider part of the process in its formulation, such as constraints and process restrictions. As a result of these integrations, the scheduling optimization process in chemical facilities became of relevance in the field of process operations [42], [43]. A first approach to the classification of the scheduling problem is through the characteristics of the machine environment. The process of transforming raw material into a terminated product is called a job. A job requires a set of specific machines to be processed, which can be arranged in multiple configurations. Maravelias provided a description of multiple machine environments that take place in industrial facilities [44]. Among these environments, the usual classes for industrial facilities are the flow-shop and the job-shop; more complex scenarios can be classified as a variation of these two types. Flow-shops are environments where the jobs that need to be done require passing through all the machines in the environment in the same sequence. In other words, the raw material needs to pass through all the machines to be completed. In job-shops, the jobs that need to be processed follow different paths in the machine configuration, and each job may or may not need all the machines in the environment. Then, in the job-shop configuration the routes (or recipes) are required to know the machines needed from a specific product. Fig. 12 shows a schematic representation of the flow-shop and job-shop configurations.

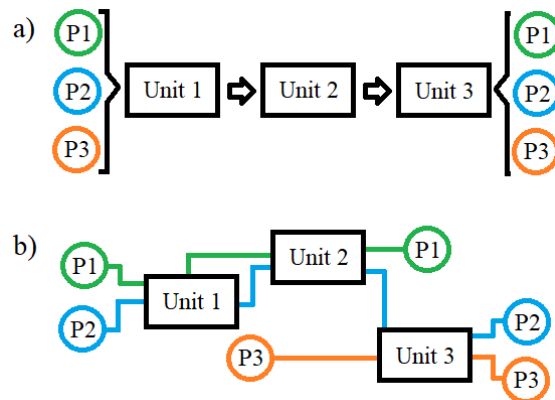


Figure 12 a) Flow-shop and b) job-shop configuration.

Planning and scheduling are two production planning methods widely used in the chemical industry [45]. Although they are synonym words, their concept in the context of industrial production planification is different. The difference between planning and scheduling relies in the length of the time horizon; while planning deals with the allocation of resources for the next weeks or months, the scheduling task is reduced to the organization of production in the order of hours or days. Moreover, planning usually focuses on economic profit maximization, while scheduling tends to search for the feasibility of the process to execute the manufacturing tasks without issue or on reducing the makespan of the process [46]. Planning methods consider objectives in the long term and define a corresponding plan with goals that have to be completed through the scheduling process in the short term. When the plan for a facility requires the constant production of large volumes of products, a cyclic scheduling process is implemented to satisfy the demand. In cases where the demand is variable along the planning period, a short-term scheduling is required which implies the constant reconfiguration of the schedules during the planning horizons [47].

For chemical engineering facilities, a similar classification framework was set by Maravelias to adjust the current scheduling model into the features of chemical processes [44]. The process can be classified as sequential, network, or hybrid process. The sequential processing essentially assumes that there are no splitting or mixing of batches in the process, and a single material is consumed/produced in the process. This process is similar to the flow-shop configuration. Network processing differentiates from sequential processing in the way materials are handled. Batches can be splitted or mixed and then be processed or stored in vessels. The processing units are commonly referred to as tasks, while the the storage units are called states; this nomenclature was established by Pantelides et al. in their work [48]. State Task Networks (STNs) are (together with Resource Task Networks) the most common type of representation for scheduling processes. Hybrid approaches have features of the previous two and cannot be entirely defined as a single class.

An important aspect of scheduling processes is the time representation used in the time horizon. Discrete-time representation consists on a time horizon which is divided into time intervals of equal length; while a continuous time-representation has time intervals of different lengths which are not known *a priori*.

The default time representation of scheduling problems is the continuous time representation, nevertheless, due to the computational requirements, this approach becomes inattractive for modeling [49]. Adding integrality constraints into the time variables allow to switch from a continuous into a discrete time representation. A continuous time formulation would provide a better understanding and modelling of the problem but the discretization has become the most common approach from a computational point of view, a detailed discussion of this topic can be found in [44].

Uncertainty plays an important role in chemical plants since a few of the parameters used for process scheduling are not known *a priori*. Sources of uncertainty in scheduling problems include variability in processing times, unexpected shutdowns, changes in prices, changes in demands, and productivity [4]. It has been shown that uncertainty impacts production and revenue, thus, it is important to design reliable and feasible schedules that can efficiently accommodate uncertainty in the operations. Current methodologies addressing scheduling processes under uncertainty are focused on schedule generation and re-scheduling that guarantee an efficient response to uncertainties while maintaining an efficient computation of the solution, [2], [3]. For scheduling generation, a robust optimization approach that considers multiple scenarios can be used to produce a schedule that remains fixed during the process. Other techniques for schedule generation under uncertainty include stochastic optimization, and fuzzy programming methods; thorough reviews for this subject can be found elsewhere [1], [4], [5]. Re-scheduling aims to correct the current schedule during operation once the uncertainty has been realized. This implies a computational effort every time an uncertainty realization takes place. The re-scheduling process can be done through reactive scheduling methods.

Schedule-generation methods integrate the probabilities of uncertainty realizations into their formulations; as mentioned before, there are several approaches for its formulation: stochastic based approaches, robust optimization methods, and fuzzy programming methods. The stochastic scheduling approach consists of treating each uncertainty as a stochastic variable, described as a discrete or continuous probability distribution. This approach is the most commonly used and is divided into two main categories: as a two-stage or as a multi-stage stochastic programming [1]. In the former, the first-stage variables are

calculated before the realization of the uncertain parameters. Then, after the realization, the effects of these decisions are measured and used to calculate the set of remaining variables from the second stage. The multi-stage approach is an extension of this method with more correction stages. Robust optimization aims to build a schedule solution that can fit to all the possible realization of the uncertain parameters, while finding an optimal solution [50]. The robustness of the generated schedule is shown through a standard deviation model, which shows the level of flexibility of the schedule to multiple realizations of uncertainty. The model is generated with data from different measurable outcomes of the schedule, for instance, the final makespan. Moreover, the fuzzy programming method uses approaches from fuzzy set theory to describe the parametric uncertainty in the process. Fuzzy set theory diverges from set theory in the addition of a parameter of membership from an element to the set. Each element of a set has a grade of membership bounded between zero and one; where zero means that the element does not belong to the set, and one means that the element is fully included in the set [51]. Fuzzy programming integrates random parameters as fuzzy numbers and constraints as fuzzy sets. These elements are used to bound the decision variables into the desired expectation from the user.

Reactive scheduling approaches include the use of real-time dispatching rules that correct the initial schedule based on the current information in the process. The original schedule is built through a deterministic manner and, as the uncertainty realizations take place, the reactive strategy is implemented [52]. Among the techniques that implement dynamic programming to solve the scheduling optimization problem, the application of Mixed Integer Linear Programming (MILP) or heuristic rules are the most common [1], for instance [53], [54], [55].

The challenges that optimization methods for scheduling problems under uncertainty currently face are related to the multiple scenarios in which the process can fall due to the uncertainty in its parameters. Multiple realizations lead to different outcomes in production, then, optimal solutions become difficult to achieve. These challenges can be summarized as follows:

a) Schedules need to be flexible to approach multiple scenarios, requiring a trade-off between solution quality and computational demands.

- b) Computational costs associated with online scheduling, with complexity growing with problem size.
- c) Most scheduling problems under uncertainty are considered NP-complete, posing challenges for optimization methods [1]. NP-complete problems are those that their solution can be verified quickly (in polynomial time) but they are very hard to solve [56].

2.5.1 DRL Methods for Reactive Scheduling

In recent years, the application of data-driven methods have been increasing in decision-making processes for chemical process optimization [6], [7], [8], [9], [10], [11], [57], [58]. DRL methods have become attractive because of their generalization capabilities. Moreover, due to their fast response to complex environments, DRL methods are suitable for online scheduling optimization problems that may be subject to uncertainty [59]. That is, the agent makes decisions considering the current state in the environment, which may also be subject to the potential realization of a few uncertain parameters, e.g., changes in demand, prices or machine availability. In the literature, there are multiple works from different fields that have applied DRL methods for process scheduling and which are presented in the following paragraphs.

The application of DRL agents for solving scheduling optimization problems has been performed under different approaches. Waschneck et al. used multiple agents trained with DQN to produce and deliver the requested lots before the due date in a complex job-shop for a semiconductor's manufactory [60]. The agents were penalized when there were no machines available, which consequently reduced the frequency of this situation. The plant reported a drastic decrease in the number of delayed lots (i.e., from 17% to 1.3%). Hubbs et al. performed a comparison between an agent trained with an Asynchronous Actor Critic (A2C) method and a MILP application to find an optimal schedule for a multi-product reactor that was subject to uncertainty [61]. Hubbs et al. built the schedule of a continuous chemical reactor using a DRL agent trained with PPO [62]. The process is subject to uncertainty in the demands and production interruptions. The agent managed to maintain inventory levels while being subject to uncertainty in demand and equipment availability. Altenmüller et al. proposed an approach to optimize a job-shop environment in

a semiconductor facility using DRL [63]. The agent oversees order dispatching using variable time constraints to make decisions in the plant. The agent reduced the violation of time constraints while scheduling the orders in the process. That study reported that the agent could address rush orders from the system and, once these were satisfied, it completed the rest of the orders.

Lee et al. proposes a DRL method embedded with robust optimization by incorporating an oracle guiding mechanism in the training phase to prevent the agent from getting stuck in local optimal solution [64]. The authors mentioned that the combination of DRL with robust optimization provides learning stability and guidance to the agent during the training. Results showed that the agent produced conservative solutions and robustness against parametric uncertainty. Chen et al. presented a DRL implementation for a large-scale, long-term refinery scheduling problem where the agent is pretrained in simpler problems first, to have an initial knowledge [65]. The technique, known as curriculum learning, is helpful to accelerate the training of the agent. Shao et al. approached the predictive scheduling problem for renewable energy sources in a desulfurization system [66]. The continuous power fluctuations due to weather instability require an adaptive scheduling policy that can change to the drift. On a similar work, Pravin et al. use a DRL agent to schedule the power delivery from a hybrid renewable energy system [67]. The agent is trained to learn an optimal policy that schedules the power dispatch considering the economy of the process. Results demonstrated an optimal power dispatch among the multiple power resources. Ikonen et al. proposes a hybrid method of DRL with an optimizer in which the trained agent takes decisions over the scheduling process [68], i.e., when to perform a rescheduling process and the allocated computing time. With this approach, better closed-loop solutions could be developed against those that were generated by humans. Gao et al. proposed the use of Q-learning for training an agent for scheduling maintenance tasks at coal-bed methane gas fields [69]. DRL is used to overcome the large-scale nature (in parameters and variables) of this problem which requires considerable time to be solved. The proposed methodology provided a more efficient solution than traditional algorithms which are inefficient to meet real time requirements.

Apart from FNNs, the use of other architectures like disjunctive graphs, Convolutional Neural Networks (CNN) and RNNs have been considered in process scheduling. Wu et al. used disjunctive graphs

to make a representation of the scheduling problem and used a DRL agent to execute priority dispatching rules considering uncertainty in the processing times [70]. Bonetta et al. used encoder-decoders from NLP to create priority lists which can be mapped into schedules [71]. Pan et al. proposed an ANN to approach the permutation flow-shop scheduling problem for different scales using RNNs, FNNs, and CNNs [72]. Han and Yang used pointer networks, attention modules and RNNs to prioritize operations sequentially in a flexible job-shop [73].

Since the scheduling problem is a combinatorial problem there is an evident interest in the literature for ANNs that can find correlations among the current state in the plant. Variants of RNNs and attention modules have this feature and are widely used in scheduling problems for job and flow-shops. Wang et al. used LSTM cells to correlate features and solve flow-shop scheduling problems [74]. Pan et al. used LSTMs and attention mechanisms to create an actor and a critic respectively to output a vector of probabilities for each job [75]. Monaci et al. took advantage of the capacity of the LSTM cells to process variable-length sequences and defined an agent's architecture for scheduling in job-shops [76]. Mowbray et al. used RNNs combined with FNNs to influence the agent's network with previous steps and thus, account for uncertainty in the decision-making process [9]. Specifically, the authors retrieve the last hidden state and introduced it as an initialization for the subsequent state evaluation. Wu et al. used a similar architecture with RNNs for supply chain and combinatorial optimization problems where the agent is sensitive to low probability, high severity scenarios [77]. Lang et al. used two DQN agents on flexible job-shops where the first agent makes allocation decisions and the second creates operations sequences for the jobs [78]. The authors used LSTMs to gather and correlate information related to operations that are taking place in the process.

Although the literature that includes the use of attention modules for scheduling optimization problems is scarce, their integration into this field is promising as they are useful for creating relationships and representations in the process. In the work of Chen et al., the authors used an attention module, a disjunctive graph and a sequence-to-sequence pattern to model the job-shop scheduling problem [79]. They proposed a representation learning method for disjunctive graphs to extract the job-shop scheduling problem features using attention mechanisms. They aimed for reducing the makespan of the optimization problem. The use

of these architectures was to overcome the problem of poor generalization of current models and to solve problems of larger scales. The framework is tested in instances of different sizes and seems to work in large instances better than the methods used as benchmarks. In the work of Zhang et al., the authors used graph attention networks to extract relevant information from a disjunctive graph [80]. A Transformer encoder is used to capture competitive relationships among the machines in the process. A soft-actor critic (SAC) and a variation of the DQN method were used to train the parameters of two agents. The agents worked collaboratively to choose 1) the candidate operations for the next time interval, and 2) the suitable machines that meet the criteria for performing those operations. The authors aimed to approach large flexible job-shops and complex scheduling. The SAC agent prioritizes eligible candidate operations and selects one to be implemented. Then, DQN prioritizes compatible machines and chooses one for executing the operation. The model was compared with multiple benchmarks including OR-Tools (a Google optimization library), and other DRL methodologies. The framework showed improved performance against multiple benchmarks. In the work of Wang et al., the authors use a self-attention module to gain insight on the complex relationships that exist between operations and machines in the scheduling process of a flexible job-shop [81]. A particularity of this approach is that it uses two attention blocks to explore the environment. The trained agents were compared with the application of priority dispatching rules resulting in a better performance of the DRL agents. Moreover, the agents showed the capacity to perform well in larger instances compared to the one in which it was trained without significant loss of generalization. Lee et al. presented a modified transformer architecture (which is integrated with attention mechanisms) to approach the job-shop scheduling problem [82]. As in the previous work from Wang et al., the authors reported that the model could be extended to larger instances. Moreover, the trained model outperformed other DRL frameworks including one using dispatching heuristics for scheduling. Magalhães et al. implemented an encoder-decoder ANN for approximating the value function in a Deep Q-learning method [83]. The trained agent is used for making scheduling decisions in a Flexible job-shop. The authors used an RNN and attention modules to enable the agent to handle inputs and outputs of variable sizes and to account for

previous states through the hidden state. The trained agent could achieve better results than a meta-heuristic algorithm used as a benchmark (i.e., the Knowledge Guided Fruit Fly Optimization Algorithm).

In the work of Gebreyesus et al. a PPO algorithm was used for sequentially generate optimal schedules on job-shops [84]. The authors used disjunctive graphs that integrated attention modules to represent the topological structure of the job-shops. They aimed to minimize the makespan of the completion of all the jobs. The agent makes the decision of which operation should be started at every time-interval. As the process advances, the model discards the operations already started from the action space. The model was compared with multiple benchmarks and the results showed an outperformance over other heuristics and baselines. The model showed the capacity to be used in larger instances, providing scalability. In the work of Liao et al., a PPO method that uses attention mechanisms for structuring the agent for solving job-shop problems was proposed [85]. The model consists of a policy that is parameterized by an attention model, which computes the scheduling priority of operations. The aim is to reduce the makespan of the process. The actor calculates the priority of the available operations. The framework was compared with multiple benchmark instances where the strong generalization capability of the model was demonstrated. Yang et al. proposed a policy network based on attention mechanisms for encoding the disjunctive graph representation of a job-shop into a state [86]. The set of actions corresponds to the decision operations that have not been performed in the shop. The aim is to reduce the makespan and the PPO algorithm was used to train the agent. The actor and critic receive the encoding of the attention mechanism and then a set of FNNs are used. The framework was tested against three baselines, which were outperformed. Also, the model could generalize for bigger instances at a cost of more training.

2.6. Summary

The use of DRL methods for approaching the scheduling optimization problem with uncertainty has become an alternative over optimization methods like robust or stochastic optimization. Among the advantages that these methods provide is the nature of the trained policy which builds the schedule in an online fashion. This implies that the policy will evaluate the state of the environment at every step, which

is convenient for approaching unexpected realizations of uncertain parameters. Contrary to reactive scheduling, where the schedule needs to be rebuilt or corrected every time these events take place, the policy provides guidance on how to proceed in the short-term scheduling. The adaptation of DRL methods requires the use of multiple techniques to process the information. The literature has focused on the exploration of multiple architectures of ANNs to model the policy and value functions. These architectures provide different options to gain insights on the state of the environment.

As a relatively new area of research, there are potential new avenues for study in the field of process scheduling. The following gaps in the literature motivated the objectives of this research:

- a) The general assumption of the scheduling problems as an MDP that was found in the literature restricts the use of historical information from the process. The study of these problems as POMDPs is scarce in the literature as it is useful to take advantage of information from multiple time-intervals.
- b) The works in the literature mostly define the action space from the scheduling process as a discrete set of actions. Although this is accurate for applications in job and flow-shops, where the decisions could be deciding between jobs or dispatching rules, this might be a limitation for chemical facilities where other decisions need to be addressed in the scheduling process. Therefore, the need to extend the decision space of DRL agents.
- c) The use of ANNs to generate policies for industrial applications has been limited by the fact that these approximators are not fully interpretable. Then, the dynamics of the process acquired during the training, along with other logics that are behind the decisions made by the agent are not available from the model. Thus, the interpretability to confirm the logics behind the decisions from the agent is required. This could be done through the special features of attention modules to provide insight into the decisions made in the process. To the author's knowledge, this study has not been reported in the literature of scheduling problems.

In the following chapters, the works for approaching these gaps in the literature are presented. These works are focused on studying the applications of DRL on the scheduling problem from different

perspectives. Specifically, the use of RNNs for approaching the POMDP, the application of parameterized agents for multi-decision environments and the use of attention modules for interpretability of the decisions of the agent. These areas of research which have not been studied in the literature will advance the implementation of novel machine learning methods, particularly DRL methodologies, to process scheduling.

Chapter 3 A Recurrent Reinforcement Learning Strategy for Optimal Scheduling of Partially Observable Job-Shop and Flow-Shop Batch Chemical Plants Under Uncertainty

Modelling the scheduling process as an MDP is a characteristic that prevails in the literature and assumes that the agent has access to all the information needed to take the next decision at each time-interval. To the author's knowledge, there are no works in the literature that approach the scheduling process in flow and job-shops with zero-wait restriction as a POMDP using Deep Recurrent Q Network (DRQN). This method is a variant of DQN that uses Recurrent Neural Networks (RNN) and it is useful for environments with partial observability, i.e., the information provided to the agent at each time-interval is insufficient to define the state of the environment. This chapter focusses on scheduling problems where the current states are not informative enough, e.g., systems where information about the processing times of the jobs and product demands that may take place during the scheduling horizon are not available at particular time instances. These conditions are assumed to be uncertain and not available to the agent, which makes this a challenging problem to be solved using the existing DRL methods based on MDPs. The DRQN method is used to handle these problems and train a policy that can return schedules under partial observability.

To assess the capacities of the present method, an environment of a shop that is lowly expressive in information and that does not follow the Markov property (Eq. 1) was created. The environment only transmits the already scheduled occupation of machines, the demand values and the time variable. Uncertainty realizations are realized randomly in the process during the training. In other works from the literature, the required information (from the present and past events) to make an action is provided to the agent at the present time interval, the present work assumes an environment that provides scarce (limited) information and where the DRL agent needs to have access to many states in the past to take a decision.

This chapter is organized as follows: Section 3.1 presents the problem statement, Section 3.2 describes the DRL methodology, Sections 3.3 and 3.4 present two case studies solved with the presented methodology. Finally, Section 3.5 presents a summary of this chapter.

3.1. Problem Statement

In this section, the conceptual and mathematical description of the scheduling optimization problem under consideration is presented. Fig. 12 presents a general representation of the chemical batch plant under consideration. The plant is divided into stages $s \in S$, and at each stage, there is a number of machines that belong to the set M_s ; note that the number of machines from one stage is independent of other stages. Let $p_{m,s}$ represent the m^{th} machines available on each stage s that belong to M_s . For every job $j \in J$ that can be executed in the plant, there is a predetermined path K_j that needs to be followed during the scheduling horizon T . Each path contains the list of machines $p_{m,s}^j$ where the superscript j represents the job that makes use of machine m at stage s . The size of each path K_j is n_s which refers to the number of stages in which the plant is divided. Thus, each element in K_j specifies the machine for each stage that will be used to complete the job j . For the case when a job does not need a machine from a specific stage, a zero is assigned for that specific stage. Note that if a job j can use two or more machines from the same stage to perform a particular task, then all the variations of the job should be declared as independent jobs. For instance, suppose that job $j = 1$ can use machine 1 or 5 ($p_{1,3}^1$ and $p_{5,3}^1$) for the third stage in the path $K_{j=1}$. Since the path only allows one machine per stage, then another job ($j = 2$) with a path $K_{j=2}$ that is equal to $K_{j=1}$, except for the machine at stage 3, needs to be set. Then, both alternatives will be available for the decision process.

Each machine has a processing time defined for the job and stage that is considered. Some of the machines' processing times are uncertain and can take different values from a finite set of realizations. The processing times are defined by the matrix $\mathbf{X}_a \in \mathbb{R}^{+n_j \times n_s}$ where n_j represents the total number of jobs. The subscript $a \in A$ is used to identify the different realizations of \mathbf{X} . The elements of \mathbf{X}_a represent the

processing times $x_s^{j,a}$ for each s stage for every job j . Note that for the stages where no machine is used in a job, a zero is assigned to denote the absence of a processing time.

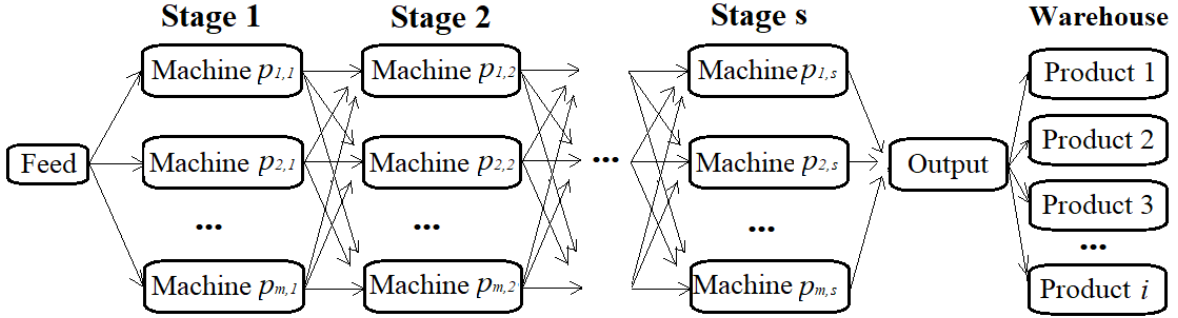


Figure 13 General plant diagram for implementing the framework presented in this work.

The products processed through the jobs j are represented by $i \in I$. Each job produces a predefined amount of product which is denoted as q_j . More than one job can be used to produce a specific product I ; hence, $\hat{J}_i \subset J$ shows the job(s) used for producing product i . During the horizon T , product demands are updated at specific time intervals $\Omega \subset T$. Updates to the demands are defined *a priori* and selected from a finite set of possible realizations. The updates are uncertain to the scheduler until it reaches the time intervals specified in Ω . A realization of the demand selected for updating the product demands is defined with the vector $\mathbf{d}_{t \in \Omega} \in \mathbb{R}^{+1 \times n_i}$ where n_i represents the number of products in the production; thus, the i^{th} element in the vector (i.e., $\mathbf{d}_{t \in \Omega}^i$) refers to the individual demand of each product at a particular time interval t . All the possible realizations of $\mathbf{d}_{t \in \Omega}$ that can take place at the specific time interval t are defined in the set $D_{t \in \Omega}$. The number of sets $D_{t \in \Omega}$ is defined by the number of updates expected to occur during operation (i.e., $|\Omega|$). The sum of the demand selected for each update results in the total demand vector $\mathbf{E} \in \mathbb{R}^{+1 \times n_i}$. The elements in \mathbf{E} correspond to the and can be defined as follows:

$$E_i = \sum_{t \in \Omega} d_t^i \quad (22)$$

The products obtained from the plant are used to satisfy the demand while any leftovers are allocated in the warehouse. Each product i has a designated capacity in the warehouse which is defined as V_i . The scheduling decision process is considered to be performed online and will specify the job that needs to be started at each time interval to fulfill the demands before the end of the horizon T . If no job needs to be

started at a specific time interval, then no action should be executed. The scheduling optimization process of this problem statement aims to reduce the time that it takes to satisfy the demands for each product i in the vector \mathbf{E} . Note that other objectives can be considered, e.g., maximize throughputs, profits, or minimize costs. The objective function considered for this problem is defined as follows:

$$H = \sum_{j \in J} \sum_{t \in T} t * W_{j,t} \quad (23)$$

where $W_{j,t}$ is a binary decision variable that is set to 1 when a job is started at time t , and 0 otherwise. This decision variable is specified for every time-interval in the horizon T . H is the sum of the times at which the jobs are initialized in the process, which also minimizes the makespan of the scheduling process. The scheduling optimization problem is shown in problem (P1). The objective function (P1.a) minimizes H in the horizon T . Eq. (P1.b) is an allocation constraint that uses the binary variable $L_{p_{m,s},t}$, which is set to 1 if the machine $p_{m,s}$ at the time-interval t is processing a product; otherwise, it is set to 0. Constraint P1.b enforces that only one product is being processed on the same machine at any time t . Constraint (P1.c) enforces that the production of the product i should meet the demand E_i and any additional surplus will be added to their corresponding warehouse with capacity V_i . Constraint (P1.d) enforces that the total production of each product i must meet the demands. Constraints (P1.e) and (P1.f) enforce the zero-wait restriction which states that the batch product from each machine must be transferred from one stage to the next immediately after the batch is completed. This restriction is defined with the variable $u_{j,s}$ that defines the starting time of the job j at the stage s . Constraint (P1.f) defines the starting time of the next machine in the stage s of a job j as the time at which the process started in the previous stage plus the processing time $x_s^{j,a}$ of that stage. Constraint (P1.g) prevents any machine from finishing after the end of the horizon T .

P1

$$\min_{W_{j,t}} H \quad (P1.a)$$

$$s. t. \sum_{t \in T} L_{p_{m,s},t} \leq 1, \quad \forall m \in M_s, s \in S \quad (\text{P1.b})$$

$$V_i + E_i \geq \sum_{j \in J_i} \sum_t^T W_{j,t} * q_j \quad \forall i \in I \quad (\text{P1.c})$$

$$E_i \leq \sum_{j \in J_i} \sum_t^T W_{j,t} * q_j \quad \forall i \in I \quad (\text{P1.d})$$

$$u_{j,1} = t * W_{j,t} \quad \forall j \in J, t \in T \quad (\text{P1.e})$$

$$u_{j,(s+1)} = u_{j,s} + x_s^{j,a} \quad \forall j \in J, s > 1, a \in A \quad (\text{P1.f})$$

$$u_{j,s} + x_s^{j,a} \leq T \quad \forall j \in J, s \in S, a \in A \quad (\text{P1.g})$$

The parameters of the total demands of product i (E_i) and processing times ($x_s^{j,a}$) are uncertain and not known *a priori*. Hence, problem P1 can be solved using reactive scheduling methods. In general, the iterative calculation of the solution of P1 at each time interval may become time-consuming depending on the size of the plant. Another approach consists of the application of stochastic programming methods such as robust optimization. Robust optimization would provide a schedule that maximizes the expectation of the objective function for any realization of the uncertain parameters which might derive in highly conservative solutions when the domain of the uncertainty is overly large [87]. Two-stage and multi-stage stochastic programming constitute another option to approach this type of scheduling problem allowing the correction of decisions after the realization of an uncertain parameter, nevertheless, there is a high computational cost involved in their implementation [88].

This chapter presents an alternative approach that makes use of a DRL strategy. The aim is to design an agent that can be implemented online, and which provides scheduling decisions in short turnaround times independent of the size of the plant. The agent makes use of the final policy (π) to make decisions online considering the recent history of the plant, i.e., the state of the plant in the previous time intervals. In the next section, the translation of problem P1 into an RL environment along with the construction of the agent and the training method are described.

3.2. Methodology

In this section, the approach used to solve the optimization problem P1 is described. The DRL method that is used to train the agent is the DRQN, which is used to handle the POMDP that is considered in this environment. In the context of DRL, the use of RNNs, CNNs, Attention modules, or Autoencoders represent alternatives for handling POMDPs [89]. In this work, RNNs were considered mainly because the number of time intervals that are meant to be evaluated for the decision-making process is not large (<1000), moreover, RNNs are easy to train and effective.

The key components from the RL implementation are: (1) The action space, consisting of the jobs in the set J that can be scheduled and initialized by the agent at every time interval t ; (2) the observation vector, which is the batch of information collected at every time interval from the batch plant (from now on called environment) and that is evaluated by the agent to take the next decision; (3) the definition of the reward function used to control and guide the learning process of the agent; (4) the integration of parametric uncertainty in the environment; (5) the architecture of the RNN employed to create the RL agent. Each of these components is explained next.

3.2.1 Action Space

The action space (i.e., the set of jobs J) that can be initialized in the plant is described in Eq. (24). Since the RL method requires the selection of an action at every time-interval t , an *idle state* action is incorporated for the case when the scheduler decides not to initialize a product. Note that the initialization of a job involves not only the start of the first machine in the path but the subsequent initialization of the machines in K_j . As mentioned in the previous section, each job produces a pre-specified amount q_j . Table 2 shows the decisions that are taken with the selection of an action. Note that the processing route, i.e., the path K_j , shows the machines $p_{m,s}^j$ in job j .

$$Actions = [job_1, \dots, job_j, idle\ state] \quad (24)$$

Table 2 Job path configuration.

Action	Processing Route K_j	Quantity produced (m^3)
job_1	$p_{m,1}^1 \rightarrow p_{m,2}^1 \rightarrow \dots \rightarrow p_{m,s}^1$	q_1
...
job_j	$p_{m,1}^j \rightarrow p_{m,2}^j \rightarrow \dots \rightarrow p_{m,s}^j$	q_j

To choose an action, the RL agent evaluates the environment and produces the vector $\boldsymbol{\varphi}$ with quality values (Q_val_j), shown in Eq. 25. The Q-values are estimations that the agent learns to predict through the training and represent an approximation of the cumulative of rewards that the agent receives during the rest of an episode for choosing an action/job. That is, the agent receives a list with a Q-value per action available at every time-interval, and the highest Q-value in the vector $\boldsymbol{\varphi}$ suggests a larger reward in the long term if the action related to that value is executed at that time interval.

$$\boldsymbol{\varphi} = [Q_val_1, Q_val_2, \dots, Q_val_j, Q_val_{idle\ state}] \quad (25)$$

One way used to enhance learning in RL is to constrain the agent to only take actions that are feasible or convenient for a given state of the environment [90]. This technique is referred to as masking and has been employed in this work to make the agent ignore products whose demands were already satisfied. Masking the action vector helps the agent to focus on products whose demands have not been satisfied and encourages the compliance of constraint (P1.d).

Once an action for a job j is implemented and completed in the plant, the demand for that particular job j is decreased by q_j . The iterative execution of actions in the environment during the training and their constant evaluation through rewards and penalties gives the agent the experience needed to distinguish between acceptable and unacceptable actions. Note that the different realizations of the uncertain parameters are included in the experience that is generated during the training. The agent uses this information to refine its Q-value predictions and eventually learns a policy that can handle different realizations in the demands and processing times.

3.2.2 Observation Vector

The agent needs information about the state of the environment (i.e., the plant) to take the next action. In the MDP, there is no other information required from the environment other than that available at the present time. For the POMDP stated in this problem, a sequence of states from previous time-intervals is considered for taking the next action. That is, a sequence of events that starts at some time in the past and concludes in the present time-interval is given as input to the agent to be processed and transformed into the vector φ . The RNN is used to merge and correlate these consecutive past events and create a hidden state (see section 3.2.5). The observation vector \mathbf{O}_t is defined as the set of features from the environment that are considered relevant to take the next action in the schedule. These features are those in the plant that reflect the effects of executing a certain action over another one. For instance, information related to the historic occupation of the machines related to a job; this feature projects the general state of a job which can be used to define an observation vector.

Eq. 26 shows the structure of the observation vector \mathbf{O}_t . The actions the agent has implemented at c time-intervals in the past are registered in the observation vector \mathbf{O}_t . The jobs registered include past jobs that are still being processed in the present time-interval in the plant; on the other hand, the jobs that were initialized in the past and that have been already completed are not included in \mathbf{O}_t . Consider that the number c of actions in the observation remains constant during the training, this implies that at each time-interval, a new action is included in \mathbf{O}_t and the oldest action is taken out from the vector. Note that c must be sufficiently large to let the jobs be followed during their processing before they are excluded from \mathbf{O}_t .

$$\begin{aligned}
 \mathbf{O}_t = & [o_{1,t-c+1}, o_{2,t-c+1}, o_{3,t-c+1} \dots, o_{s,t-c+1}, action_{t-c+1}, \\
 & \dots \\
 & o_{1,t-1}, o_{2,t-1}, o_{3,t-1} \dots, o_{s,t-1}, action_{t-1}, \\
 & o_{1,t}, o_{2,t}, o_{3,t} \dots, o_{s,t}, action_t, \\
 & z_1, z_2, \dots, z_i, t]
 \end{aligned} \tag{26}$$

There are four types of elements that are registered in \mathbf{O}_t : 1) the element $action_t$ lists the decision that was chosen by the agent at time t . This decision could be either a job or the *idle state* action; the jobs are assigned to the $action_t$ by using as an identifier an integer number, whereas the *idle state* action is assigned through a 0. 2) The element $o_{s,t}$ stores the time that a machine has been in use from the stage s in the job recorded in $action_t$. Note that the machines used in each job j at each stage s are shown in the path K_j (see Table 1). If the machine has already finished the process assigned in the job, a -1 is then assigned to the corresponding element $o_{s,t}$. Also, if the machine has not started yet, a 0 is recorded. If no job is chosen in the $action_t$ (*idle state*), the elements $o_{s,t}$ are then recorded with zeros. 3) the demand that is left for each product is specified as z_i which represents the amount of each product i that still needs to be produced. This variable is defined in Eq. 27.

$$z_i = \sum_{j \in J_i} \sum_{t=0}^t W_{j,t} * q_j, \quad \forall i \in I \quad (27)$$

If a demand is completely satisfied, the warehouse for that product is the next to be filled, and the value of z_i appears negative, meaning that the production of that product is for the warehouse and not for the demand. Furthermore, the time-interval t at which the information of the machines and the demands are registered in the environment is also collected in the observation vector. The features stated in \mathbf{O}_t were selected because they can effectively project the effects of an action executed in the environment. This projection is needed by the agent to develop correlations that exist between different \mathbf{O}_t vectors. An illustrative example featuring how the information is registered in the observation vectors is presented in section A.1.a of the Appendix A.

3.2.3 Reward Function

The reward is a scalar value that indicates the performance of the action in the environment at every time interval t . Usually, the reward is provided concerning the accomplishment of one specific objective, for instance, reaching (or not reaching) a specific state. For more complex problems such as scheduling optimization problems, the reward can consist of multiple rewards that motivate the non-violation of

constraints and the alignment of the actions to the objective function, e.g., minimize makespan or maximize throughput. This method is referred to as reward shaping. As shown in Eq. 28, the reward-shaping method involves the use of sub-rewards $b \in B$ that promote different objectives in the policy [91]. The sub rewards are collected and returned as a single (scalar) reward to the agent for their actions at each time-interval.

$$Reward_t = \sum_{b \in B} r_{b,t} \quad (28)$$

To implement the reward shaping method in the scheduling optimization problem P1, the straightforward process consists of assigning a reward to the objective function and each constraint in the problem. Penalties (negative rewards) are given in cases where an action is not aligned with the aims of the objective or when a constraint is violated. At every iteration in the training, the environment calculates the value of each sub reward and adds them up to generate the $Reward_t$. The DRL method uses this reward to update the agent’s policy π in the direction of the maximization of the rewards.

Table 3 lists the sub rewards (as rewards and penalties) designed for the optimization problem P1. The scalar values of the rewards and penalties can be changed depending on the relative importance of the constraints or the user’s preferences. The specific mathematical representation of the reward function is provided in Section 3.3 and is specifically tailored for the case studies used in this work to illustrate and assess the performance of this framework. Back to the description of Table 3, the first row, which refers to the objective function (P1.a) (minimization of the makespan) consists of a sub reward that is given to every product that is introduced in the batch plant. To motivate the reduction of the makespan, the agent receives higher sub-rewards for the early introduction of products within the horizon. This sub-reward also motivates the compliance of constraint (P1.d) since the agent is motivated to initialize products to satisfy the demands. The second and third rows in Table 3 show that a penalty is given if constraint (P1.b) or (P1.c) is violated. Constraints (P1.e) and (P1.f) are not explicitly considered in the reward shaping method because the calculation of the starting times for each machine is estimated by the environment and the decisions of the agent have no influence over this process. If the last constraint (P1.g) is violated because an action allocates a process outside the time horizon T, a penalty is assigned to the agent as shown in the last row in Table 3.

As an additional measure to ensure the likelihood of compliance with the constraints, the $Reward_t$ was set to discard any positive sub reward if a constraint is violated, remaining only with negative values.

Table 3 Reward decomposition in sub-rewards.

Equations from optimization problem	Translation into Reward Function
Minimization of the Makespan (P1.a)	Give higher reward to products introduced closer to the first time-interval.
Allocation Constraints (P1.b)	Penalty for violating a constraint.
Capacity Constraint (P1.c)	Penalty for exceeding the available capacity.
Production Constraint (P1.d)	Reward for initializing products to satisfy the demand.
Time Limitation Constraint (P1.g)	Penalty for allocating a process after the time horizon T .

Alternative methods to reward shaping include the use of constrained RL where the objective function is subject to constraints. This method uses probability distributions to describe the cost (value) of a decision process. These distributions are modified to reduce the value of specific decision processes that represent the constrained space. Constrained RL reduces the expectation of the value function proportionally to the risk that a specific state represents to the agent in terms of rewards [92], [93]. Setting this level of risk in the model allows the agent to prevent highly risky situations and even tolerate certain levels of risk. Reward shaping was implemented in this work because the experiments showed it to be an effective method to reduce the number of constraint violations and the overall makespan.

3.2.4 Parametric Uncertainty

For the agent to learn about the uncertain parameters and produce actions that consider their effects on the schedule and the plant, the agent needs to experiment with different realizations of the uncertainty. Hence, the processing times and the demands need to be changed during the training such that the agent can develop a policy that can provide acceptable actions. Fig. 14 shows a schematic representation of one episode in the training. In step 1, the environment is set with a specific realization of the uncertain parameters. The realizations of \mathbf{X}_a and the demands $\mathbf{d}_{t \in \Omega}$ are chosen using a uniform distribution and then established in the environment. Note that this selection is not known to the agent and learns about them

through the interaction with the environment (step 2) by experimenting with the generation of schedules. In step 3, the DRL method uses the data collected from step 2 (i.e., states, rewards, and actions) to update the agent’s parameters in the direction of maximizing the rewards. After completing an episode, the cycle shown in Fig. 14 starts again for the next episode in the training. The updates are made using an Adam optimizer [94]. Hence, it is expected that at the end of the training, the agent will develop a policy that can perform decisions considering the objective function while avoiding constraint violations.

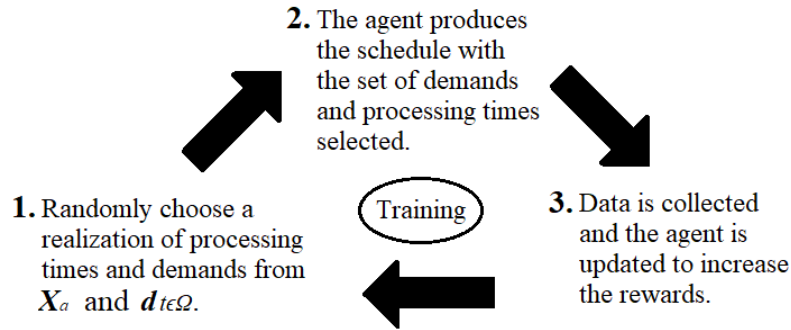


Figure 14 Method for integrating the uncertain parameters in the training.

3.2.5 Agent

The agent has an architecture that contains an RNN to handle the POMDP and a linear layer that transforms the outputs of the RNN into a vector with the Q-values for the actions. The input to the agent is a sequence with length Γ that includes the observation vectors \mathbf{O}_t collected from the process. Therefore, the hyperparameter Γ represents the number of observations from the past that will be provided as input in the agent. Note that this hyperparameter is selected considering the length of the machine’s processing times and can be changed according to the characteristics of the problem. Since the information that is gathered while the process takes place is related to the occupation of the machines, it ought to be considered that the input sequence should accumulate the sequence of time intervals when a machine is processing a product. Hence, the length Γ is the maximum processing time among the possible realizations included in X_a . Eq. 29 shows the input sequence of observations at time t . Note that Γ defines the number of observations into the past that should be included in the input, whereas c in Eq. 26 restricts the number of actions to be registered in \mathbf{O}_t such that only the jobs that are taking place in the plant at the current time

interval are considered. A simple example that illustrates how the \mathbf{input}_t sequence can be constructed for different time intervals is presented in section S.1.b of the Supplementary Material.

$$\mathbf{input}_t = [\mathbf{O}_{t-\Gamma+1}, \dots, \mathbf{O}_{t-2}, \mathbf{O}_{t-1}, \mathbf{O}_t] \quad (29)$$

The RNN uses one hidden layer with λ hidden states in each recurrent cell giving an output dimension of $\lambda \times \Gamma$ values. The number of hidden states is defined according to the size of \mathbf{O}_t , for sizes <1000 , a maximum value of 20 is recommended. Additionally, experimentation with smaller sizes is suggested as different numbers of hidden layers can derive similar results. The output of the RNN constitutes the input for a linear layer that outputs $\boldsymbol{\varphi}$.

The RNN is used to find correlations between the time-intervals that take place in the batch plant. Fig. 15a illustrates how the agent takes four time-intervals ($\Gamma = 4$), at time-interval 3 as input to produce the vector $\boldsymbol{\varphi}$. For the present scheduling problem P1, the agent will develop correlations between the changes in processing times and the demands as the process moves forward.

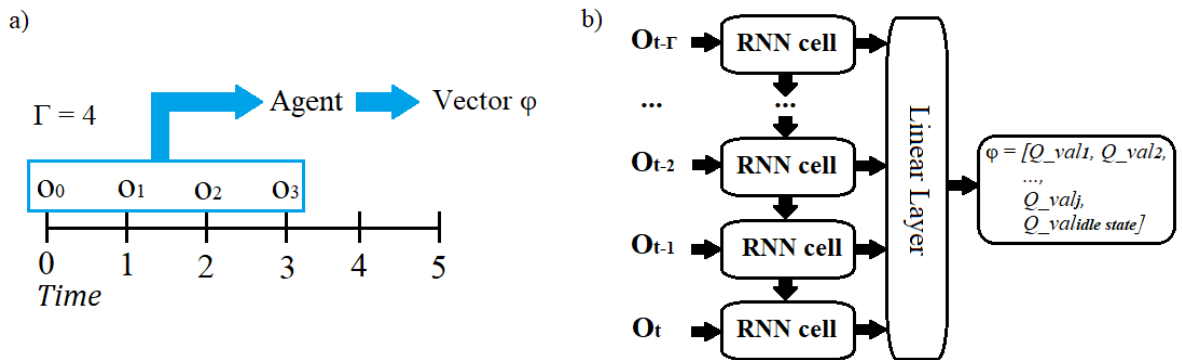


Figure 15 a) Observations taken at time-interval 3, b) Architecture used for this work.

Fig. 15b shows the general architecture of the Neural Network with the input sequence, the processing of this input in the RNN cells, and the transformation into the vector $\boldsymbol{\varphi}$. The vanishing/exploding gradients in RNNs when performing BPTT was not an issue of consideration because the trainings didn't show signs of policy collapse presumably because the number of time-intervals used as the number of inputs per sequence was short (four elements) and the results showed convergence in the training. In the next chapter, the use of LSTM showed a similar performance as the RNN cells used in the work showed in this chapter.

The proposed DRL framework was designed to create an agent that can build online schedules for batch plants that can fit the description provided in Fig 12. The design of the plant allows to extend the number of products that can be processed and the horizon to work on larger instances. Moreover, the hyperparameters listed above can be adjusted according to the scale of the plant in consideration. For more complex scenarios where Γ needs to be larger to attain longer processes, a different ANN architecture could be employed using different types of recurrent cells. These changes can be readily implemented in this framework. On the other hand, the features registered in the observation \mathbf{O}_t , which include data about the processing time and the state of the production and demands, are sufficient to provide informed inputs to the agent. Once it has been trained, the time it takes the agent to provide a decision for a given state is in the order of milliseconds despite the scale of the plant. The evaluation of the agent does not perform any calculation as in other optimizers to take a decision, the decision-making process only consists of the evaluation of the ANN for the given input. The next section provides details of the implementation of this framework in different cases.

3.3. Case Studies

The methodology presented in the previous section was tested using two cases adapted from the literature. The performance of the proposed framework was tested by comparing the agent trained with and without uncertainty, and under different conditions, i.e., plants of different sizes and under different realizations in the uncertain parameters. The two cases presented here involve job-shop scheduling problems for chemical manufacturing systems. For brevity, a third case study featuring a flow-shop batch plant is presented in section A.3 in the Appendix A. The source codes for these three implementations can be found at https://git.uwaterloo.ca/ricardez_group/luis-ricardez-sandoval-drqn.git.

The sources of uncertainty that impact scheduling decisions are related to processing times, which can be related to slow or fast reaction rates in chemical reactors, and product demands, which can change unexpectedly due to changes in market conditions. The first case study presents a small job-shop plant of four products and four stages and is divided into two instances: scheduling with and without uncertainty in

processing times and demands. The second case study is a larger job-shop plant involving ten stages and ten products whose production is subject to uncertainty in the processing times of four machines and uncertainty in product demands, which are realized during operation. The training performed for the three case studies was implemented on a processor Intel Core i5-7500 CPU at 3.40 GHz.

3.3.1 Case Study 1

The plant under consideration is shown in Fig. 16 and involves four stages ($|S| = 4$). The plant is used to produce four products ($|I| = 4$) which can be labeled as Pr1, Pr2, Pr3, and Pr4. Stages one and four have one machine, whereas stages two and three have two and three machines, respectively. At the end of the plant, there is a warehouse available for each product. The time horizon for this process is set to $T = 50$. A process is completed once it reaches the fourth stage, and it will be stored in the warehouse only if the demand is already fulfilled.

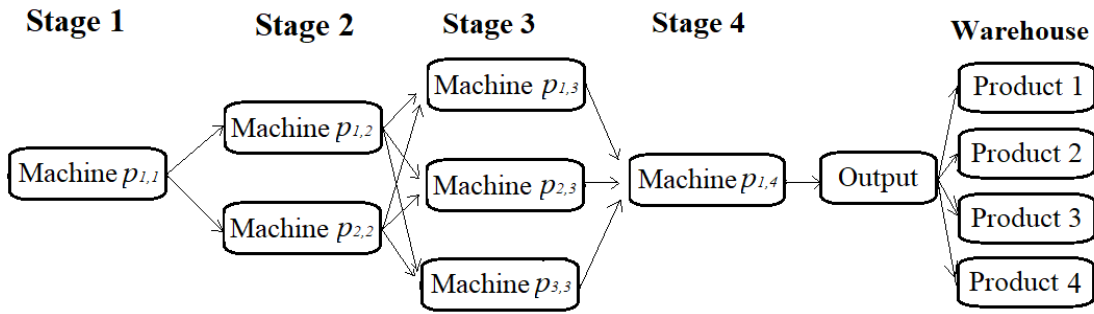


Figure 16 Plant configuration for case study 1.

Table 4 describes the available jobs, the paths K_j in the job-shop that need to be followed by each job, and the corresponding quantity q_j (m^3) that is produced from each job j . Note that a product i can be produced through different jobs which differ in the machines used and the quantity produced. The vector of actions (Eq. 24) contains the elements displayed in the second column (Jobs in J) of this table plus the idle action.

Table 4 Available jobs/actions for case study 1.

Product	Jobs in J	Processing Route K_j	Output q_j (m^3)
Pr1	job_1	$p_{1,1}^1 \rightarrow p_{1,2}^1 \rightarrow p_{3,3}^1 \rightarrow p_{1,4}^1$	80
Pr1	job_2	$p_{1,1}^2 \rightarrow p_{2,2}^2 \rightarrow p_{3,3}^2 \rightarrow p_{1,4}^2$	100

<i>Product</i>	<i>Jobs in J</i>	Processing Route K_j	<i>Output q_j (m^3)</i>
Pr2	job_3	$0 \rightarrow p_{1,2}^3 \rightarrow p_{1,3}^3 \rightarrow p_{1,4}^3$	50
Pr2	job_4	$p_{1,1}^4 \rightarrow 0 \rightarrow p_{2,3}^4 \rightarrow p_{1,4}^4$	80
Pr3	job_5	$p_{1,1}^5 \rightarrow p_{2,2}^5 \rightarrow p_{1,3}^5 \rightarrow p_{1,4}^5$	50
Pr4	job_6	$p_{1,1}^6 \rightarrow p_{1,2}^6 \rightarrow 0 \rightarrow p_{1,4}^6$	80

This case study considers uncertainty in the processing times of a few machines and in the product demand realizations during the process. Eq. 30 shows the set of matrices \mathbf{X}_a that contains all the possible realizations of the processing times. As mentioned in Section 3.2.4, during the training, each of the matrices represents a realization of the processing times and is selected randomly at each episode during the training to define the processing times of the machines. Note that the rows in \mathbf{X}_a represent the jobs whereas the columns represent the stages. For instance, the processing time of the machine in stage 4 for job_3 (row 3, column 4) is of one time unit. As shown in Table 4, this value corresponds to machine 1 at stage 4, i.e., $p_{1,4}^3$ according to $K_{j=3}$. For this case study, the processing times subject to uncertainty are located at stage three (third column) in job_1 and job_2 (first and second row) in each matrix shown in Eq. 30. Note that the agent has no information about the processing times of any job and this knowledge is acquired during training.

$$\{\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3\} = \left\{ \begin{bmatrix} 1 & 2 & 3 & 2 \\ 1 & 2 & 3 & 2 \\ 0 & 2 & 2 & 1 \\ 2 & 0 & 2 & 2 \\ 2 & 2 & 2 & 1 \\ 1 & 2 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 2 & 2 \\ 1 & 2 & 2 & 2 \\ 0 & 2 & 2 & 1 \\ 2 & 0 & 2 & 2 \\ 2 & 2 & 2 & 1 \\ 1 & 2 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 2 \\ 0 & 2 & 2 & 1 \\ 2 & 0 & 2 & 2 \\ 2 & 2 & 2 & 1 \\ 1 & 2 & 0 & 1 \end{bmatrix} \right\} \quad (30)$$

The initial demand is set to 80 m^3 for each product, thus, $\mathbf{d}_{t=0} = [80, 80, 80, 80]$. The demand for each product will be updated at the times specified in the subset $\Omega = \{10, 20, 30\}$. The possible demand realizations used at each time in Ω are assumed to be the same and are listed in Eq. 31. Each vector in the set contains the update for the four products. So, for instance, if the first vector, $[120, 90, 100, 90]$, is selected for $t = 10$, then the demand of the first, second, third, and fourth products would be increased in 120 m^3 , 90 m^3 , 100 m^3 , and 90 m^3 respectively. The selection of the demands $\mathbf{d}_{t \in \Omega}$ is performed randomly

using a uniform distribution. The demands selected are integrated into the production demand at the times stated in Ω . The capacities of the four warehouses in m^3 are $V = \{80,80,80,80\}$.

$$D_{10} = D_{20} = D_{30} = \begin{pmatrix} [120, 90, 100, 90], \\ [100, 110, 110, 40], \\ [100, 120, 100, 110], \\ [100, 120, 40, 110] \end{pmatrix} \quad (31)$$

The specific reward-shaping method considered in this case study is described in Table 5. These sub-rewards follow the objective and constraints in P1 as presented in Table 3. The second column in Table 5 provides a mathematical description of the sub rewards. Moreover, the values selected for each reward are also reported in this table. The definition of these values was made through an experimental process, looking for a set of rewards and penalties that would make the agent avoid constraint violation, complete the demand, and reduce the makespan of the process. The third column in Table 5 describes the aim of the specific sub-reward.

Table 5 Explicit reward shaping for Case Study 1.

Equations from P1	Reward shaping function	Translation into Reward Function
Minimization of the Makespan (P1.a)	$R1 = \begin{cases} 0.2 * \sum_{t=0}^t W_{j,t} & \text{if } \sum_{t=0}^t W_{j,t} \geq 1 \\ 0 & \text{if } \sum_{t=0}^t W_{j,t} = 0 \end{cases}$	If a job is allocated successfully, a reward of +0.2 is given for the rest of the horizon.
Allocation Constraints (P1.b and P1.d)	$R2 = \begin{cases} +8 & \text{if } W_{j,t} = 1 \\ -3 & \text{if } W_{j,t} > 1 \\ 0 & \text{if } W_{j,t} = 0 \end{cases}$	Reward of +8 if the job does not have any conflict with any job. Penalty of -3 if there is a conflict.
Constraint for Production Capacity (P1.c)	$R3 = \{-3 \text{ if } \sum_{j \in J_i} \sum_t^T W_{j,t} * q_j > V_i + E_i\}$	If the addition of a job exceeds the plant capacity, then a penalty of -3 is given.
Constraint for time limitation (P1.g)	$R4 = \{-8 \text{ if } L_{p,m,s,t} = 1 \text{ and } t > T\}$	If a machine is activated after the horizon give a penalty of -8.

A set of 3,500 episodes, where each episode represents a full horizon run of the plant, was used to train the agent, this number of episodes was set after experimenting with different lengths of training. It was noted that after 3,500 episodes of training, the policy converged into a stable response. A learning rate $\alpha =$

$5e - 5$, a discount factor $\gamma = 0.7$, and an epsilon-greedy strategy with a starting value of $\varepsilon = 0.8$ were defined for the agent’s training. The complete set of hyperparameters is listed in Table A.2 in Section A.2 of the Appendix A.

3.3.2 Sub-Case 1: Deterministic problem

To ensure that the framework from Section 3.2 was designed to solve problem P1 presented in Section 3.1 and that the penalizations and rewards from Table 5 guided the agent satisfactorily, the previous case study was first solved without uncertainty. For this instance, only one realization of the matrix \mathbf{X} in Eq. 30 is considered, i.e., \mathbf{X}_2 . The demands assigned are $\mathbf{d}_{10} = [120, 90, 100, 90]$, $\mathbf{d}_{20} = [100, 110, 110, 40]$, $\mathbf{d}_{30} = [100, 120, 100, 110]$.

Since the processing times and the demands were known *a priori* and fixed in every episode, the RL framework found a unique schedule for problem P1. Fig. 17 illustrates the schedule obtained with the policy of the agent. The agent did not violate constraints during operation and was able to organize the schedule online to fulfill the demands. The red vertical lines shown in Fig. 17 indicate the time-intervals when an update to the demands was performed (i.e., values from Ω). At the end of this process, the demands were fully satisfied and a surplus of 3% of the original demand was produced and allocated in the warehouse. This figure also shows that the agent avoids the initialization of products in the last four time-intervals of the process, which is desired to promote the reduction of the makespan and to prevent processes from finishing after the horizon of $T = 50$.

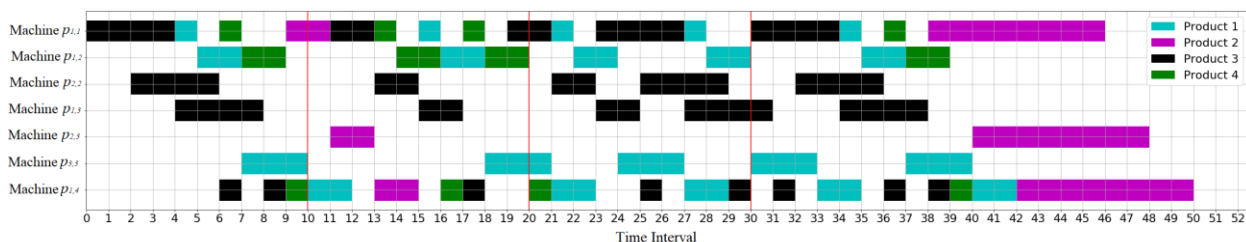


Figure 17 Final schedule from case study 1 with no uncertainty.

Note that the distribution of the schedule is not homogeneous as most of Pr2 was left for the end of the schedule to be produced but this can be changed if their production is prioritized with other sub-rewards. One training took approximately 1,509 seconds. After the agent was fully trained, the time it took to provide an action for a given state was $8.84e-4$ seconds on average. Results from this scenario confirm that the proposed DRL strategy can produce online schedules that minimize makespan and meet the product demands without any constraint violations.

3.3.3 Sub-Case 2: Problem Under Uncertainty

The instance with uncertainty in processing times and product demands previously described is considered next. This instance used the same hyperparameters and number of episodes, i.e., 3,500. The final policy π^* obtained from the training was able to achieve stable results in the presence of uncertainty. The agent's policy was able to fully complete the demands for the possible demand realizations considered in the sets D_{10} , D_{20} , and D_{30} . Nevertheless, the agent also committed a small number of violations in terms of the allocation of the products. Fig. 18 shows the number of violations in the allocation constraint (P1.b) produced at every episode during the training. At the beginning, the number of violations is high, and a progressive reduction is observed which can be related to the agent's learning process. During the last 1,000 episodes, the agent was not able to keep reducing the number of violations, which indicates that it achieved a stable policy that will not change considerably with additional training. According to this figure, the agent commits no more than five violations during the entire process horizon using the final policy π^* . Note that each decision used to initialize a job within the plant contemplates the future allocation of three to four machines. This means that the incorrect initialization of one job can result in three to four machines receiving more than one product.

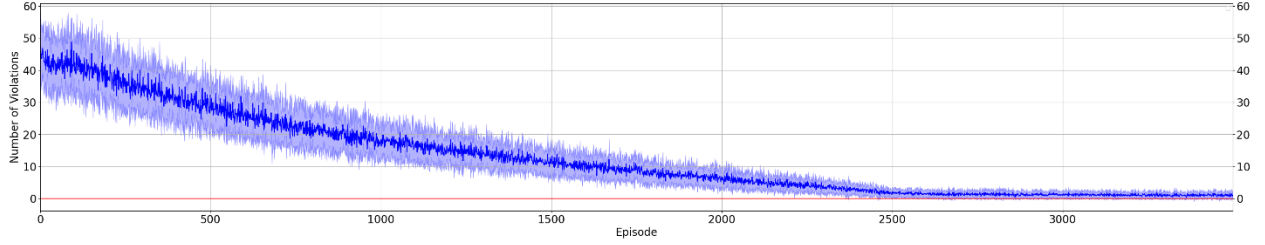


Figure 18 Average violation of allocation constraint at each episode over multiple trainings.

The agent reached a stable learning at the end of the training in Fig. 18, the variation at the end is related with the different realizations of the uncertain parameters and their effect in the final schedule. Also, this happened because the actions under uncertainty could have different outcomes in the environment. That is, for a given \mathbf{input}_t the action that implements a job_j might have different values for Q_{val_j} , then the ANN that is in charge of learning this unique (sole) value will be constantly changing due to the variation in the processing times of job_j . Additional information related to the states of the machines and the processing of each product could provide more information about the process and help the agent reduce the number of overlaps in machines from Fig. 18. The extension of this feature is contemplated for future work.

A schedule without any constraint violation that was generated during the training is shown in Fig. 19. The demand was fully satisfied and the overproduction of 9.37% concerning the total demand was placed in the warehouses. For the particular instance shown in Fig. 19, the matrix \mathbf{X}_2 described the processing times for the jobs and the demands assigned were $\mathbf{d}_{10} = [120, 90, 100, 90]$, $\mathbf{d}_{20} = [100, 110, 110, 40]$, $\mathbf{d}_{30} = [100, 120, 100, 110]$. As in the previous schedule shown in Fig. 17, the agent left most of product 2 for the end of the process. The agent uses the whole horizon of 50 intervals to distribute the production and complete the demand on time. It was observed that the agent preferred to start the jobs with larger q_j in the cases when there was more than one job to produce a product. Jobs 1 and 2 for Pr1 (cyan color in the figure) were subject to uncertainty and were effectively inserted into the schedule.

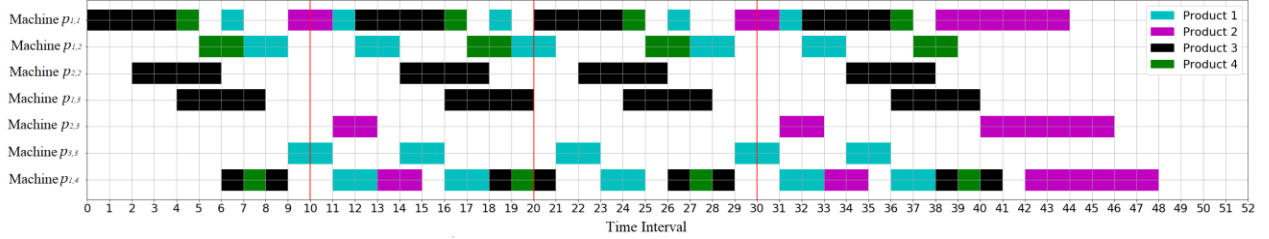


Figure 19 Schedule for uncertainty in processing times and demands.

The final policy π^* was validated using 10,000 instances with different combinations of the uncertain parameters \mathbf{X}_α and $\mathbf{d}_{t \in \Omega}$ shown in Eq. 30 and Eq. 31. The policy did not result in any constraint violation on 52.31% of the instances and fully satisfied the demand in 97.23% of the instances. Statistically, at each instance, there were at most 2 violations of the allocation constraint during the scheduling horizon. From a deeper understanding of the decision-making process, it was found that the violations made in the process were a consequence of one action (on average) during the process. About the total number of actions taken in the process, this action represents 1.7% of all the decisions. A comparison between Fig. 18 and Fig. 19 show that the agent generates similar strategies for generating the schedules; for instance, production of Product 2 is left at the end in both cases. Nevertheless, the agent reacted different in the cases under uncertainty where it became conservative to avoid as much as possible constraint violations. On the other hand, in the deterministic case the agent learnt a unique solution for the problem that was completely defined for the agent. It would be expected that a larger variation of schedules would be observed if, for instance, not all the products were produced at every episode or if a larger number of machines were set in the environment, each one with different requirements, e.g., products that can process, or requirement of times for cleaning.

To reduce the number of constraint violations, specifically machine overlapping, the penalty for committing this violation was increased in the sub-reward R2 (Table 5) such that the agent would focus on avoiding this constraint violation. This reward retuning initially returned better results in terms of reducing the number of violations as the penalty was increased until a minimum point where the agent could not reduce it further. If this penalty threshold is surpassed and the penalty becomes too large, then the agent,

which aims to maximize its overall collection of rewards during a process, will likely opt for producing the minimal portion of products that would warrant no constraint violations. This behavior would have a direct impact on the reduction of demand satisfaction. Fig. 20 shows the implementation of this concept, a set of trainings, with 10,000 iterations each one, was performed with different values of the penalty when a constraint was violated. The values of the penalty were -3 (the one used in this case study), -6, -9, and -100. From this figure, it can be concluded that, at the limit, the agent will eventually leave more demand unattended to the point where the best option would be to not produce any product and completely avoid the risk of constraint violations.

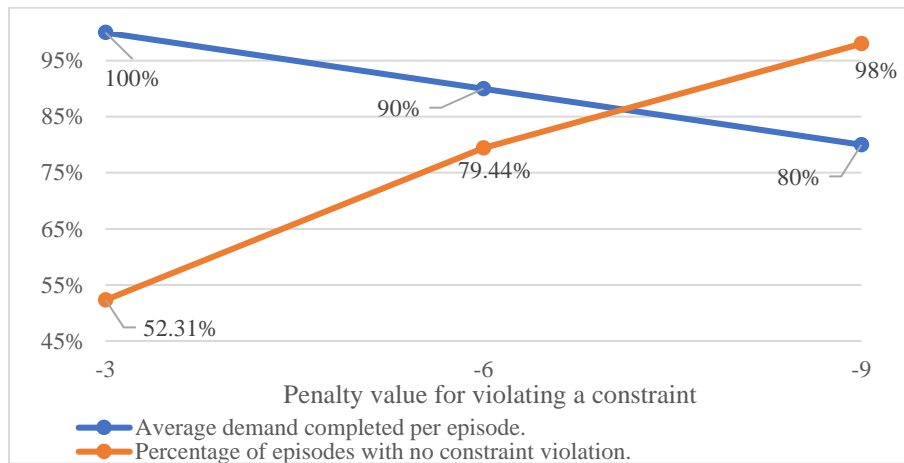


Figure 20 Demands and Constraint violation from different trainings with 10,000 episodes each one.

It was demonstrated that for the second case that considers uncertainty in processing times, the policy obtained from the training phase was prone to commit several violations. In practice, the overlapping of machines is an infeasible scenario. To account for this condition, two heuristics can be implemented to correct for these actions. After a decision is sent from the agent into the environment, the latter can predict if there will be any conflict with the schedule of machines from jobs that already started. Recall that the environment keeps track of all the processes from previous initialized jobs, including those that have not been initialized yet in subsequent stages of a job. For the training phase, the violations of constraints are allowed as these experiences are used to enhance the agent’s decisions, but in practice, a reset of the last decision would be recommended. This heuristic consists of the omission of the initialization of the job that

is known to cause a conflict in the occupation of machines at some point in the process. Then, the agent continues its prediction in the following time interval. Note that this method cannot be applied when there is uncertainty in the processing times because the environment does not consider the information to predict a future violation. For cases involving uncertainty, a second heuristic, which consists of the cancellation of the job that causes a conflict in machines at any time in the path is recommended. Applying this heuristic ensures that no constraints are violated, including the zero-wait and the allocation constraints. By implementing this second heuristic on the final policy π^* , the corrected production for the instances where there was a constraint violation was 8.035% smaller than the actual reported production. Moreover, implementing these heuristics guarantees that the generated schedules from the agent are always feasible at the expense of reducing the expected production in the few cases where there was a constraint violation.

The two scenarios in the first case study allowed to demonstrate the capabilities of the proposed framework. For the two instances presented, the demands could be fully satisfied but for the second scenario, there is a tradeoff between constraint violation and demand satisfaction. To compare the POMDP used in this work to approach the problem under uncertainty, an MDP was set to train the RL agent; details of this implementation are provided in Section A.4 of the Appendix A. The policy was modeled with an FNN and trained using a DQN algorithm for the same number of episodes as the DRQN. Since the DQN works with less information in the input data than the DRQN, the memory required was less, and thus, the computational time was of about 1,624 seconds for one training using the DQN. Results showed that the maximum reward reached by the DQN was less than 0 reward units, meaning that the agent stayed idle during the entire horizon i.e., without initializing any product. On the other hand, the agent developed with the proposed framework obtained values over 200 reward units, thus illustrating a key advantage of the proposed approach. This was expected since DQN does not have explicit mechanisms for deciphering the underlying state in a partially observable environment and will only be effective if the states follow the Markov property [95].

As an additional experiment to demonstrate the performance of our framework under a reduced state (partial observability), case study 1 was performed under a random reduction of observations from even

time intervals. That is, parts of the observations selected randomly were not provided to the agent. Details of this instance are described in section A.5 in the Appendix A. Results from this instance showed the capacity of the proposed framework to provide acceptable scheduling decisions that can handle the inaccuracy or reduction of information in the data that is provided to the agent (e.g., noise, interruptions, and inaccuracies).

3.4. Case Study 2

This case study involves a batch plant under uncertainty. The number of stages is 10 and a total of 20 machines are used in the plant. Each stage contains two machines. The number of products is 10 and there are 15 jobs available to produce them (see Table A.3 in Appendix A). Products 1 to 5 can be produced with up to two jobs whereas products 6 to 10 can be produced through only one job. The time horizon is set to 150 time-intervals. The set of four matrices with the possible processing times that each production realization can take is presented in Eq. A.2 in the Appendix A. The initial demand is set to 80 m³ for each product, then the demand is updated three times at time-intervals given by $\Omega = \{30,60,90\}$. The vectors with the demands are described in Eq. A.3 in Appendix A. The hyperparameters were the same as in case study 1 for this agent, listed in Table A.2. The capacities of the 10 warehouses are the same and set to 100 m³.

A training of 10,000 episodes was set to train the agent; convergence was achieved by episode 5,000. The cumulative rewards per episode that show the evolution of the learning of the agent are shown in Fig. A.1 of the Supplementary Material. The evolution of constraint allocation violations during training in the first 5,000 episodes as a function of the episodes is shown on Fig. 21. The continuous reduction of violations is slowed down approximately at episode 4,000; at this point, the agent didn't reduce further the number of violations for the rest of the training.

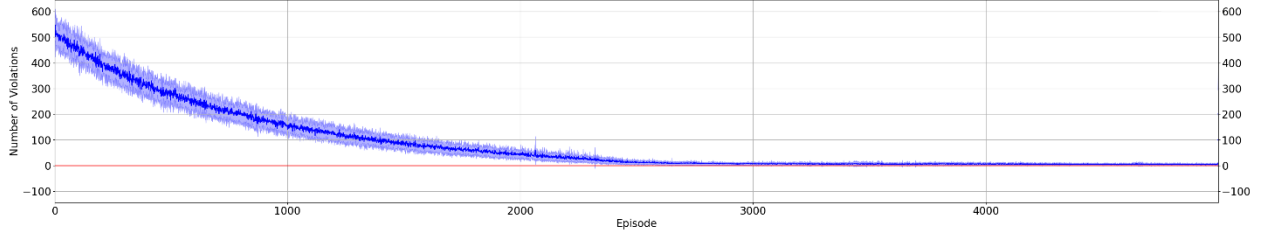


Figure 21 Average violation of allocation constraint at each episode over multiple trainings.

A test that consists of 10,000 instances on the resulting policy from this case study was performed. The results showed that the agent was able to complete the demand for all the instances. Nevertheless, it was found that 46.86% of the instances resulted in violations of the allocation constraints. Specifically, there were on average 1-2 machines in those instances that would receive two or more products at the same time during the process. Further analysis showed that these overlapping in the machines happened for 1-2 jobs during the entire scheduling horizon; note that a job j that is initialized implies the subsequent start of multiple machines included in their path K_j . Hence the initialization of a job could be responsible for multiple overlapping. Considering that the total number of decisions for this case is 150, two initializations of jobs represent 1.33% of the total number of decisions that the agent takes during the whole horizon.

The agent required 15,103 seconds to complete a training of 10,000 episodes and convergence was observed after 7,551 seconds at episode 5,000. After the agent was fully trained, the time it took to provide an action for a given $input_t$ was of $1.18e-3$ seconds on average. Fig. 22 shows a schedule extracted from the last stage of the training of the agent where there were few cases with constraint violations. In this schedule, there is a violation of constraints. For this instance, the agent completed the demand, and a surplus of 7.49% concerning the total demand was produced and allocated in the warehouses. The vectors for updating the demand were $\mathbf{d}_{t=30} = [100,90,90,80,100,100,90,80,80,80]$, $\mathbf{d}_{t=60} = [100,90,90,80,100,100,90,80,80,80]$, and $\mathbf{d}_{t=90} = [100,90,90,80,100,100,90,80,80,80]$. It was observed that the agent leaves some spaces between some products because it develops a *preventive policy* such that it avoids violation of the constraints. Fig. 22 also shows that the agent avoids the initialization of products close to the end to avoid the production of unfinished products.

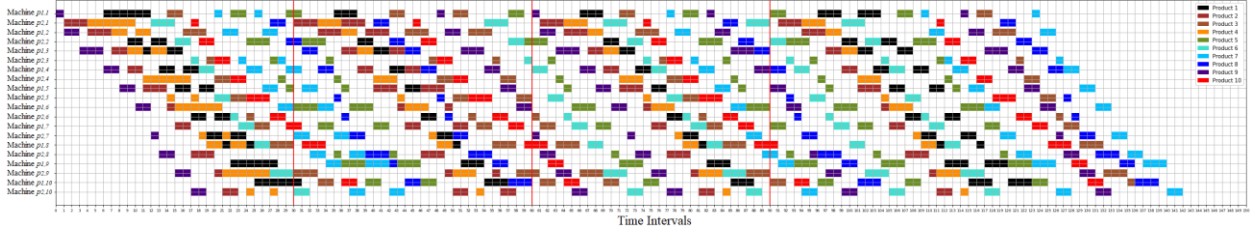


Figure 22 Schedule from the second case study.

To estimate the impact of jobs that were not allocated properly in the processes due to constraint violations, the same heuristic that was implemented in the previous case study was also considered for the present case study, i.e., the production from those jobs that caused an allocation violation was canceled. It was found that with the correction of these actions, the corrected production value would be 4.38% smaller than the one reported.

An additional case study featuring a flow-shop plant is presented in Appendix A in section A.3. The training also consisted of 10,000 episodes and convergence was achieved after 2,750 episodes. The results are similar in performance to the previous case studies in the sense that the agent could learn the processing times for the machines; nevertheless, several violations in the testing of the final policy were also detected. The details of this case study and the results are described in Appendix A.

Overall, the DRL training decreased the number of constraint violations and managed to accommodate the production to satisfy the demand that was subject to changes during the process. As mentioned above, the agent learned the processing times from each product without having access to them but through experience with the environment. For each case study, the agent struggled to learn how to deal with the uncertainty of specific products as it did not reach zero constraint violations but found a way to reduce the impact of the uncertainty in the process. Policies that could find optimal solutions became more difficult to reach as the complexity of the problem increased in terms of scale (i.e., size of the plants) and uncertain realizations. Real-world problems in scheduling have become a challenge in terms of reward definition as the final solution is usually suboptimal [60], [63]. Moreover, ANNs are not yet fully reliable to be in full charge of a process, this is related to the fact that the heuristics that they develop are encoded in the network

and cannot be directly interpreted. For this reason, supervision is often needed and human assistance is desired when the agent faces an extraordinary event [62].

For instances of the problem statement where the number of jobs is small and there is no uncertainty, the problem could be readily translated into a MILP and be solved online in a short time using standard MILP solvers, e.g., CPLEX, and GLPK. This is advantageous over the DRL method because the optimality in the solution can be guaranteed, and the response can be provided in real-time. Nevertheless, for larger problems (i.e., problems involving more jobs/machines) or for cases under parametric uncertainty, the complexity increases, i.e., the resulting formulations may become nonlinear thus requiring lengthy computational times.

The problems considered in this work are subject to exogenous and endogenous uncertainty and the realizations appear in the process in the form of demand updates (exogenous) and uncertainty in processing times (endogenous, type II [96]). The uncertainty in the problem implies the use of more sophisticated techniques such as mp-MILP, MILP sensitivity analysis, or robust optimization [97], [98]. These techniques have limitations such as challenging formulations, requiring a large number of samples, or providing conservative solutions [99], [100]. Accordingly, the problem becomes more demanding if we consider an online implementation where a reactive scheduling framework has to be adopted and return solutions within a specific time interval [1]. Consequently, it was considered that a fair comparison of the solution of an online scheduling problem between the proposed RL framework and a MILP method would be difficult to establish. Also, if we compare the nature of the solutions, we can find a few differences: The solution of the DRL method is a trained ANN that generalizes the process dynamics and can output the decision-making process for multiple situations while a MILP searches for an optimal solution considering the level of uncertainty in the problem. High levels of uncertainty in the parameters might lead to considerable computational times needed to find a solution. To make a comparison between the response of a MILP and a DRL agent, there is the need to make assumptions in the problem, for instance, a deterministic formulation [9]. Since this work focuses on demonstrating the adaptability of the DRL agent to uncertainty and the generation of feasible schedules online, it was considered that other comparisons might be useful to

demonstrate the performance of our framework. In our work, to show that the method converges to a unique solution satisfying the constraints and demands, we solved the problem without uncertainty and showed that it did not fail to violate constraints or meet the demand.

Once the agent is trained, the states are evaluated in the ANN and a decision is provided. The time taken for this evaluation is quite fast, e.g., in the order of milliseconds for the cases presented in this work. We expect that solving the corresponding optimization problem under similar considerations may take from seconds to minutes. Note that these decisions are made at every time interval considering the uncertainty in the process and the objectives of the agent (minimize the makespan, meet demands, and avoid constraint violation). The computational effort that the agent needs to produce a decision is independent of the size of the plant, i.e., the ANN is a black-box model trained to provide the best actions at a given time. Although it is well known that DRL does not guarantee an optimal solution [101], the advantage of computational time, responsiveness to uncertainty, and online implementation constitute the main advantages of the proposed DRL method. While the optimal value is not guaranteed on DRL, it can be demonstrated that the method aims for the optimal policy [102]. Reaching the optimal value depends on fine-tuning the hyperparameters and the computational budget available to train the agent.

Experiments with uncertain parameter values that did not occur during the training were performed. Although the capacity of ANNs for generalizing on the data is well known, the processing times that were proposed do not correlate with one another and were defined randomly. Hence, the model failed in almost all of the test instances to avoid constraint violations, and the demand was only met in less than 35% of the tests. If the uncertain processing times were correlated and some dependency existed between them, the model would have been capable of assuming this correlation and then, be able to adapt to demands/machining times realizations that were not included in the training set, i.e., part of the learning process by the agent would have been focused on learning these correlations. Our problem was designed to correlate machine occupation and product demands with the initialization times of the jobs on the horizon. Thus, the agent was expected to learn how to produce schedules for different configurations of processing times and demand realizations during the horizon. At the end of the experiments, it was observed that the

agent developed a preventive (conservative) behavior on the violation of constraints by making decisions that considered multiple possible realizations in the processing times. The trained policies “learned” almost all the processing times with their respective deviations and were able to produce a convenient decision to also meet the uncertain demands on time. Consequently, the ineffectiveness of the agent in handling different parameters from those used in the training set is justified since the purpose of the agent was (among other aspects) to indirectly learn the processing times and demands in the plant through an observation that provides machine occupation and production.

3.5. Summary

In this chapter, a methodology to produce an agent trained with a DRQN method to produce an online schedule for job-shops and flow-shops subject to uncertainty was presented. Uncertainty realizations in the processing times of specific machines from specific jobs are considered. Also, the demand is updated recurrently at explicit times during the process. The agent trains in an environment where the uncertainties in the processing times and demands are chosen randomly, and it needs to accommodate the products subject to specific constraints. Moreover, the agent is oriented to learn how to reduce the overall makespan of the process given a specific time horizon. Results showed that the agent was able to learn a policy that could meet the demand for both case studies. Nevertheless, the agent fails a small percentage of the decision-making process (less than 2%) which leads to constraint violations. The impact of these incorrect decisions can be addressed by reshaping the reward with a direct impact on demand satisfaction.

The RNN was able to register the difference of each realization on the products with uncertainty, hence the agent developed a way to perceive the difference for each realization in the process. There was a prevalence of constraint violations in the final policies, presumably due to a need of more information in the *input*_{*t*} about the uncertainty (e.g., probabilities or causes that can help to estimate it). Nevertheless, the agent was capable to manage different realizations of uncertainty in the scheduling process while meeting the demands. An alternative to this lack of information consists in the use of more sophisticated mechanisms to detect these features in the state, for instance, attention modules or autoencoders, which are

commonly used in sequential processes such as language recognition. This aspect will be explored in Chapter 6.

Chapter 4 A Recurrent Reinforcement Learning Strategy with a Parameterized Agent for Online Scheduling of a State Task Network Under Uncertainty

In the works described in Section 2.5.1, the DRL agent makes one discrete decision at every time-interval that specifies the job/task that should be initialized, and which belongs to a set of possible actions. This approach is limited to scheduling problems that require parameterized actions where not only a discrete action is needed but also another level of description for that particular action, e.g., machine specifications of a task such as unit capacity, processing times, or the initial capacity load of a batch. This limitation was presented as a gap in the literature (Section 2.6, item b). Methods to approach this type of instances are available, e.g., multi-agents [103] and hierarchical RL [104]. Those studies mainly partition the problem into subproblems and approach them with individual agents. Those approaches are computationally expensive as they require multiple trainings for the multiple agents or decision levels to obtain a trained system of policies to perform. Motivated by this, and with the aim of reducing the computational burden required for integrating multiple (hybrid) decisions in one agent, the use of a hybrid agent that can solve multiple hybrid (i.e., discrete or continuous) decisions at every time-interval in the scheduling horizon is presented in this chapter.

The contribution from the previous chapter approaches the scheduling problem as a POMDP for job and flow-shops under uncertainty using Recurrent cells [16]. The framework presented in this chapter expands upon that work and improves this feature to scheduling processes on STNs, which are widely used for short-term scheduling in chemical facilities. Moreover, the use of a hybrid agent allows multiple decisions to be made at every time interval; these aspects were not considered in our previous work.

This work is structured as follows: Section 4.1 presents the problem statement. Section 4.2 presents the methodology for solving the problem statement with a hybrid agent. In section 4.3, two case studies are presented and used to test the performance of the proposed RL framework. Conclusions and future research directions are described in section 4.4.

4.1. Problem Statement

This section describes the details of the scheduling processes under uncertainty that can be addressed with the method proposed in this work. First the concept of STN is described, followed by the optimization problem. Then, the limitations of current methods to approach these problems are highlighted. The relevant features from our framework that will be useful to handle these limitations are shown at the end.

As shown in Fig. 23, an STN is a visual representation of a multi-product batch process optimization problem that was proposed by Kondili et al. and uses four essential entities for representing a process: tasks, states, utilities and units [48]. Tasks are unit operations that transform materials into intermediate or final products, e.g., reactions or separations; they are represented with rectangles. States are storage tanks where the raw materials, intermediates, and final products are allocated, represented with circles. The STN can account for the use of utility services as cold or hot streams. Each task in the STN has one or more unit (or machine) to perform the task and it is possible that a task shares its units with other tasks.

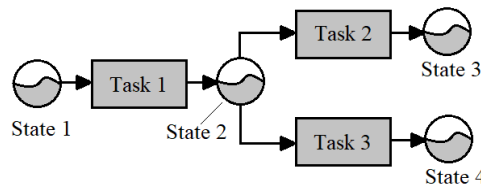


Figure 23 State Task Network representation.

It is assumed that the STN is subject to time dependent uncertainty realizations that could be aleatoric or epistemic. The realizations can affect the development of a task since they may affect processing times, material outputs, capacities, etc. Moreover, it is assumed that communication devices like sensors used for gathering information of the plant may be subject to malfunction and therefore provide inaccurate data for

the decision-making process. The basis of the systems that can be addressed with the framework presented in this work are as follows:

- A multi-product batch plant that is composed with N_i tasks, with N_j number of units available, where each task can be executed with at least one unit included in N_j .
- A set of N_s states for storing products in the plant which can receive (produce) material from (of) a task in N_i .
- A recipe that describes the process(es) that takes place in the multi-product batch plant.
- A set Ψ that contains the parameters ψ of the process, which are used to formulate material balances, residence times, translation of products from tasks to states and vice versa, etc. These parameters include capacities, input and output proportions of states from tasks, processing times, etc.
- A characteristic function for the aleatoric $\theta(t)$ and epistemic $\omega(t)$ sources of uncertainty that affect a subset of the parameters in Ψ .
- A set F_t^d with n^d discrete decision variables, and a set F_t^c with n^c continuous decision variables. These set of decisions can be taken at every time-interval t during the scheduling horizon.
- An objective function G to be optimized towards the maximization of economic or production objectives.
- A finite time horizon H which starts at t_s and ends at t_f . H is divided in time-intervals of equal length, t .

The aim of this work is to develop a hybrid agent (i.e., a policy) that builds a schedule in an online fashion while optimizing the objective function G and complying with the process and operational constraints. Moreover, the agent should consider the realizations of uncertain parameters in the environment. The scheduling optimization problem that considers the features described above can be conceptually formulated as follows:

$$\begin{aligned}
 & \max_{F_t^d, F_t^c} G(F_t^d, F_t^c, \Psi, \theta(t), \omega(t), t) & \text{(P2)} \\
 \text{s.t.} \quad & h(F_t^d, F_t^c, \theta(t), \omega(t), t) = 0, \quad \forall t \\
 & g(F_t^d, F_t^c, \theta(t), \omega(t), t) \leq 0, \quad \forall t \\
 & F_t^d \subseteq \mathbb{R}^{n^d} \\
 & F_t^c \subseteq \mathbb{R}^{n^c} \\
 & \Psi \subseteq \mathbb{R}^{N_\psi}
 \end{aligned}$$

$$\begin{aligned}\theta &\subseteq \mathbb{R}^{N_\theta} \\ \omega &\subseteq \mathbb{R}^{N_\omega} \\ t &\in [t_s, t_f], H = t_f - t_s\end{aligned}$$

where h and g represent the model equality and inequality constraints, respectively. The uncertain parameters considered in P2 require the need to use either robust or stochastic optimization methods. The former method considers the worst-case scenario that can take place in the scheduling process; thus, the schedule resulting from these approaches tend to be conservative [50]. On the other hand, stochastic methods often consider probability distributions to build schedules and make use of methods such as two-stage optimization or multi-stage optimization [105], [106]. These techniques adjust the schedule as the uncertainty reveals during the process, which can result in long turnaround times if the problem is considerably large.

The DRL framework proposed in this work aims to provide an alternative to solve online scheduling problems under uncertainty for multi-product batch plants. The key idea is to train a policy that provides fast (online) solutions during operation. The decision-making process of the agent considers the evolution of events in the plant, and thus, the agent is aware of the realization of uncertainties and its effects over time. Additionally, with the hybrid agent, the DRL can be expanded to complex environments in the chemical engineering field that require compound actions, i.e., more than one decision at each time-interval. This allows to use a unique agent rather than, for instance, a multi-agent system or a hierarchical RL method, which for some instances may be more expensive to train. Overall, the implementation of hybrid DRL agents to scheduling problems enables fast online reaction to uncertainty realizations while taking into consideration the integration of multiple decisions in the operations.

4.2. Methodology

In this section, the methodology to train a policy that can be used for online scheduling in the multi-product batch plant described in the previous section is presented. This section is divided in two parts: The first part describes the DRL hybrid agent and the algorithm used, namely the PPO method. Also, the modifications to this method to handle a hybrid agent and the architecture of the ANN for handling a

POMDP are described. The second part describes the translation of the problem statement into an environment that can be used for training. The techniques used for exploring the environment and restricting infeasible actions to accelerate the learning of the agent are described at the end.

4.2.1 Deep Reinforcement Learning Hybrid Agent

PPO is an on-policy algorithm that has shown good performance for training DRL agents in discrete and continuous action spaces [7], [38], [107]. It is a well-known Actor-Critic algorithm that trains a policy (actor) and a value function approximation (critic). The key advantages of this method include the use of trust region strategies for optimization of the ANN parameters, and the multiple updates with minibatches performed for sample efficiency [38]. Other advantages of this method include robustness for tuning hyperparameters and the simplicity of implementation compared to similar methods like Trust Region Policy Optimization [37].

The architecture of the hybrid agent considered in this work was adapted from Fan et al. [108]. A key feature in the present framework is the addition of the correlational module to handle the POMDP. A schematic representation of the resulting hybrid agent is shown in Fig. 24a. A sequence of length Γ of consecutive observation vectors \mathbf{o}_t at time interval t from the environment is referred to as the observation window \mathbf{O}_t . Note that, in contrast to the work from Chapter 3, \mathbf{o}_t is a vector (not an independent value) that contains the information from the environment at t . While \mathbf{O}_t remains as the input vector of the agent that holds Γ observation vectors \mathbf{o}_t . This sequence is fed into the agent and processed through a set with α layers of LSTMs. Then, a set with K linear layers is used to integrate the Γ outputs from the correlational module. The output from the linear layers is an input into n individual ANNs with Z linear layers where n represents the total number of discrete and continuous variables, i.e., $n = n^d + n^c$. Each ANN is used to output one of the elements in the set F_t , which considers both F_t^d and F_t^c .

For discrete actions, the output of a decision is passed through a SoftMax function to produce a categorical distribution of the actions for that decision. For continuous actions, the output becomes the value for the corresponding action. The architecture of the critic network is shown in Fig. 24b; the inputs

to this network are the same observation window with Γ observations that is provided to the actor, then a set with α LSTM layers is used to correlate these observations. A set with K linear layers is used to output the critic value.

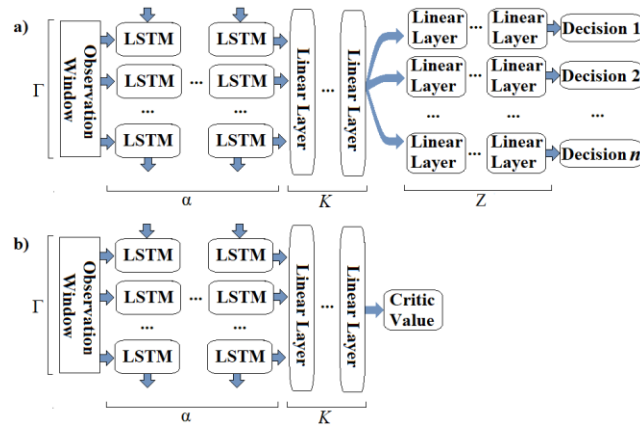


Figure 24 a) Architecture of the agent and b) architecture of the critic.

Applications with multiple levels of decisions can be handled with this type of agent. These decisions correspond to those that require the specification of a parameter of the main decision. For instance, assume that a first decision in a scheduling process corresponds to the task that needs to be started, and for that selected task, a second decision corresponds to the selection of the machine for that task. As an example, suppose there are 3 tasks and each task has a set of 2 machines available, then the first discrete decision has 3 possible actions (i.e., the 3 tasks) whereas the second discrete decision has 2 possible actions (i.e., the machine to use for the task). Since each task has its own pair of machines, the second decision is different for each task; thus, three independent decisions are then considered (one for each task). Fig. 25a shows a diagram of the architecture of this hybrid agent. As shown in this figure, the agent will have to consider four discrete decisions ($n = 1 + 3$). Note that when a task is not chosen, the second decision for that task is not used in the environment either. To build an agent with continuous actions, it can be assumed that those actions are correlated between each other and can be output from the same set of linear layers instead of adding one independent set per action. This last option is feasible but increases the computational effort of the training. Fig. 25b shows two linear layers used for four decisions: the first layer outputs a categorical distribution that selects a task (first discrete decision) while the second set of layers outputs three continuous

values (second, third and fourth continuous decisions) that correspond to a particular feature of each task, for instance, batch size, task timing, utility service allocation, etc.

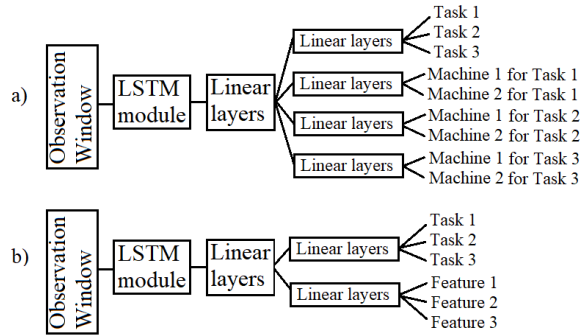


Figure 25. a) Hybrid agent with multiple discrete actions. b) Hybrid agent with discrete and continuous actions.

The mathematical definition of the set of scheduling decisions (F_t) is shown in Eq. 32. As mentioned above, this set includes n elements where each decision d has an action space that can be discrete or continuous.

$$F_t = \{d_1, d_2, \dots, d_n\} \quad (32)$$

The main advantage of the hybrid agent is that it allows the decomposition of the decision space of the scheduling process. This decision space may consist of several decisions that are required to be executed in the process (i.e., in parallel) or for a specific task (i.e., hierarchically). The hybrid agent decomposes the compound decision into simpler action spaces, thus generating multiple policies in one that uses the same input data (i.e., \mathbf{O}_t). Moreover, this method can be used for hierarchical decisions or parallel decisions in the process. A hierarchical decision is structured in multiple levels where high-levels define broad strategies while lower-levels refine or specify the way these strategies are implemented. Parallel decisions refer to separate independent decisions that take place in the same process. Naturally, increasing the complexity of the scheduling process, or the number of decisions, increases the computational effort required to train the hybrid agent.

4.2.2 Environment

Details on the translation of the scheduling optimization problem (STN) into a DRL environment are described in this section. The environment is a simulation of the facility where the scheduling is performed. It is through the environment that the agent trains and develops policies that aligns with the goals outlined with the scheduling problem depicted in P2. The time horizon H is discretized into time-intervals t at which the environment sends an observation window \mathbf{O}_t to the agent. Then, for each \mathbf{O}_t at every time-interval, a corresponding set of decisions F_t is returned to the environment to be executed. The performance of the actions is evaluated through a reward function $r_t(\mathbf{o}_t, a_t)$ which is a scalar value that describes how much a decision is aligned with the objectives and constraints outlined included in P2. The rewards are used to update the parameters of the hybrid agent towards the maximization of the objective function and meeting the problem constraints. After executing the set of decisions, the environment evolves and a new observation (\mathbf{o}_{t+1}) is added to the observation window, while the oldest observation is dismissed. This is to keep a constant length in the vector \mathbf{O}_t corresponding to a specific number of past observations.

Since the agent needs to learn from experiences that include infeasible scenarios, the environment must be set up to handle actions that violate constraints. Infeasible actions might be handled in different ways; for scheduling processes, it was observed from experimental tests that cancelling the execution of these actions in the environment and providing a penalty to the agent was the strategy that affects the least the learning process. Another option consists of terminating the episode. While this would give a clear and strong signal to the agent about violations, it has been found that this method reduces the generated data to train the agent and also leads to instability in the training due to the variation in the length of the episodes [109].

4.2.2.1 Uncertainty

The policy proposed in this work can provide online scheduling decisions subject to uncertain realizations in the scheduling process. A policy makes decisions in light of the current conditions of the

environment, in accordance with the objective function shown in P2. The use of ANNs to model the policy leverages the potential of this framework by increasing the capacity to generalize to scenarios that are new to the agent [76]. For these reasons, the use of DRL methods is attractive for handling realizations from aleatoric and epistemic sources of uncertainty.

During the training, the agent needs to be exposed to multiple realizations of the uncertainties. These can be incorporated into the environment through Probability Density Functions (PDF), e.g., Gaussian or Uniform distribution are popular choices. This process is done by defining the realization of the uncertain parameters in the environment at each episode using these distributions. Although the DRL framework cannot guarantee that an optimal policy is achieved at the end of the training, the fact that it aims for the optimal policy and progressively performs better in the environment has been shown empirically for many RL algorithms, including PPO [38], [107], [110].

Aleatoric and epistemic uncertainties are handled by the DRL agent in different ways [111]. Aleatoric uncertainty represents the inherent stochasticity of the environment, which cannot be reduced by exploration. For instance, stochastic demands that are difficult to predict due to external factors, e.g., continuous changes in customers' demands; thus, they cannot be confined to a set of scenarios from which the agent can learn. In these cases, the DRL method develops a conservative policy that can provide a schedule that maximizes the reward (objective function) while considering worst case realizations of the uncertainty. On the other hand, epistemic uncertainty is related to the lack of information that is needed to provide accurate prediction, for instance, the processing time of a reaction, which could be predicted through a rigorous observation of the kinetics of the reaction. This type of uncertainty is potentially more promising to be learnt by the agent as it depends on the exploration of features that indirectly describe the uncertainty in the parameters. In this work, it is assumed that the agent accounts for enough information in the observation window to build this understanding of the epistemic uncertainty, and that sufficient exploration and training are provided.

POMDP provides benefits for handling scheduling processes under both types of uncertainty. Recall that a sequence of events into the past (i.e., the observation window \mathbf{O}_t) is correlated with a set of LSTM

layers and then used to produce the decision set F_t . The observation window contains the parameters affected by the uncertainty that can be used to gain insights. For the epistemic uncertainty in the environment, the previous history provided through \mathbf{O}_t is expected to deliver insights to better understand the transition dynamics that resulted in the uncertainty realization. For the aleatoric uncertainty, the enlargement of contextual information allows the agent to learn about the consequences of a stochastic realization in the scheduling process, allowing the agent to learn how to handle the effects produced by such realization.

4.2.2.2 Observation Window

Each observation \mathbf{o}_t in the observation window \mathbf{O}_t contains information from the environment at a given time t that is relevant for making the set of decisions F_t . For a multi-process batch plant, this information is related to the tasks, batches, states, utility services, machines, processing times, demands, cleaning periods, and uncertainty realizations. To represent each feature in \mathbf{o}_t , one or more numerical elements can be used. To maintain a stable and smooth training performance, each value in the observation should be normalized in a value in the range of $[0,1]$ or $[-1,1]$. There are different techniques in ML that can serve for this purpose: a) min-max scaling, b) sigmoid or tanh transformations, c) one-hot encoding, and d) clipping values [26].

As shown in Eq. 33, the observation window \mathbf{O}_t is a sequence of observations with length Γ that provides the recent history of events that took place in the environment. This method is fundamentally the same used in Section 3.2.2 from the previous chapter. The hyperparameter Γ is user-defined and is dependent on the number of time intervals in which an event that contains relevant information is expressed. For instance, if the user wants the agent to observe the development of a task in the process, then Γ should be large enough to contain all the time-intervals that a task takes to be completed. The agent will then be aware of the evolution of that task. Another alternative is that the length Γ can be set based on other events that occur periodically during the process, e.g., updates in demands and market prices, or seasonal changes

that affect the scheduling. Under this scheme, the development of information during the process can be used to make more informative decisions.

$$\mathbf{O}_t = [\mathbf{o}_{t-\Gamma+1}, \dots, \mathbf{o}_{t-2}, \mathbf{o}_{t-1}, \mathbf{o}_t] \quad (33)$$

Note that our approach assumes partial observability of the environment, i.e., missing information from the environment produces an incomplete observation for the agent to make a decision. The use of the POMDP is justified by the need to increase the contextual information that is sent to the agent for handling uncertainties in the scheduling process. Additionally, in chemical processes where information is generated asynchronously, i.e., not every feature of the environment provides information at every time-interval t , the assumption of a Markovian environment may not be sufficient to develop an adequate policy since it does not account for enough information to make effective decisions.

4.2.2.3 Rewards

PPO is an on-policy algorithm that updates the policy using *fresh* data produced in the environment at the time of updating the parameters from the network. Thus, PPO progressively improves its policy as the agent moves towards a desired behavior, which gives its characteristic stability in learning. Consequently, a dense system of rewards that can provide an accurate metric on how far an action is from the desired behavior is required to align with the characteristics of PPO. A reward shaping strategy similar to the one presented in Section 3.2.3 and Eq. 28, is proposed here to provide guidance to the agent on the discrete and continuous actions. This method adds the independent sub rewards $q \in Q$ from the discrete and continuous actions at each time interval t . The formulation of the reward at time-interval t is as follows:

$$rew_t = \sum_{q \in Q} r_{qt} \quad (34)$$

The characteristics of the rewards for discrete and continuous decisions are different because their effects on the environment are measured considering different metrics. Rewards for the discrete actions are defined depending on the relevance of the decision in the scheduling process. The system of rewards should provide guidance to the agent in the environment on performing actions that align with the objective

function. For instance, the agent needs to learn the recipe(s) in the scheduling process, then, larger rewards should be provided when tasks that are closer to the end in the recipe are initialized. This prevents the agent from getting stuck in initial tasks and incentivizes the agent to explore beyond this point in the process. Penalties are delivered if constraints are violated, e.g., allocation, capacity, and time constraints. Additionally, and as it was discussed above, infeasible actions should be cancelled and allow the scheduling process to continue for a particular episode in the training.

Continuous actions should be rewarded considering the optimal value that can be held by those decisions in the current conditions of the environment. For instance, assume that decision d_1 can take any value between 0 and 1, now suppose that at a given time in the scheduling process this decision can take a maximum value of 0.65, then, if the agent provides this value, the maximum reward should be delivered. The reward should decrease in the proportion that the decision moves away from this maximum value. If the agent exceeds the maximum value, and that action becomes infeasible, then that action is cancelled, and a penalty is delivered.

A limitation of the reward scheme for continuous actions is that the reward increases linearly until the maximum value, and then it drops to a penalty, which is usually a value less than or equal to zero (see Fig. 26a). Ideally a bell-shaped curve of rewards is desired, where the agent moves around a certain value until it finds the best (see Fig. 26b). This bell-shaped reward scheme cannot be used if the decision becomes infeasible because in that case a penalty needs to be delivered. Preliminary tests found that the agent often leaves a gap between the maximum possible value and the actual value to avoid the risk of falling into a penalty.

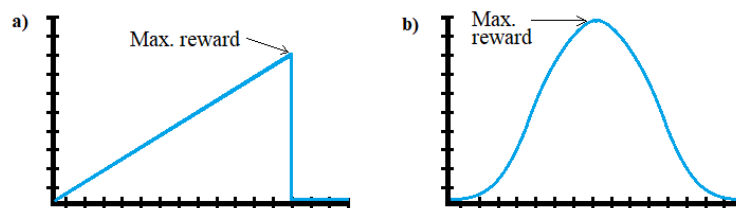


Figure 26: a) The reward model considered in this work. b) A bell-shaped reward with the maximum reward in the middle.

4.2.2.4 Exploration methods

An annealing technique is used to systematically reduce the value of the hyperparameters that influence the exploration in discrete and continuous decisions and the learning rate. For the decision-making process, this technique refines the selection of actions and makes the hybrid agent greedier as the training progresses, then the exploration space is gradually reduced. The function for the annealing technique is shown in Eq. 35. The variable E_t represents the hyperparameter at a given time interval t during the training time horizon T . The values of $E_{t=0}$ and $E_{t=T}$ are user-defined for each variable, typical values for these parameters are described below for specific exploration methods.

$$E_t = E_{t=0} + \left(\frac{E_{t=T}}{E_{t=0}} \right)^{\frac{t}{T}} \quad (35)$$

For discrete actions, a hyperparameter referred to as temperature is used [112]. This hyperparameter scales the output from the Z hidden layer (see Fig. 24a) of each discrete decision before applying the SoftMax activation function. That is, before the categorical distribution is output from the agent, the output values are scaled to modify the resulting distribution. The value of the temperature hyperparameter is equal to 1 at the beginning of the training, which does not modify the distribution. This value is gradually decreased as the training advances, causing the distribution to sharpen and producing larger differences among the probabilities, and ultimately making the agent more decisive when sampling an action. Preliminary testing is needed to adjust this hyperparameter, our simulations showed that a default final value of 0.1 for the temperature is suitable.

In the case of continuous actions, the outputs from the continuous part of the hybrid agent are integrated into a covariance matrix as mean values. This covariance matrix is used to sample actions around the mean value that was output from the agent. The variance of the matrix is gradually reduced during the training using Eq. 35 such that the agent takes closer samples to the mean value. This allows the agent to widely explore the continuous actions at the beginning of the training and progressively make refined adjustments. It was observed that default values of the variance hyperparameters are 0.2 at the beginning and $1e-5$ at the

end are suitable. Furthermore, the learning rate is also reduced with the formulation described in Eq. 35 such that changes to the parameters of the hybrid agent are refined at the end of the process. Default values for the learning rate can span between $1e-4$ and $1e-6$ between the beginning and end of the training, respectively.

4.2.2.5 Masking Discrete Actions

The multi-product batch plant scheduling problem under uncertainty shown in P2 can be considered a complex environment. To reduce the complexity of the scheduling problem, a few constraints can be set as masks for actions. This prevents the consideration of specific discrete actions at certain states because they are infeasible or unsuitable at specific states. Masking an action consists in reducing its probabilities to zero in the categorical distribution that is output from a discrete agent. The masking vector is a Boolean vector with the length of the discrete action space. The actions to be masked are set to False and those that should not, are set to True. This vector is used to set the corresponding logits l_n (outputs from d_n) into very small values M , before the SoftMax activation function is applied. Then, the SoftMax function will convert the masked logits into a probability close to 0%. To exemplify this process, Eq. 36 shows an application of the masking method based on the work from Huang and Ontañón [113]. The discrete action space has four actions where the third action should be masked. Assume that logits l_0, l_1 and l_3 have the same value so the final probabilities are equal.

$$\begin{aligned}
 \text{mask vector} &= [\text{True}, \text{True}, \text{False}, \text{True}] \\
 \pi(\cdot | \mathbf{O}_t) &= \text{SoftMax}(l_0, l_1, M, l_3) \\
 \pi(\cdot | \mathbf{O}_t) &= [0.33, 0.33, 0.00, 0.33]
 \end{aligned} \tag{36}$$

The masked actions are known *a priori* to be infeasible or inadequate at some period of time in the environment. Masking infeasible actions reduces the search space and accelerates the learning process. The masking technique can serve to divide the scheduling process in campaigns. This consists of partitioning the time horizon H into periods where specific tasks are performed, where the output products of the campaign will be inputs into the next stage/campaign. At each stage, the decisions that are not allowed are

masked such that the agent does not consider them in that part of the process. With this method, the learning process becomes more effective since the space where the agent explores is reduced.

4.3. Results

The methodology presented in the previous section has been implemented on two case studies adapted from the literature. The first case is used to validate the optimization capacities of the DRL hybrid agent by comparing its performance with the optimal solution generated with a model-based optimization formulation. Then, three different instances of this case study are formulated with either epistemic or aleatoric uncertainty realizations. The second case study, which is a larger plant than that considered in the first case study, is used to evaluate the method with a different environment and to demonstrate the influence of the masking function in the training. The hybrid agent developed with this framework (referred hereinafter to as *POMDP agent*) is compared with a hybrid agent without recurrence (referred hereinafter to as the *MDP agent*). The MDP agent only has access to the current time interval contrary to the POMDP agent that considers an observation window. To adapt the POMDP to the MDP, the section with α LSTMs in Fig. 24a is replaced with a set of three linear layers that only inputs the current observation \mathbf{o}_t .

The method was developed with PyTorch version 2.1.0 and Python 3.11.3, other libraries include NumPy 1.24.3 and Gym 0.26.2. The training performed for the case studies was implemented on a processor Intel Core i5-7500 CPU at 3.40 GHz. The average time of the training with 15,000 episodes of the POMDP agent required on average 15.1 h while the MDP agent needed on average 6.47 h for training the same number of episodes. The list of hyperparameters used for all the trainings is shown in the Appendix B (Table B.1).

4.3.1 Case Study 1

This case study corresponds to the STN depicted in Fig. 27, which has been adapted from the literature [48]. The objective is to maximize the production of products 1 and 2, which are stored in states E and I, respectively. There are 5 tasks and 9 states in total in this network and 4 units/machines available to perform the tasks. A time horizon H of 31 time-intervals is set for this problem.

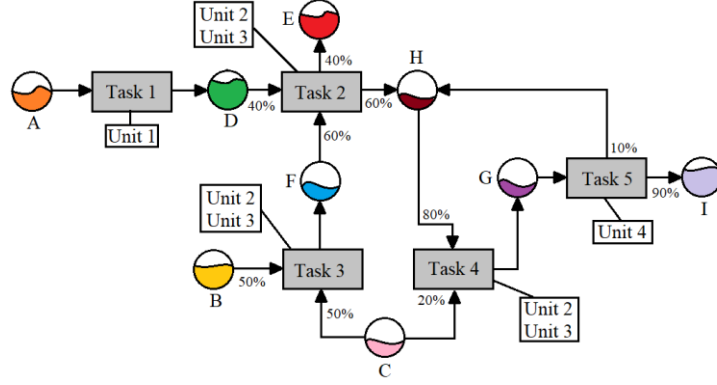


Figure 27 State Task Network.

Table B.2 in Appendix B summarizes the relations between tasks and units and the processing time (expressed in intervals t) from each task. The relation between tasks and states is shown in Table B3 (Appendix B), i.e., the states from which each task collects and drops off product. States A, B, and C are fed states assumed to have unlimited product availability. The rest of the states have a capacity of 300 units and are assumed to be empty at the beginning of the process. Table B.3 (Appendix B) also describes the proportions in the network from input states to tasks and from tasks to output states. The objective function of the scheduling problem aims to maximize the final products at States E and I at the end of the horizon H . Eq. 37 shows the objective function where S' represents the subset of states that contain the final products of the process (i.e., E and I).

$$G = \sum_{s \in S'} S_{s,H} \quad (37)$$

The corresponding scheduling optimization problem for case study 1 is shown in problem (P3). The description used was based on the mathematical formulation suggested by Kondili et al. [48] with the omission of constraints related to interruptions, pre-emptive operations, etc. The objective function (P3.a) maximizes the production in the horizon H . The decision variables in this problem are: 1) the discrete binary variable W_{ijt} that indicates the initialization of task i in unit j and time t , and 2) the continuous variable that indicates the capacity B_{ijt} at which such tasks should be initialized.

$$\max_{W_{ijt}, B_{ijt}} G \quad (\text{P3.a})$$

$$s. t. \sum_{i \in I_j} \sum_{t'=t}^{t-p_i+1} W_{ijt'} \leq 1 \quad \forall j, t, i \in I_j \quad (\text{P3.b})$$

$$W_{ijt} V_{ij}^{min} \leq B_{ijt} \leq W_{ijt} V_{ij}^{max} \quad \forall i, t, j \in K_i \quad (\text{P3.c})$$

$$0 \leq S_{st} \leq C_s \quad \forall s, t \quad (\text{P3.d})$$

$$S_{st} = S_{st-1} + \sum_{i \in \bar{T}_s} \bar{\rho}_{i,s} \sum_{j \in K_i} B_{i,j,t-p_{i,s}} - \sum_{i \in T_s} \rho_{i,s} \sum_{j \in K_i} B_{i,j,t} \quad \forall s, t \quad (\text{P3.e})$$

$$S_{st} = S_{s,0}^{init} \quad \forall s \quad (\text{P3.f})$$

where I_j represent the set of tasks which can be performed by unit j ; p_i represents the processing time for task i . V_{ij}^{min} and V_{ij}^{max} are the minimum and the maximum capacity of unit j when used to perform task i . S_{st} is the amount of material stored in state s , at the beginning of t . C_s represents the maximum storage capacity dedicated to state s . T_s and \bar{T}_s are the set of tasks receiving and producing material from state s , respectively. $\rho_{i,s}$ and $\bar{\rho}_{i,s}$ correspond to the proportion of input of task i from state $s \in S_i$, and the proportion of output of task i to state $s \in \bar{S}_i$. K_i is the set of units capable of performing task i .

Eq. P3.b represents an allocation constraint whereas Eq. P3.c bounds the size of the batches that are being processed in the units. Eq. P3.d sets the maximum capacity for the state nodes. Eq. P3.e defines the material balance for every state; showing that a given state S_{st} will contain the amount of product from the previous time step (S_{st-1}) plus the aggregated production at that time minus the product sent to the tasks. The aggregated production is expressed as the addition of all tasks that enter the state, expressed as the product of the proportion of the task ($\rho_{i,s}$) times the capacity of the task ($B_{i,j,t}$). Similarly, the product sent to the subsequent tasks is expressed as the product of the proportion ($\bar{\rho}_{i,s}$) of each output times the capacity of each task. Eq. P3.f defines the initial amounts for each state.

The set of discrete and continuous decisions to be used by the DRL agent (F_t) is described in Eq. 38, the action space of each of these decisions is defined in Eq. 39 and Eq. 40, respectively. The discrete action space consists of the combination of tasks i with units j making a total number of eight actions. Additionally, an action where the agent does not start any task in the environment at a given time interval is added (i.e., idle action). The continuous set of decisions (c_i) is composed by eight values which

correspond to the unit's load from the task selected in the discrete action. For instance, if *task 1 at unit 1* is selected as a discrete action and c_1 (the corresponding continuous value to that action) is equal to 0.78, then the compound action consists of *activating unit 1 at 78% capacity to execute task 1*.

$$F_t = \{action_d, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\} \quad (38)$$

$$action_d = \{task\ 1\ at\ unit\ 1, task\ 2\ at\ unit\ 2, task\ 2\ at\ unit\ 3, \\ task\ 3\ at\ unit\ 2, task\ 3\ at\ unit\ 3, task\ 4\ at\ unit\ 2, \\ task\ 4\ at\ unit\ 3, task\ 5\ at\ unit\ 4, idle\ state\} \quad (39)$$

$$c_i = [0,1] \quad i \in \{1,2,3,4,5,6,7,8\} \quad (40)$$

The information in the observation \mathbf{o}_t indicates to the agent the current capacities of the states and the units in the plant, the amounts of products 1 and 2 that have been produced at the current time-interval, and the time that each unit has been operating (if they are being used). This information is normalized into values between 0 and 1 for effective training (see section 4.4.2). The observation window \mathbf{O}_t is set to contain four observations (i.e., $\Gamma = 4$) considering the length of the tasks which is equal or less than four time-intervals. The architecture of the hybrid agent consists of three layers of LSTMs (i.e., $\alpha = 3$); followed by three linear layers ($K = 3$). Then, the output from this module becomes the input for two ANNs with three hidden layers each (i.e., $Z = 3$). The values of α, K , and Z were determined after experimenting with different structures aimed at finding an adequate network that can represent the policy. The first ANN outputs the categorical distribution to select a discrete action, while the second ANN outputs eight values which correspond to the continuous decisions c_i .

Following the reward shaping function shown in Eq. 34, the total reward at each time-interval is the addition of multiple sub rewards that the agent achieves for specific accomplishments, including meeting the constraints and maximizing production. The agent receives a reward r_q for the initialization of every task through the discrete decision, larger rewards are granted for tasks that are near the end of the process (i.e., Tasks 4 and 5). If the allocation constraint Eq. P3.b is violated, then the agent receives a penalty (negative reward). The continuous action is evaluated for capacity and material constraints Eq. P3.d-e. For

every constraint that the continuous action meets, a reward equal to $c_i r_q$ is given. Since the objective function aims to maximize production, the larger the value of the continuous action, the larger the reward received. The values for r_q for each task are specified in Table B5 in Appendix B.

If all the constraints are satisfied, the action is executed in the environment; otherwise, the rewards are nulled, and a penalty is generated. The penalty term Pd_t is used to assess violation of allocation constraints from the discrete decision. As shown in Eq. 41, this term consists of nulling the rewards minus an additional penalty (cte), which was set for this case study to 100 based on preliminary tests. On the other hand, the value of the penalty for continuous decisions Pc_t consists in nulling the rewards minus the difference between the maximum acceptable value (ce_i) of the continuous action and the actual value provided by the agent (c_i), as shown in Eq. 42. Note that the value assigned to ce_i is assumed to be known *a priori* by the environment.

$$Pd_t = -rew_t - cte \quad (41)$$

$$Pc_t = -rew_t - |ce_i - c_i| \quad (42)$$

4.3.1.1 Validation of the Method

A comparison was made between a policy developed in a fixed (deterministic) environment (i.e., no uncertainty) against the solution found from solving problem (P3) from optimization. The POMDP agent (described at the beginning of this section) was trained in the environment without uncertainty with the objective of maximizing the production of products 1 and 2. It is expected that the policy will behave as an optimizer when the environment is deterministic. On the other hand, the same environment and objective function were solved in PYOMO using the CPLEX solver. The aim of this instance is to benchmark the results attained by the present method, and to demonstrate the capacity of the DRL framework to find a policy that can effectively build a schedule that is close to the optimal trajectory of decisions. Moreover, this validation serves to assess the effectiveness of the reward function for guiding the agent to find near-

optimal solutions. Fig. 28 and Fig. 29 show the schedules obtained from optimization and the DRL approach, respectively.

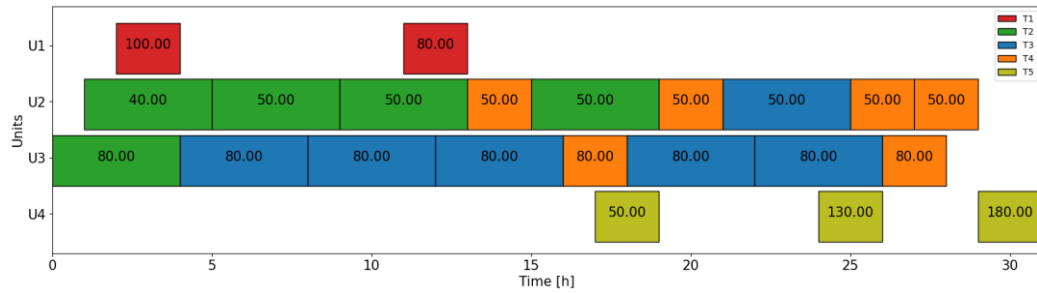


Figure 28 Deterministic solution for the scheduling problem of Case Study 1 developed with PYOMO.

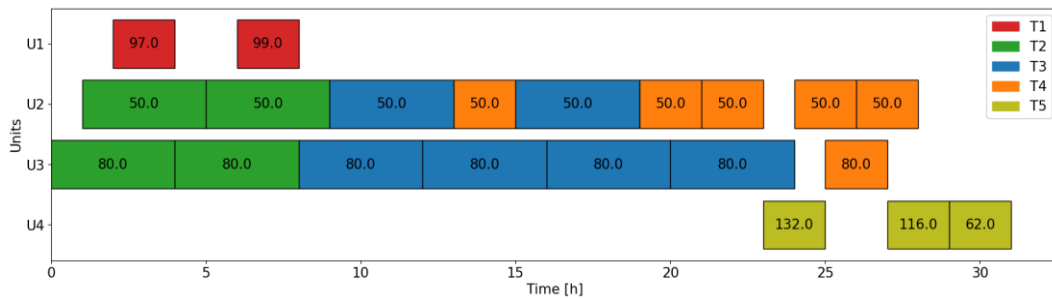


Figure 29 Deterministic solution for Case Study 1 developed with the presented methodology.

The optimal solution was found by PYOMO in less than 10 seconds which showed the advantage of these methods for short scheduling problems under no uncertainty. On the other hand, the POMDP agent found a suboptimal policy after 15 hours of training. Real-world scheduling problems have become a challenge in terms of defining an adequate reward function as the final solution is usually suboptimal [63], [103]. While the optimal solution produces 180 units of product 1 and 324 units of product 2, the POMDP agent produced 168 units of product 1 and 284 units of product 2, which represents almost 90% of the total production attained by the optimal solution. The main difference relies on the production of the agent from Task 2 at the beginning of the process which was not as high as that identified by the optimization-based solution. The limited product from Task 2 resulted in less production from the subsequent tasks.

The POMDP agent achieved a suboptimal trajectory in part because the DRL approach aims to find solutions (a policy) to multiple scenarios that could happen during operation. Also, the exploration in the DRL makes use of historical information from the process, i.e., no future decisions are considered to take

a decision, while the optimization solver explores the problem not as a sequential decision-making process but as an integral problem that considers all the time intervals while searching for the optimal solution. This comparison was useful to refine the reward function and to validate the correct performance of the environment. An MDP agent was also trained in the same fixed environment to compare the performance against the POMDP agent. Fig. B1 (Appendix B) presents two plots showing the mean performance across three trainings conducted with the POMDP agent and the MDP agent. At the end of the training the resulting MDP agent showed an almost 12% decrease in performance in terms of rewards when compared to the POMDP agent. For the results presented in the next sections, the policy of the POMDP agent developed under no uncertainty is used for comparison and referred to as the *deterministic policy*.

4.3.1.2 Epistemic Uncertainty

In this section, an instance with parameters under epistemic uncertainty is presented. This scenario aims to simulate a case where the output products of a reaction task are different from the expected outcome. For instance, when raw materials coming from multiple suppliers with slightly different specifications may produce variability in product quality, or when residues being left in the equipment (e.g., due to inadequate cleaning practices) may also result in deviations in product quality. As shown on Fig. 27, Task 2 transforms 40% of its input into Product 1, which goes to State E, and the rest goes to State H (i.e., 60%). To provide a more realistic scenario, these proportions (i.e., 60% going to State H and the rest to State E) are assumed to change every time Task 2 is executed. This can represent the case where intermediate species present in States D and F in Fig. 27 exhibit variations that can impact the production of Product 1 and therefore the production sent to State H every time Task 2 is activated. To model the variation of the output proportions, a Gaussian distribution is used to assign their percentage to the Task. The parameter $\omega \sim Y(\mu, \sigma)$ describes the percentage of product that goes to State H while the remaining percentage changes accordingly and goes to State E (i.e., $1 - \omega$). The value of μ and σ are set to 0.6 and 0.05, respectively. The distribution is truncated 5% above and below the mean value, thus the minimum and maximum values that ω can take, at most, are 55% and 65%, respectively. Note that previous studies modelled epistemic uncertainty using a

similar approach [114], [115]. For this work, the Gaussian distribution was used to represent variations in the production, corresponding to predictable changes when the characteristics of the raw materials are known.

Fig. 30 shows two plots. The solid lines in those plots represent the mean of five trainings of the POMDP agent and the MDP agent trained under this uncertain scenario, respectively. When compared to Fig. B1 in Appendix B, it was observed that, for both agents, the addition of uncertainty reduced the overall rewards compared with the *deterministic policy*. The POMDP agent trained in this case showed a gap of 14.24% against the *deterministic policy*, while the MDP agent showed a gap of 18.0%.

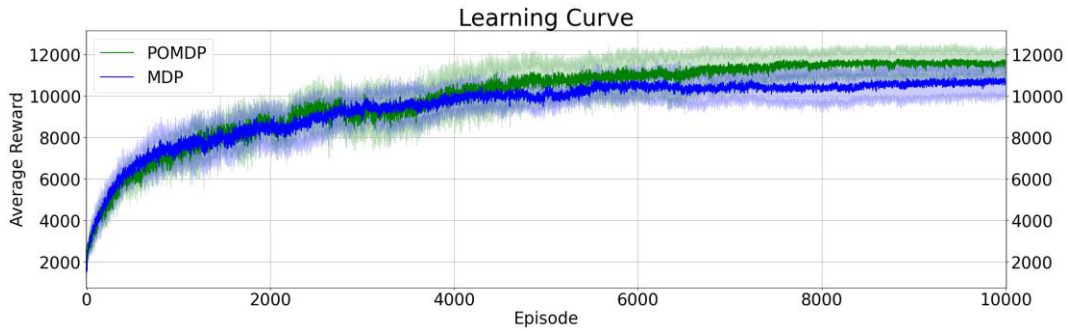


Figure 30 Learning curves for case study 1 with uncertainty in production of Task 2.

4.3.1.3 Aleatoric Uncertainty

In this section, two instances with aleatoric uncertainty are used to train the two agents. In the first instance, Gaussian noise is added to the states that hold the intermediate products of the process. In the second instance, a reduction of the information contained in observation \mathbf{o}_t is performed randomly at every time interval. Note that in both instances, the uncertainty affects the communication channel between the environment and the agent, i.e., the first instance deals with the problem of noise in measurements whereas the second instance address the issue of lack of information from the plant (maybe due to malfunction in the measurement devices).

In the first instance, \mathbf{o}_t is modified at every time interval through the addition of noise described by a Gaussian distribution. The noise is added to the States D, F, G, and H, which store intermediate products. States A, B, C, E, and I (i.e., feeds and product storage) are not considered. The feeds entering States A, B

and C are assumed to be unlimited so they were not incorporated into the observation \mathbf{o}_t as they have no relevant influence in the decision-making process. Similarly, the product storage in States E and I are not used for future tasks and therefore have no further influence in the decision-making process. The truncated Gaussian distribution $\theta \sim Y(1.0, 0.03)$ is used to generate four random values (i.e., one for each intermediate state) within a range between 0.95 and 1.05. Each of these values is multiplied by the actual values from the states. The new state values are incorporated in the observation \mathbf{o}_t and then sent to the agent to process the next action. This experiment aims to show the performance of the agent in industrial environments where multiple sources of noise (e.g., measurement noise) may corrupt the signals from the sensors [116].

Fig. B2 (Appendix B) shows two plots with solid lines representing the mean of five trainings of each agent (not shown here for brevity). Similar to the results shown in Fig. 30, the performance of the POMDP agent and MDP agent decreased by 7.67% and 13.97%, respectively, when compared to the *deterministic policy*. With this comparison, the effects of the uncertainty in the performance of the DRL agents can be observed and it can be concluded that these effects are more significant for the MDP agent.

For this instance, the POMDP agent retrieves more readings from the states (as it inputs multiple time intervals) which allows this agent to make more accurate predictions of the real value under aleatoric uncertainty. On the other hand, the MDP agent, which does not account for recurrence, can only consider the present observation, which does not provide sufficient and accurate information about the plant measurement. Thus, the MDP agent needs to be more conservative to avoid constraint violation thereby resulting in a decrease in performance, as measured by the rewards function.

For the second instance, the capacity of the POMDP agent to correlate observations and derive useful information from past events is tested. This instance suffers from strong interference in the observation window \mathbf{O}_t . In this instance, 50% of the values in the observation vector \mathbf{o}_t were randomly selected and set to zeros. With this experiment it was aimed to evaluate the capacity of the agent to find correlations when the information is scarce and highly intermittent in the process. These conditions can take place when there are multiple sensor failures, human error in reporting information, delayed information due to connectivity

issues, or unexpected environmental/market fluctuations. This shortage in information was a challenge for both agents due to the significant lack of plant data considered in the instance.

Fig. 31 shows the plots from the mean learning curves of the POMDP and MDP agents. A larger variance in the plot from the MDP agent can be observed in the figure. This is explained by the random reduction to the input of the agent which did not allow a convergence. This phenomenon can also be seen for the learning curve of the POMDP agent but in a less drastic way. Compared to the *deterministic policy*, the POMDP agent showed a decrease in rewards of 19.35% while the MDP exhibited a larger gap of approximately 27.57%.

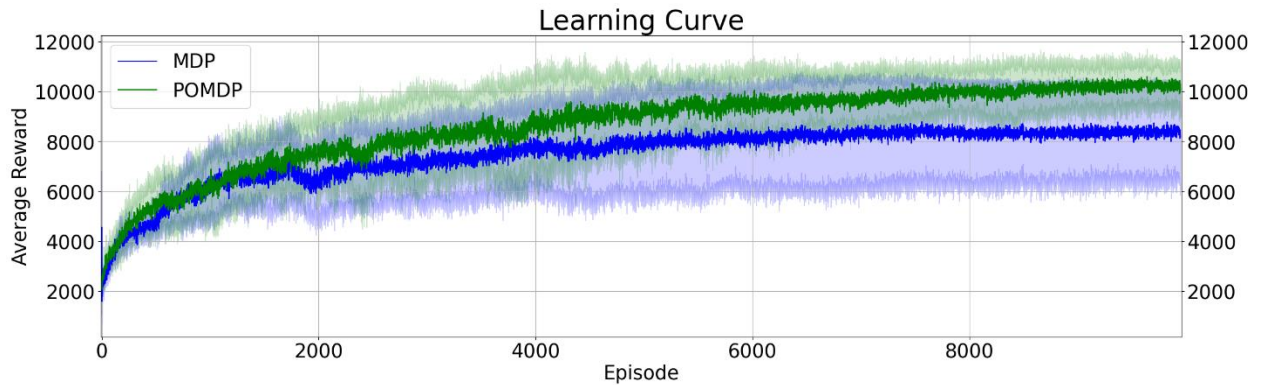


Figure 31 Learning curve for case with partial reduction on the observations.

4.3.1.4 Constraint Violation

The number of constraint violations that the agents made in the deterministic environment and the three instances under uncertainty considered the present case study are evaluated next. Recall that each time an agent violates a constraint, this will receive a penalty, and such action is cancelled and would allow the process to continue. To test the performance, the agents were tested on 1000 episodes where the percentage of violations for the discrete and the continuous actions were recorded. As mentioned above, the scheduling horizon consists of 31 time-intervals. At each time interval, a discrete and a continuous action were made; thus, a total of 62 decisions are made by the agent at each episode. Table 6 shows the percentage of violations per episodes for each agent presented for 1) the *deterministic policy*, 2) epistemic uncertainty in the production of Task 2, 3) Gaussian noise added to the measurements of the states in the plant, and 4)

intermittent communication in the channel between the agent and the environment. For most cases, the POMDP agent showed a reduction in the percentage of constraint violations, which demonstrates a better understanding of the dynamics of the environment under these conditions when compared to the MDP agent. Although the rewards for the POMDP agent were larger than the MDP agent in the 4th experiment (see Fig. 31), it was observed that the percentage of violations were larger for this agent, which may be due to the development of a more conservative policy.

Table 6 Average number of constraint violations from one thousand evaluations of the MDP and POMDP agents.

Experiment	Average number of violations per episode	
	MDP Agent	POMDP Agent
1.- <i>Deterministic policy</i>	16.53%	10.27%
2.- Instance under epistemic uncertainty in Production at Task 2	15.54%	12.58%
3.- Instance under aleatoric uncertainty applying Gaussian Noise in the measurement of the states	17.06%	11.43%
4.- Intermittent communication between the agent and the environment, representing aleatoric uncertainty.	12.11%	15.19%

4.3.2 Case Study 2

The present methodology was evaluated for a larger and more complex STN that was adapted from Papageorgiou and Pantelides [117]. That study presented a decomposition method to address a problem that was too large to be computationally tractable with common procedures. The approach consists of separating the process in campaigns that individually handle multiple tasks, which are necessary to complete an intermediate product. This intermediate product is then the input to the next campaign to produce the next intermediate product; the cycle is then repeated until the last campaign produces the final product. More details about this procedure can be found in [41]. Campaigns are useful to reduce the complexity of the problem by distributing the tasks in groups without affecting the operation process. In this work, campaigns can be readily applied through the masking technique, which is part of our methodology. The STN for this case study is presented in Fig. 32 and consists of 10 tasks, 3 products and 2 feeds. The process is divided into two campaigns (indicated with a red dashed line in Fig. 3): in the first

campaign, the intermediate products 1 and 2 (Int 1 and Int 2 in the figure) are produced from two independent sequences of tasks. The second campaign also consists of two sequences that share the intermediate product 1 to produce 3 final products (i.e., P1, P2 and P3) and a waste residual (Waste). The time horizon for this problem was set to 48 time-intervals.

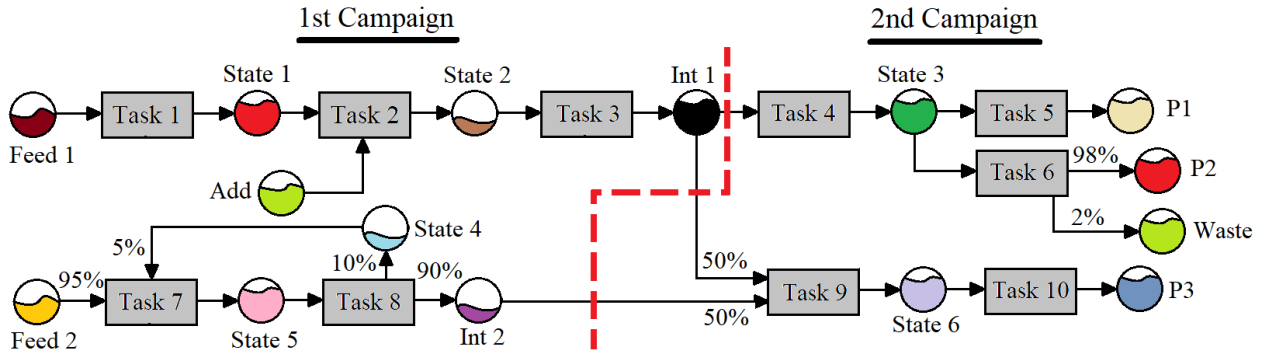


Figure 32 STN for Case Study 2.

Section B2 in Appendix B outlines the specifications of the problem regarding the relations between the states and tasks, as well as the specifications on the use of utility services. Specifically, the utility services for this case study are a) cold water, b) low-pressure service, and c) high-pressure service. Essentially, this case can be represented as an optimization problem with the same formulation presented in P3. Apart from the constraints presented in P3, this case study includes a constraint that defines the use of the utility services, i.e.,

$$0 \leq \beta_{ui\theta} B_{ijt} \leq U_u^{max} \quad \forall u, t, i, j \in K_i, \theta = 0, 1, \dots, p_{i-1} \quad (43)$$

where $\beta_{ui\theta}$ represents the required amount of the utility service u per kg in the task i in the related batch B_{ijt} . θ is the time-interval where the service is required, this time spans from the start to the end of the task. U_u^{max} is the maximum capacity of the service u .

The scheduling decisions for this case study are described in Eq. 44 whereas the discrete and continuous action spaces are depicted in Eq. 45 and Eq. 46, respectively. Similar to the previous case study, a list of tasks to units is presented; note that in this case, any task has more than one unit to process the product. The continuous action space describes the capacity at which the task is activated, which is bounded between

[0,1]. The rewards were designed as in the previous case study, with their corresponding weights shown in Table B7 (Appendix B).

$$F_t = \{action_d, c_1, c_2, \dots, c_{10}\} \quad (44)$$

$$actions_d = \{task\ 1\ at\ unit\ 1, task\ 2\ at\ unit\ 2, task\ 3\ at\ unit\ 3, \\ task\ 4\ at\ unit\ 1, task\ 5\ at\ unit\ 4, task\ 6\ at\ unit\ 4, \\ task\ 7\ at\ unit\ 5, task\ 8\ at\ unit\ 6, task\ 9\ at\ unit\ 5, \\ task\ 10\ at\ unit\ 6, idle\ state\}$$
(45)

$$c_i = [0,1] \quad (46)$$

The first campaign takes place from time intervals 0 to 24 whereas the second goes from time 25 to 48. The actions allowed in the first campaign are Tasks 1, 2, 3, 7, and 8; the rest of the actions are masked from the action space of the agent. During the second campaign, the tasks from the first campaign are masked while the rest of the tasks are allowed, i.e., Tasks 4, 5, 6, 9, and 10. Fig. 33 shows two plots where each solid line represents the mean of three trainings from the POMDP agent in the environment with and without masking. The difference relies on the use of the masking technique to reduce infeasible actions, i.e., avoid actions that are known *a priori* to be unnecessary, and separate the process into campaigns. The figure shows that the POMDP agent without masking converged in the first 500 episodes to a low reward value while the agent with masking keeps an increasing learning for a longer number of episodes.

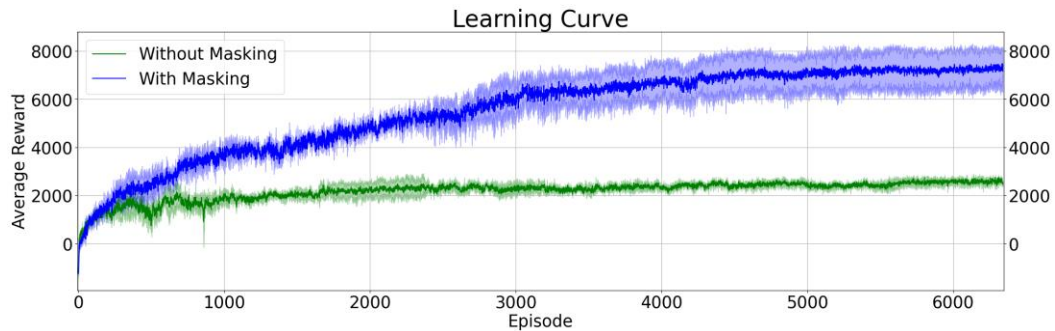


Figure 33 Effect of Masking technique.

Next, the POMDP agent with masking was evaluated for constraint violations with and without uncertainty. Note that the agent was only trained in the environment with uncertainty, and the environment

without uncertainty was not made available to the agent. The environment with uncertainty consisted in the partial interruption of 50% of the information from the observations \mathbf{o}_t . The POMDP agent was tested 1000 times in both environments, i.e., with and without uncertainty. This case has a horizon of 48 time-intervals in which discrete and continuous decisions are made, thus a total of 96 decisions were made by the agent. An individual analysis of the constraints that are evaluated considering the decisions of the agent was performed. These constraints are the allocation constraint (Eq. P3.b), the utility service constraint (Eq. 43), and the material balance constraints (Eqs. P3.c and P3e). Other constraints that are not related to the agent's decisions were not considered in this evaluation, for instance, the initial amount of material at each state (Eq. P3.f). Table 7 summarizes the average percentage of constraint violations performed in the environment. The allocation constraint is related to the correct selection of available units to be used, i.e., the discrete action. The utility service and the material balance constraints are related to the continuous decision of the agent. As shown in Table 7, it was found that continuous decisions were subject to violations, which could be reduced by increasing the penalties in the reward function, at the expense of generating a more conservative policy. Moreover, the POMDP agent could generalize in the environment with uncertainty and meet the constraints in a similar fashion as in the environment without uncertainty. Nevertheless, it was observed that on average the total production of the POMDP agent was reduced by 16.33% in the environment with uncertainty compared to the production in the *deterministic policy*.

Table 7 Average of constraint violations performed in 1000 evaluations of the agent.

	Environment without uncertainty	Environment with uncertainty
Allocation constraint	0.00%	0.00%
Availability of utility service	0.00%	0.02%
Material balance constraint	14.29%	18.96%

The experiments with this case study showed that using a masking technique decreased the complexity of the environment by discarding known infeasible actions. Thus, the masking method allowed a more efficient learning process. Nevertheless, the agent should not be excessively masked as this could restrict exploration to a small part of the action space. Therefore, fine tuning this technique is recommended to prevent the agent from adopting an overly conservative policy. The generalization capabilities of the

POMDP agent were demonstrated with the evaluation of the performance in an environment with uncertainty. The agent could maintain a similar record of constraint violations as in the original environment, which may be presumably because the source of uncertainty was in the communication channel between the agent and the environment rather than in the environment itself. Moreover, the use of LSTMs played an important role in the performance in the environment with uncertainty due to its capacity to adapt to the lack of information.

4.4. Summary

In this work a methodology for training DRL parameterized agents for online scheduling on STNs under aleatoric and epistemic uncertainty is presented. The main features of the agent are its parameterized action space which allows to complement an action, and the use of RNNs to approach the problem as a POMDP. The aim of the former is to increase the contextual information of the agent by providing observations from the past and complement the present information, which is a useful feature to handle different types of uncertainty. Challenges in this approach are related to the complexity of the parameterized agent with RNNs and the representation of the environment. For the latter, the hyperparameter tuning was crucial to achieve stable training and good performance, whereas for the former, a reward shaping method combined with a masking technique were needed to guide the learning of the agent.

The methodology was tested on two plants of different sizes and subject to uncertainty. The cases with uncertainty were solved with the present methodology and compared with an agent that does not use recurrence, i.e., an MDP agent. Results showed a better performance from the presented POMDP framework. The larger plant was used to demonstrate the functionality of the masking technique to simplify complex plants by separating the process in campaigns. In both case studies, the use of the hybrid agent allowed to train a policy that could handle multiple decisions in the process. On the other hand, the use of RNNs improved the collection of information from the environment which derived in better decisions and larger rewards.

Although RNNs are good to generalize POMDPs, these models require considerable training times and tuning. Another challenge in the training was that in some cases the algorithm would not reach the average performance, then the training would need to be repeated, presumably, due to the complexity of the environment. Future work to address this issue includes more sophisticated exploration techniques, such as the use of intrinsic curiosity modules. In the following chapter, the methodology presented in Chapter is modified and improved to consider the integration of the scheduling and control tasks with the hybrid agent.

Chapter 5 Hybrid Deep Reinforcement Learning Agent for Online Scheduling and Control for Chemical Batch Plants

The need for approaches that simultaneously consider the key decision variables from scheduling and control is justified by the increasing sources of information of endogenous and exogenous events taking place in the process during operation [45], [118]. Examples of these include the growing sensory data that can be collected from the process, the demand and production of a larger variant of products, or the changes in the prices in the market defined by external factors. Moreover, changes in these parameters are registered more frequently during the horizon which leads to the need for reactive methods that can avoid the infeasibility of a schedule or the missing of product quality and safety requirements. Both scheduling and control are dependent on each other but their purpose in the process is different, while the scheduling process defines the batches, starting times, and state reference, the control system aims to keep the process on target and close to its desired set-points [119], [120]. Also, they often evolve at different time scales just as the planning and scheduling problems [121]. To the author's knowledge, this integration has not been attempted in the literature of DRL applications in chemical processes.

In this study, a framework to address the integration of scheduling and control with a DRL agent for a flow-shop under uncertainty is presented. A hybrid agent that can output the scheduling and control decisions is used for the integration. To accomplish this goal, the method presented in the previous chapter is adapted to the integration process. The sequential decision problem is assumed to be a POMDP for which an ANN that uses LSTMs is considered. To illustrate the characteristics of this approach, an agent is trained with this scheme to take online decisions of the scheduling and control of a chemical flow-shop batch plant subject to variability in the process inputs, i.e., inlet material flows. The concentrations of these flows are assumed to follow a stochastic behaviour. The scheduling decision defines the task that should be initialized

while the control actions aim to accommodate the stochastic perturbations in the inputs while aiming to meet the task's production targets.

This study is organized as follows: Section 5.1 presents the problem statement; section 5.2 shows the proposed methodology; Section 5.3 shows a case study and results; conclusions and future work are presented at Section 5.4.

5.1. Problem Statement

This section presents the scheduling and control problem that can be addressed with the present approach. The problem definition was adapted from a previous work that addressed the integration of scheduling and control for chemical batch plants under stochastic uncertainty [122]. Our work does not approach the problem under uncertainty per se but instead assumes that there are stochastic parameters (ψ_{var}) defined by known PDF. The value of these parameters is only known when a unit related to these parameters is turned on, i.e., a task is assigned to this unit. Their values remain constant during the operation of that task. In the case of units involving process dynamics, values for these parameters are known at the beginning of the operation and its value remains constant until the end of that specific operation. Hence, appropriate control actions must be determined to accommodate these changes and be able to meet the unit operation targets. This situation demands an online decision-making process for scheduling and unit operation control that needs to adapt to the outcomes of such stochastic parameters. For the systems that can be considered with the method presented in this work, consider the following:

- A flow-shop plant that is composed by N_k set of tasks, with N_j set of equipment.
- A set of chemical processes described by the mechanistic dynamic model f for N_p states of the system, and expressions h that contain the set N_q of constraints of the system.
- A set Ψ of fixed model parameters ψ_{fix} and of stochastic parameters ψ_{var} described by a PDF known *a priori*.
- A set C that considers the cost information of utility services, raw materials, products, and by-products.

- A finite time horizon H which starts at t_s and ends at t_f .
- A finite number of equal-length time-intervals t that belong to the set N_t and are used for the discretization of the scheduling horizon H . At every time-interval t , scheduling and control decisions are taken.
- A set T of processing times indicating the length of time $\tau_{k,j}$ that the units in N_j takes to complete the tasks in N_k .
- An economic function G that considers product profits, operational times, costs related to utility services, penalties incurred during the operation, and other related costs.

The optimization formulation for the batch plant described above is stated as problem P4. This integrated optimization problem aims for an optimal schedule plan with dynamic control profiles that maximizes the profits of the process.

$$\begin{aligned}
& \max_{u_{k,j}(t), S_{k,j,t}} G(x_{k,j}(t), u_{k,j}(t), \varphi, \tau_{k,j}, S_{k,j,t}, c) & (P4) \\
\text{s.t.} \quad & f_p(x_{k,j}(t), \dot{x}_{k,j}(t), u_{k,j}(t), \varphi, \tau_{k,j}, S_{k,j,t}, t) = 0 \\
& \forall t, p \in N_p, t \in N_t, q \in N_q, k \in N_k, j \in N_j \\
& h_q(x_{k,j}(t), \dot{x}_{k,j}(t), u_{k,j}(t), \varphi, \tau_{k,j}, S_{k,j,t}, t) \leq 0 \\
& \forall t, t \in N_t, q \in N_q, k \in N_k, j \in N_j \\
& \tau_{k,j} \in \tau \quad \forall k \in N_k, j \in N_j \\
& S_{k,j,t} \in \{0,1\} \quad \forall k \in N_k, j \in N_j, t \in N_t \\
& x \in X \subseteq \mathbb{R}^{N_x \times N_j \times N_k} \\
& u \in U \subseteq \mathbb{R}^{N_u \times N_j \times N_k} \\
& c \in C \subseteq \mathbb{R}^{N_c} \\
& \psi_{fix}, \psi_{var} \in \Psi \subseteq \mathbb{R}^{N_\psi} \\
& \psi_{var} \sim f_{\psi_{var}}(\psi) \\
& \tau \in T \subseteq \mathbb{R}^{N_\tau} \\
& t \in [t_s, t_f], H = t_f - t_s
\end{aligned}$$

where f_p is the p^{th} differential-algebraic equation of the model and h_q is the q^{th} model constraint. $x_{k,j}(t)$ is a state variable from the system for task k taking place at unit j and $\dot{x}_{k,j}(t)$ is the derivative of that variable. $u_{k,j}(t)$ is a control decision variable involved in task k at unit j . The control actions $u_{k,j}(t)$ that can be taken at each time-interval t by the time-dependent unit operations j to maintain the process on target are limited

to a set of discretized actions included in the set U . $f_{\psi_{var}}(\psi)$ is the PDF used to describe the likelihood of a realization of ψ_{var} . $s_{k,j,e}$ is a binary decision variable that specifies the scheduling process for the batch plant. This variable is defined for every task k at unit j during the set of time intervals t and tells if such task and unit are occupied or not.

Problem P4 can be defined as a stochastic Mixed Integer Dynamic Optimization (MIDO) problem, which may become challenging to solve since the realizations of ψ_{var} are not known *a priori*. Approaches consisting of the decomposition of the problem into MILP or MINLP can be used to reduce the problem's complexity but they present their own challenges for online implementation due to intensive calculations [45]. Simplifications can be made with a trade-off between time response and quality of the solution. The DRL method presented in this work is used to train the agent for stochastic realizations of the input parameters and then, once trained, it can be implemented online and provide immediate (online) responses to the integrated process thus providing a fast reliable solution to this problem.

5.2. Methodology

In this section, the methodology to design the DRL agent that can provide online solutions to the flow-shop scheduling and control problem described in P4 is presented. A PPO method is used to train the agent as it is a stable and general-purpose algorithm. The following key components in the DRL framework are explained in this section: 1) the action space for the scheduling and control tasks; 2) the input sequence vector for the agent, which contains relevant information used to take a decision; 3) the reward function, 4) the time-dependent stochastic parameters, and 5) the architecture of the hybrid agent.

5.2.1 Action Space

The number of decisions that an agent will take at each time-interval t in the horizon H include both scheduling and control decisions. It is assumed here that both tasks need at least one action to be executed in the process, but this can be extended to multiple actions that correspond to each task. The set of action spaces is shown in Eq. 47, where all the needed actions for scheduling and control tasks are included. Note

that all the action spaces in this set are discretized; it is also possible to set continuous spaces, but this is beyond the scope of this work.

$$Actions = \{a_1, a_2, \dots, a_n\} \quad (47)$$

The scheduling action space can specify the task and unit to be initialized at each time-interval. Since an action is delivered at every time-interval t , the action space also includes an idle action that do not start any task. Other discrete actions related to the scheduling task can be added to the agent, for instance, the capacity at which a task should be started (i.e., 100%, 20%, etc.). Similarly, the number of control actions is given by the number of control decision variables included in the set U defined above. Since the control actions are discretized, the control profiles generated by the agent for each control variable $u_{k,j}$ will be similar to step-like functions applied to each time interval t .

During implementation, at a specific time-interval, the environment sends the sequence of observations from the process to the agent. Then, for every action space in $Actions$, a categorical distribution p_{a_n} is provided (Eq. 48). The largest probability at each action space distribution defines the action executed in the environment. Note that at all time-intervals, the agent generates categorical distributions for all the action spaces described in the set $Actions$. Naturally, not all of them are used at all time intervals, for instance a control action will be only used when the corresponding unit is active. Actions that are not required at a certain time interval are ignored by the environment.

$$p_{a_n} = (p_1, p_2, \dots, p_m) \text{ where } \sum_{m \in M} p_m = 1 \quad (48)$$

5.2.2 Observation Vector

A vector with the information from the environment called observation vector (\mathbf{o}_t) is generated at every time-interval and sent to the agent to produce the next set of actions. The vector \mathbf{o}_t defined in Eq. 49 contains the features that have a measurable effect in the scheduling and control tasks. In this work, \mathbf{o}_t gathers the following information: i) the occupation of the available units j ; ii) processing times of the units that are in use; iii) information relevant to the control problem, e.g., concentrations, flow rates, and temperatures; and iv) the current time-interval. This information is retrieved at every time-interval from the

environment and is also normalized to stabilize the learning process. Note that the representation of these variables in \mathbf{o}_t depends on the user's preferences. For instance, the use of probabilities, constant values, one-hot encodings, or normalized values can be used to pre-process these features and then add them to the observation vector \mathbf{o}_t .

$$\mathbf{o}_t = [x_{k,j}(t), s_{k,j,e}, \tau_{k,j}, t] \quad (49)$$

Although this state representation provides full observation to the agent, the use of the POMDP approach is justified to provide the agent with additional information from past events. The LSTMs correlate information in a temporal context, which is key to capturing the interactions between scheduling and control.

5.2.3 Reward Function

A reward shaping method was used in this work to decompose the total reward into m sub rewards r , i.e.,

$$Reward = \sum_m r_m \quad (50)$$

To implement the reward shaping method in the integrated scheduling and control problem P4, the straightforward procedure consists of defining rewards for each task. Rewards should be provided to actions that align with the objective function. Likewise, penalties should be delivered when constraint violations take place. The subset of rewards r_m for the scheduling actions is based primarily in allocation constraints and the correct order of task initialization in the flow-shop. This ensures that the conditions for initiating a task are met, for instance, availability of material or availability of units. Moreover, penalties for machines staying idle during a process should be set to ensure the reduction of the makespan. Furthermore, the reward should be increased as the process reaches the end, i.e., to provide larger rewards to the agent for initializing tasks that are closer to the end.

Regarding the control decisions, the agent should account for the economic incentives associated with these tasks. For instance, providing larger rewards to utility services that are less expensive. Penalties

should be provided for constraint violations that may occur during transient operation, e.g., surpassing a safety limit or not reaching the expected product specifications at the end of the task. It is assumed that any task k that comprises a dynamic system will need at least one time-interval to be completed. Hence, a sequence of control decisions, each implemented at each time-interval t , are required for the duration of that task. Then, the reward for that sequence is provided at the end of task k . Here, the POMDP becomes useful as the sequence of actions (i.e., the control profile) is receiving a reward, rather than only the present action. Thus, the use of LSTMs results convenient for handling the partial observability from the environment. After being sent to the environment, the actions from the set *Actions* that were implemented (recall that not all of them might be used) are evaluated with the system of rewards. When all the rewards are assigned to the scheduling and control decisions, they are added up and sent as a unique (scalar) reward to the agent.

5.2.4 Stochastic Parameters

The environment is a representation of the process where the agent can learn by trial and error on how to perform a task. The limitations of the real process (i.e., constraints) and the description of fixed and stochastic parameters (i.e., ψ_{fix} , ψ_{var}) should be added into this simulation. During each iteration of the training, the stochastic parameters are set to present the agent a specific realization of this parameter. The agent gathers experience from these realizations and generalizes a policy that can choose the best action for a given realization of ψ_{var} . The present works assumes that the models considered in the present framework are internally stable, i.e., the inputs and outputs from the system will be bounded to guarantee process stability. A Bounded Input Bounded Output (BIBO) approach of the system can be used to guarantee the stabilization of the process [123]. Nonlinear control systems might require different approaches like Lyapunov methods that can guarantee that the signal will not destabilize during the process [124].

In the environment, the mathematical description of the stochastic parameters should be incorporated through a PDF, e.g., a Gaussian or Uniform distribution. These values should also be incorporated in the observation that are sent to the agent as they are relevant for the next decision. Note that the increase in

time-varying parameters has a direct impact on the length of the training. This is because the agent needs to explore multiple realizations to learn its policy. Thus, the exploration space grows as more stochastic parameters are considered.

5.2.5 Hybrid Agent

The architecture considered in this study for the hybrid agent was adapted from [108] and is essentially the same as in the previous chapter. Since the control decisions are discretized and the scheduling decisions are all discrete, the set of n decisions generated by the agent are all categorical distributions. Each distribution gives the probability to each possible outcome of that particular action (see Eq. 48).

As in the explanation from Section 4.3, the hybrid agent uses a correlational module built with α layers of LSTMs to gain insights of the sequence of observations \mathbf{o}_t . The sequence \mathbf{O}_t is the same as in Eq. 33, in Section 4.4.2, (i.e., $\mathbf{O}_t = [\mathbf{o}_{t-\Gamma+1}, \dots, \mathbf{o}_{t-2}, \mathbf{o}_{t-1}, \mathbf{o}_t]$). As previously explained, this vector allows to register all the steps that a task takes to complete. That is, the agent can access the sequences that contain the evolution of the tasks from start to end, which is not possible under the MDP scheme.

The general architecture of the neural network for this approach is depicted in Fig. 34. The observation window \mathbf{O}_t is an input to the correlational module with α layers. Then, the outputs are passed through a set of K linear layers. The final output of this section is input into n separated sets with Z linear layers. The output of each set is passed through a SoftMax activation function to output each categorical distribution.

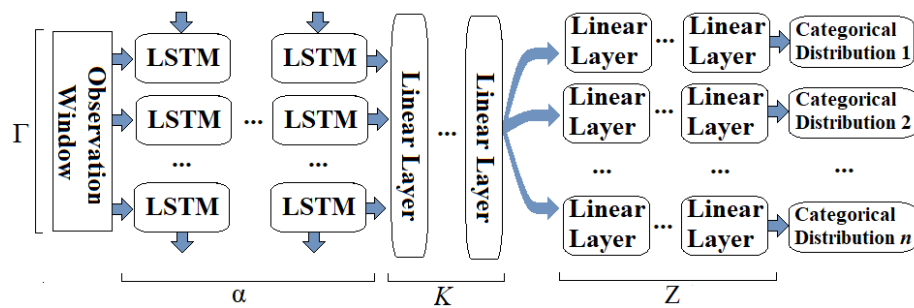


Figure 34 Hybrid agent with correlational module.

With this architecture, an agent that can coordinate both scheduling and control tasks can be trained. The incorporation of LSTMs into the agent allows to correlate the observations. This is a key feature that

provides the agent with a wider visualization of the development of the processes. On the other hand, limitations of networks with recurrence include the computational effort that they require as they evaluate the sequence element by element, i.e., not in a parallel fashion.

5.3. Case Study and Results

The methodology presented in the previous section is applied to perform the simultaneous scheduling and control of a flow-shop batch plant adapted from the literature [122]. Fig. 35 shows a representation of the process which is composed by four tasks: 1) a set of chemical reactions (RI), 2) a filtration process (FI), 3) a second set of reactions (RII), and 4) a separation process (SI), i.e., $N_k = \{RI, FI, RII, SI\}$. The time horizon H is set to 15 hours and is divided in even time intervals $t = 0.5$ h; thus, there are 30 time-intervals t where the agent can make decisions.

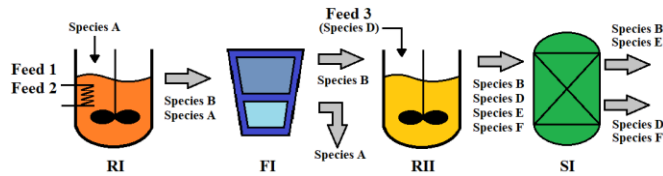


Figure 35 Batch Plant model.

The process begins in RI, where a non-isothermal batch reactor is used to transform substance A into an intermediate product B. The temperature of the reactor is controlled through the flow rates of Feed 1 and Feed 2, which correspond to a hot and a cold stream of water, respectively. The resulting mixture of products A and B from RI is then filtered in task FI. The intermediate species B is then passed to a semi-batch reactor where species D is added through Feed 3. The flowrate of Feed 3 is controlled during the process to regulate the production of products E and F. The mixture of species B, D, E, and F (where E is the desired product) is separated in task SI, as shown in Fig. 35. Processes FI and SI are assumed to achieve perfect separation and are stationary processes. The four tasks have a fixed operation time of 2h. RI and RII are assumed to be dynamic processes and are expected to finish at their corresponding processing time. Each task has one unit, i.e., $N_j = \{Reactor1, Filter, Reactor2, Separator\}$. After completion, each unit will store the outgoing material until the following task is initialized. While the material is stored in that

unit, a subsequent task for this unit cannot be initialized. The mechanistic models of the dynamics systems, the process constraints, model parameters and initial conditions for this case study can be found in the supplementary material from [122]. For the present case study, the initial concentrations of species A entering RI and D in Feed 3 vary according to a uniform distribution. For the former, the distribution is bounded between 1.01 and 1.15 $kmol/m^3$, while the latter is bounded between 0.80 and 0.89 $kmol/m^3$.

The economic function is described in Eq. (50) and aims to maximize the revenue from the production of product E while considering the minimization of costs related to the utility services from the three controlled flow rates corresponding to Feeds 1, 2, and 3 as described above.

$$G = cost_E * Storage_{SI} + cost_{utility\ services} \quad (51)$$

where $cost_E * Storage_{SI}$ denotes the revenue obtained from the product that is output from task SI . The costs of the product E and utility services are specified in the supplementary material from [122]. The objective of the DRL agent is to design a policy that can produce an online schedule and a control strategy considering the stochastic variability in the concentrations in the raw materials, i.e., species A and D.

At every time-interval t , the agent will provide four ($n = 4$) decisions: 1 for the scheduling task, and 3 for the control task. For the scheduling task, the agent will decide which task to activate; hence, this decision has 5 possible outcomes (i.e., 4 tasks and 1 idle action). The control task manipulates the flows of Feeds 1, 2, and 3 (see Fig. 35) while the task RI and RII are in operation. For every time interval cloistered in the operation time of these tasks, the agent sends a control decision to the environment. Then, the states of tasks RI or RII are sent back to the agent where another control decision is made for the next time-interval. This is repeated until the end of the task. The control actions for each flowrate are discretized as shown in Table 8.

Table 8. Flow-rates for the control tasks.

	Low m^3/h	Medium m^3/h	High m^3/h
Feed 1	3.75	7.5	16.0
Feed 2	3.75	7.5	16.0
Feed 3	0.08	0.9	1.7

Since the processing times ($\tau_{k,j} = 2.0h$) are the same for every task and since a time interval is of length 0.5h, the control profile for each process has four stages. If the reaction task does not reach the desired concentration at the end of the processing time, or if one of the constraints is violated during operation, then the product will be wasted away from the process. For the subsequent actions, the agent should consider this interruption and schedule tasks accordingly. The system of rewards shown in Table 2 was used to guide the learning of the agent to aim for the maximization of profit.

Table 9. Rewards assigned to the actions.

	Description of the rewards
Scheduling actions	Reward +5 if the unit for that task is not busy when initializing a task and -10 if the unit is busy. If the task is completed to the end, then give a reward of +10, if the process is interrupted due to constraint violation, give a penalty of -5. For every time interval that tasks RI, FI and RII keep the product stored, a penalty of -0.5 is addressed.
Control actions	Feeds 1 and 2: If Task 1 (RI) is running, provide a reward of +3 if the high profile is chosen, +6 for the medium profile, and +9 for the low profile. This is done for each hot and cold flows. The total reward is the addition of both.
	Feed 3: If Task 3 (RII) is running, provide a reward of +6 if the high profile is chosen and applied, +12 if it is medium profile, and +18 if it is the low profile.

The length of the observation window is set to 4 (i.e., $\Gamma = 4$) since each unit is assumed to require 4 time-intervals to complete a task. The relational module has three layers of LSTMs ($\alpha = 3$), followed by two linear layers, i.e., $K = 3$. Then, the output of this set of linear layers is the input of each of the sets of linear layers that represent each action ($n = 4$); each one with three linear layers ($Z = 3$). Hidden layers use tanh activation functions while a SoftMax activation function is used to generate the categorical distributions. The DRL method was developed in Python 3.11.3 and PyTorch version 2.1.0. The training was performed using Adam’s optimizer, which is gradually decreased with a learning rate annealing technique, starting at $1e-4$ and finishing at $1e-6$. To refine the exploration as the training progressed, the smoothness of the categorical distributions was sharpened using temperature annealing, starting at 1 and ending at 0.001.

A test with 1000 multiple realizations in ψ_{var} was performed with the trained policy π_v . Fig. 36 shows the number of times (cycles) that the process in Fig. 35 was executed in the horizon H . In almost 90% of the realizations, the agent could complete two to three cycles. The agent could not set three cycles in all the instances due to time limitations or a task failure, as discussed below. To compare the performance of the policy obtained by this method (π_v), a second policy (π_f) was trained on the same environment but under no variability in the inlet concentrations, i.e., ψ_{var} were fixed to 1.08 kmol/m^3 and 0.845 kmol/m^3 for species A and D, respectively. The resulting policy π_f was then tested under 1000 stochastic realizations in ψ_{var} . As shown in Fig. 36, the strategy learned by π_f could handle many of the scenarios on which π_v was trained. This was expected because the control profiles that were learnt by π_f may be adequate to accommodate most of the stochastic scenarios; nevertheless, they might not align with the objective function. In half of the processes, π_f could not produce a full cycle, i.e., no production; only in 13.3% of the instances it was able to complete three cycles as policy π_v .

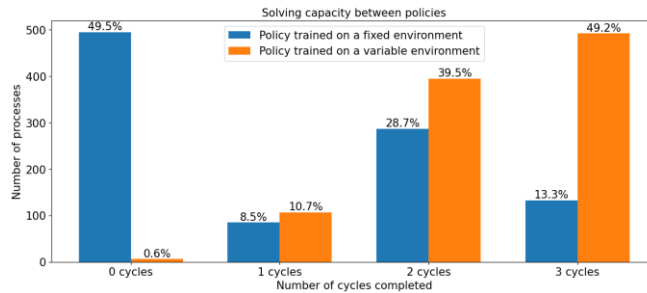


Figure 36 Comparison between policies.

Fig. 37 shows a schedule built with π_v in which the first cycle was interrupted due to the cancellation of Task RII. Once this event took place, the agent chooses the initialization of Task RI instead of Task S, which aims to produce more product E thereby improving the plant's profits. This decision corresponds to the reaction of the agent to the suspension of the first cycle. It was observed that the agent left blank spaces in between tasks, presumably due to a lack of sensitivity to the observation window. Also, it was observed that some tasks that were already in operation were selected to be initialized by the agent at those specific time-intervals, which results in an infeasible action. This problem can be addressed by increasing the

penalties associated with violation of allocation constraints in exchange of a more conservative agent. A heuristic that cancels the initialization of infeasible actions was added to the environment, i.e., new tasks that must be started on a unit that is already in operation are cancelled.

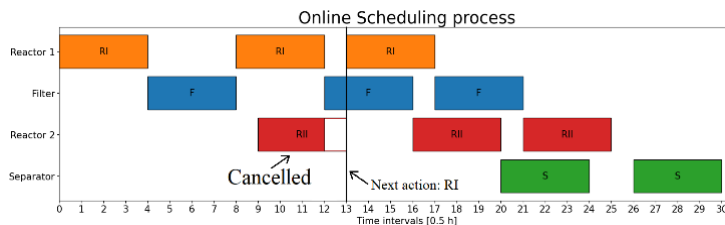


Figure 37 Schedule with three cycles initialized.

In most of the cases, the control profiles found by π_v completed reactions RI and RII without any constraint violations (not shown for brevity). Results from Fig. 36 confirm the effectiveness of the control actions chosen by the hybrid agent since it completed multiple cycles under stochastic variability in the inputs parameters affecting RI and RII.

5.4. Summary

In this work a methodology based on a DRL to address the integration of scheduling and control in flow-shop batch plants is presented. The DRL method allows to build online schedules and control profiles according to the conditions that take place in the plant. A POMDP approach is used to retrieve past history from the plant and consider the tasks evolution in the decision-making process. To handle multiple decisions taken during transient operation, a hybrid agent is used to output more than one action at every time-interval. To validate the method, a batch plant was considered for optimal scheduling and control. The agent designed an effective online schedule and control policy that can accommodate stochastic realizations in the input parameters. The application of this method for partially observable environments is part of the future work. Extending this approach for integration of scheduling and control considering discrete and continuous actions is also recommended for future work.

In the next chapter, the attention modules are used as an encoding method, similar to the job that the RNNs performed in this and the previous chapter: i.e., to correlate the sequence of information from

multiple time-intervals. The interpretability that the attention modules provide is analysed and evaluated for cases under uncertainty and for non-stationary environments.

Chapter 6 Interpretable Online Scheduling for Chemical Batch Plants with Attention Augmented Reinforcement Learning

Agents

In this chapter, a method for approaching the scheduling problem in Chapter 4 using self-attention modules is presented. The use of these attention modules in the agent's architectures is justified with the growing interest that these structures have gained in areas of NLP where RNNs were substituted with Transformers [31], i.e., a sophisticated encoder-decoder module used for sequence-to-sequence transformation which uses self-attention modules in its configuration. The Transformer is useful for training NLP models due to their outstanding capacity to handle sequences, for instance in language translation or human-machine interaction. In this work, the self-attention module included in the Transformer is adapted to the scheduling problem under epistemic and aleatoric uncertainty [31]. Essentially, the scheduling environment presented in Section 4.3.1 is represented here as a graph. This graph representation is required by the attention module that is incorporated in the agent. The RNNs used in the previous agent are substituted with self-attention modules, and the hybrid section of the agent which is useful to produce multiple decisions at every time-interval is maintained. The hybrid agents in this work were trained with the PPO algorithm.

SAMs are used to generate attention matrices ($\mathbf{A}_{t,k}$) at every time-interval t from a particular training course k to gain insight from the decisions of the agent. The attention matrix is a square matrix where all the nodes of the graph are correlated with themselves through attention values. Every time the agent processes an input, an attention matrix is produced and can be extracted to observe the current focal points of the agent in a particular state from the environment. The extraction and interpretation of the attention matrices from the agent are part of this methodology. This interpretability is an alternative to address the

condition of the agent being a black box with encrypted logics in its parameters. This lack of visibility from the agent's logics is a relevant aspect of DRL implementations where decisions involve safety and security facets that require confirmation and evidence of attention from the decision-maker.

As mentioned in Chapter 1, the implementation of the self-attention modules in the DRL method for the scheduling problem under uncertainty is based in the work from Zambaldi et al. [29]. Originally, this method is for application and interpretation of DRL agents in video games, where the agent must respond in an online fashion to the challenges of its environment. The interpretation that this method provides allows the user to see the focal points of the agent when it is moving, fighting, aiming, or defending. The methodology presented in that previous study has been modified such that it can be used on the scheduling optimization problem under uncertainty considered in this work. The case study 1 in Chapter 4 and a variation of such case are used in this chapter to test the present method and the results obtained with the current implementation are compared with those obtained from Chapter 4. In this work, we enhanced the basic architecture of the module such that the agent can build a general perspective of the relevance of each element in the environment. This additional element in the architecture is called bias matrix (\mathbf{B}_k) and is generated for every training course k . The use of the attention and the bias matrices in the scheduling problem will allow the validation of the system of rewards. Also, they can provide evidence of the sensitivity of the agent to different rewards, which can provide insights into the design of rewards and eventually more attractive policies.

This chapter is organized as follows: Section 6.2 presents the problem statement; Section 6.3 describes the methodology; Section 6.4 presents two case studies solved with the presented methodology; Section 6.5 presents a summary of this chapter.

6.1. Problem statement

Section 4.1 presents the problem statement that will be considered in this work. The scheduling process is represented as an STN and requires compound decisions, i.e., multiple decisions at the same time. The decisions made by the agent can be discrete or continuous (e.g., flow-rates, batch capacities, processing

times). The problem statement is explained and described as a mathematical formulation in Eq. (P2). The agents with attention developed with this method are evaluated in environments with epistemic and aleatoric uncertainty, as described in Section 4.3.1.2 and 4.3.1.3, respectively.

The agent in this work is also hybrid and can make multiple decisions in an online fashion to improve the objective function G while taking into consideration operational, quality and/or logistic constraints. Apart from making decisions in the online scheduling process, this agent aims to provide interpretability in its decisions through $\mathbf{A}_{t,k}$ and \mathbf{B}_k . In the context of this work, this interpretability is a numerical estimation of the relevance that each part of the process represents to the agent when making a decision. Actually, both the actor and the critic are structured with essentially the same architecture, differentiated in their outputs. Thus, both trained models can provide insight into different focal points that the agent develops through interactions with the environment. The scope of this work centers in the interpretation of the actor. Additionally, the agent designed in this work provides the benefits described in Chapter 4, i.e., a fast response at implementation, its capacity to handle complex environments that require compound actions, and the online reaction to uncertainty realizations.

6.2. Methodology

The methodology to approach the scheduling problem under uncertainty with agents that incorporate SAMs is presented here. Also, the methods for self-attention matrix interpretation ($\mathbf{A}_{t,k}$) and bias matrix interpretation (\mathbf{B}_k) are provided. Fig. 38 shows a flowchart that connects the methods presented in this section. The design of the RL elements based on the scheduling problem is presented in Sections 3.1 to 3.5. As shown in Fig. 38, once a trained agent has been accomplished, the methods for interpretation are used on $\mathbf{A}_{t,k}$ and \mathbf{B}_k . Two methods are presented in this Section: the interpretation of $\mathbf{A}_{t,k}$ (Sections 6.2.5 and 6.2.6) and the interpretation of \mathbf{B}_k (Section 6.2.7). While the former provides insight into the decisions of the agent at every time-interval (and is useful to track uncertain parameters), the latter provides a general perspective of the agent's attention during an entire episode within the scheduling horizon. Note that an episode refers to a cycle of the entire scheduling process. Moreover, a method for retraining non-stationary

environments is proposed to address cases where a change in the system of rewards used to train the current policy is required. Examples of these changes can occur in a chemical plant when the cost of a process is updated, e.g., changes in the costs of raw materials or service utilities or when new products are introduced in the process. Thus, the decisions of the current policy trained without these considerations may not be optimal and need to be updated. In this method, after the rewards are redesigned, a transfer learning method is performed before retraining. This step ensures that the new policy maintains the basic knowledge from the previous policy (further details in Section 6.2.7).

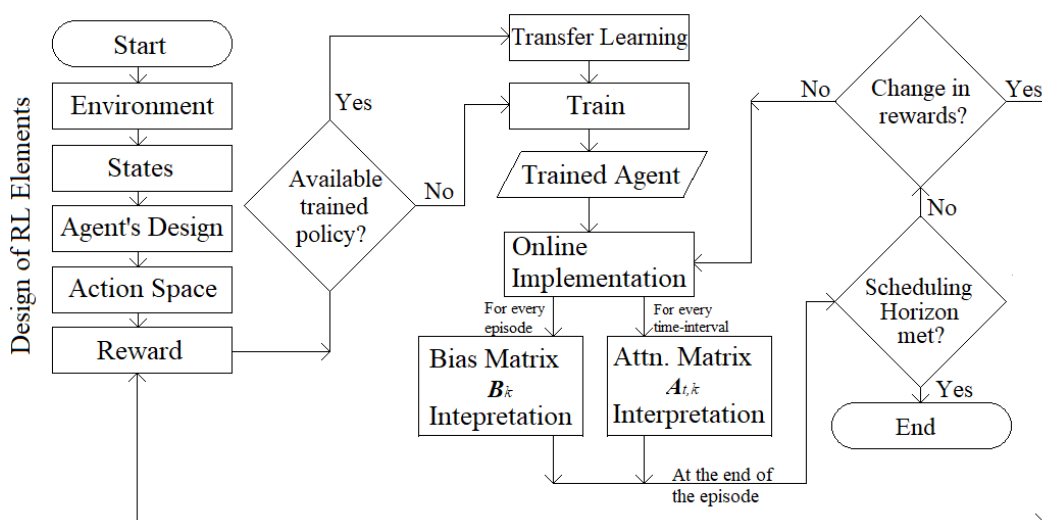


Figure 38 Flowchart of the methods presented in this work

The main novelties with respect to the method from Chapter 4 (referred from heretofore as *RNN approach*) are listed below:

- The architecture of the agent includes a self-attention module which requires modeling the agent's inputs as graph representations of the environment.
- An interpretation of the agent's decision is provided at every time-interval.
- An additional interpretable matrix (bias matrix) is used to observe the relevance from each part of the partially observable environment.

6.2.1 Graph Representation of the Environment

The design of the environment requires emulating a graph representation $G(\zeta, e)$ with nodes ζ and edges e . For the batch plant, the nodes correspond to the interconnected equipment and storage tanks that exist in the STN, for instance, the four states and three tasks depicted in Fig. 23. This is a distinctive feature introduced the current work since it allows a more organized representation of the environment (through nodes and features) in comparison with the state representation (expressed as a vector) from the *RNN approach* [18]. Although the literature has shown the use of disjunctive graphs to make representations of job-shops, in this work, complete graphs are used due to the posterior use of the attention matrices [79], [80], [85]. Complete graphs allow to generate connections between all nodes to establish a relationship between every processing and storage unit. On the other hand, disjunctive graphs create connections only between nodes that are directly associated in the environment, and will not provide attention values between nodes that might appear unrelated in the process. The use of complete graphs allows to observe correlations between these processes that are apparently not connected and, at a higher level of abstraction, the observation of correlations between processes from different time-intervals. Fig. 39 shows the representation of a complete graph where all nodes are connected.

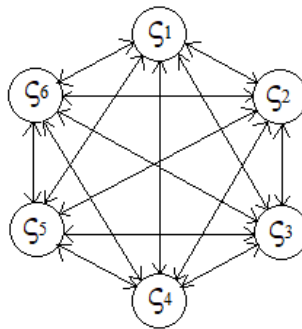


Figure 39 A complete graph with connections between nodes ζ which refer to the state and tasks from the environment. The edges show the existing connections between nodes, each arrow represents two edges and describe the attention from one node to the other.

Since the present approach contemplates the environment as a POMDP, the process of inputting a sequence of time-intervals is followed. Specifically, the SAM inputs one matrix with the information from Γ time-intervals concatenated. This input matrix is described in detail in Section 6.2.4. In this section, we

focus on the information expressed in one time-interval, which is also expressed as a matrix, and referred to as the observation-matrix Ξ_t . The number of rows m in such matrix corresponds to the number of nodes (ζ) that are in the graph representation. The number of columns l specifies the features η used to collect the characteristics of these nodes.

The nodes in the graph representation Ξ_t correspond to the processing units and storage-tanks in the environment. A more explanatory matrix can be made if the nodes denote individual processes instead of processing units. For instance, if a unit can perform two different processes, then each of these processes is defined as a node in Ξ_t . The features collected from the plant correspond to the state variables that describe the current state of it. For instance, variables that describe occupation and allocation, the current capacity of a unit, the processing time of a process, and other parameters that describe the node at the current time-interval. The addition of fixed parameters to Ξ_t can be done to provide information to the agent about inherent features that all nodes share, for instance, minimum capacity requirement or maximum temperature constraint. Eq. 52 describes the composition of the observation-matrix at a given time t .

$$\Xi_t = \begin{bmatrix} \xi_{1,1} & \cdots & \xi_{1,\eta} \\ \vdots & \ddots & \vdots \\ \xi_{\zeta,1} & \cdots & \xi_{\zeta,\eta} \end{bmatrix} \quad (52)$$

In the observation matrix Ξ_t , both types of nodes (i.e., processing units and storage tanks) may not necessarily share the same features. For example, assume that one of the features listed in Ξ_t is the processing time of the node, this applies well to a process like filtration as it is a process with this temporal feature, but for a storage tank, this feature may not apply. For these cases, the feature is left with a zero value in the input matrix. The experiments presented in Section 6.3 show that this does not represent an issue for the decision-making process of the agent or the interpretability of the attention matrices.

6.2.2 Uncertainty

In the present work, the policy training method can provide online scheduling decisions subject to uncertainty realizations in the scheduling process. DRL training allows the agent to develop a policy that can be applied in multiple circumstances, and to act accordingly to events in the environment. The training

of the agent can be performed under epistemic or aleatoric uncertainty. Parametric uncertainty is incorporated into the environment using a PDF such as Gaussian or Uniform distributions. The proposed agent gathers an observation window with the information from a sequence of time-intervals. Aleatoric uncertainty is introduced as intermittency or noise in the communication channel between the agent and the environment, resulting in a partially observable environment. This intermittency is carried out by randomly selecting a section from the input matrix and then modifying it into a zero value. This value was chosen as it is used to indicate that the unit/machine represented by the node is empty, other values might negatively interfere the training and make the agent fail on identifying this intermittency. With a zero value, the agent will receive a fixed signal in every value that is randomly altered in the input matrix.

The interpretability of $\mathbf{A}_{t,k}$ is used to confirm the interest from the agent on parameters under uncertainty that are represented as nodes. An analysis of this information can support the consideration of uncertainty in the decision-making process. Moreover, since the environment is approached as a POMDP, this analysis can be done in a temporal context, and the influence from past events can be measured and compared between each other. As in the *RNN approach*, the history of observations provided to the agent, augments its capacity to understand the consequences of the uncertainty in the process. Thus, the transition dynamics from the environment can be learned by the policy regardless of the uncertainty realizations.

6.2.3 Action Space and Rewards

Both the definition of the action space and the technique used for defining the rewards are the same as the method presented in Chapter 4. Moreover, the methods for exploration are also like those used in Chapter 4. The masking method and the rate annealing are used in the same way in this work. In this section, a brief review of these methods is presented next.

The agent holds the same hybrid characteristics as in the method presented in Chapter 4, i.e., an agent that can make multiple decisions at every time-interval t during the horizon H . The mathematical definition of the set F_t of decisions is the same as that one described in Eq. 32. This set includes n decisions d , where each one is described by an action space that could be discrete or continuous. Similarly, the reward shaping

method that was described in Section 4.2.2.3 is implemented in this case. To achieve effective updates of the actor and the critic, PPO needs explicit signals from the reward, which can be done through reward shaping. Thus, this methodology keeps these techniques as they are useful for the learning algorithm.

The use of weights to define the rewards is used as well, with the difference that such weights are scaled down from the values in Table B5 (see Appendix B) due to the use of normalization layers in the architecture of the agent. This change aims to prevent instability during the training. Nevertheless, the proportion of these values is maintained so the comparison between the results can be made.

6.2.4 Self-Attention Hybrid Agent

Like the RNN layers that were used for processing the input in the *RNN approach*, a SAM is used to process the input matrix. Both the RNNs and the SAM build an interpretation of the elements from the sequence that is gathered from the process. This interpretation (referred to as the hidden state in the context of RNNs) is processed in a set of linear layers, and it is then translated into concrete actions that are sent to the environment. In this section, the structures of the actor and the critic networks are described, including the attention and bias matrices ($\mathbf{A}_{t,k}$ and \mathbf{B}_k) that are generated during the process. The hybrid part of the structure is also described. The use of SAMs is proposed to verify the attention from the agent to specific aspects of the scheduling process.

A schematic representation of the input matrix (also called observation window \mathbf{O}_t) is presented in Fig. 40. As mentioned before, the input matrix holds Γ independent observation matrices $\mathbf{\Xi}_t$ in sequence from the process. The dimension of the input matrix is $\Gamma\zeta \times \eta$, in which the multiplication of the terms defines the total number of nodes in the input (equal to the number of nodes ζ in the plant multiplied by the number of time-intervals Γ in the observation window). The variable η correspond to the number of features describing each node, which are depicted as columns in Fig. 40. Since the SAM cannot differentiate the temporal position of each node in the input matrix, a temporal encoder must be added to the second dimension (η) in the input matrix as an extra set of features. RNNs have the inherent capacity to do this because the input sequence is distributed into a set of LSTMs that are equal in length. The temporal

encoding (or positional encoding) provides a unique reference to every node that belongs to a specific time-interval. In other words, each time-interval in the horizon H has its own time code, and this is added as a set of ϱ extra features to each node. Then, the dimension of the input matrix is modified to $\Gamma\zeta \times (\eta + \varrho)$. Note that the time encoding is the same value for all the rows belonging to the same time-interval.

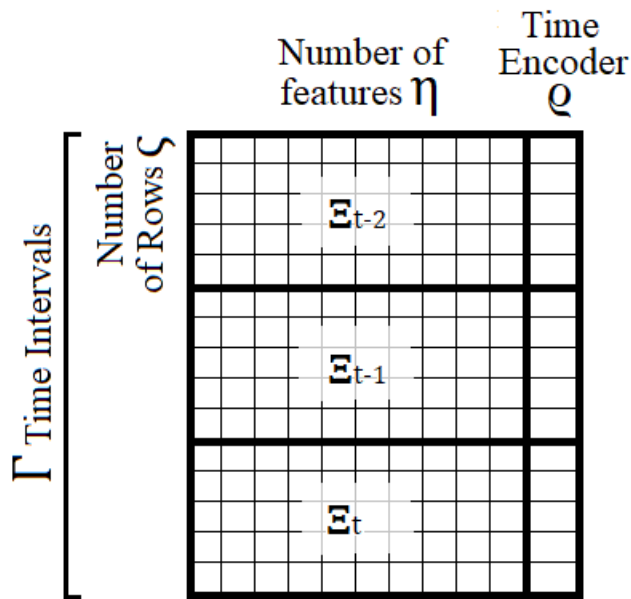


Figure 40 Depiction of input matrix with temporal encoding aggregated. In the figure, three observation matrices \mathbf{E}_t are concatenated.

Fig. 41 shows the architecture of the actor that is used in this work and that was adapted from previous studies [29], [108]. The observation window \mathbf{O}_t is copied three times, and each copy is projected (transformed) using a linear layer and then normalized with a normalization layer with dimension $\Gamma\zeta \times \iota$, where ι is a feature dimension that expands the original features from the input matrix. The three outputs correspond to the queries Q , the keys K , and the values V referent to the input matrix. As mentioned in the explanation in Section 2.2.3, the attention module is a matrix with the correlations between the nodes listed in the graph representation and it can hold multiple attention heads; thus, $\mathbf{A}_{t,k} \in \mathbb{R}^{\Gamma\zeta \times \Gamma\zeta \times h}$.

The output projections from the queries and the keys are then passed through an additive attention module, which describes the relation that exists between queries and keys. Since both projections in this module are derived from the same input, the compatibility module captures the correlations between nodes in the graph representation of the environment. In the additive attention module, the outputs from the

projections are independently passed through a linear layer, summed, passed through an Exponential Linear Unit (ELU) activation function, and passed through another linear layer. The output from the compatibility function ($\mathbf{A}_{t,k}$) is passed through a SoftMax activation function, which scales the values from each row in such a way that they add up to one. Before passing through the activation function, $\mathbf{B}_k \in \mathbb{R}^{\Gamma_\zeta \times \Gamma_\zeta \times h}$ is added to $\mathbf{A}_{t,k}$. \mathbf{B}_k is composed by learnable parameters that are updated during the training. In other words, \mathbf{B}_k holds fixed values (e.g., weights or coefficients) that increase the attention value regardless of its current state value. For instance, if the value of \mathbf{B}_k for a specific node-to-node relation in a specific head is of 0.03 and the self-attention values for those relations in $\mathbf{A}_{t,k}$ for states at times 0, 1, and 2 are {0.02, 0.00, 0.1}, then the new values of $\mathbf{A}_{t,k}$ after the addition would be {0.05, 0.03, 0.13}. The values of the parameters from \mathbf{B}_k are defined by the training process and can be positive or negative. The incorporation of \mathbf{B}_k to the architecture from the SAM is a key feature used to identify the relevant nodes in the environment during the scheduling horizon.

\mathbf{B}_k exerts an influence on the values from $\mathbf{A}_{t,k}$ that can be interpreted as a positive or negative encouragement. Positive values in \mathbf{B}_k will increase the values from the respective nodes in $\mathbf{A}_{t,k}$. On the other hand, negative values discourage the attention from the nodes that are not relevant to the agent. The encouraged values are those that are relevant to the process during the entire episode, while the discouraged values correspond to those that are not contributing to the decisions of the agent.

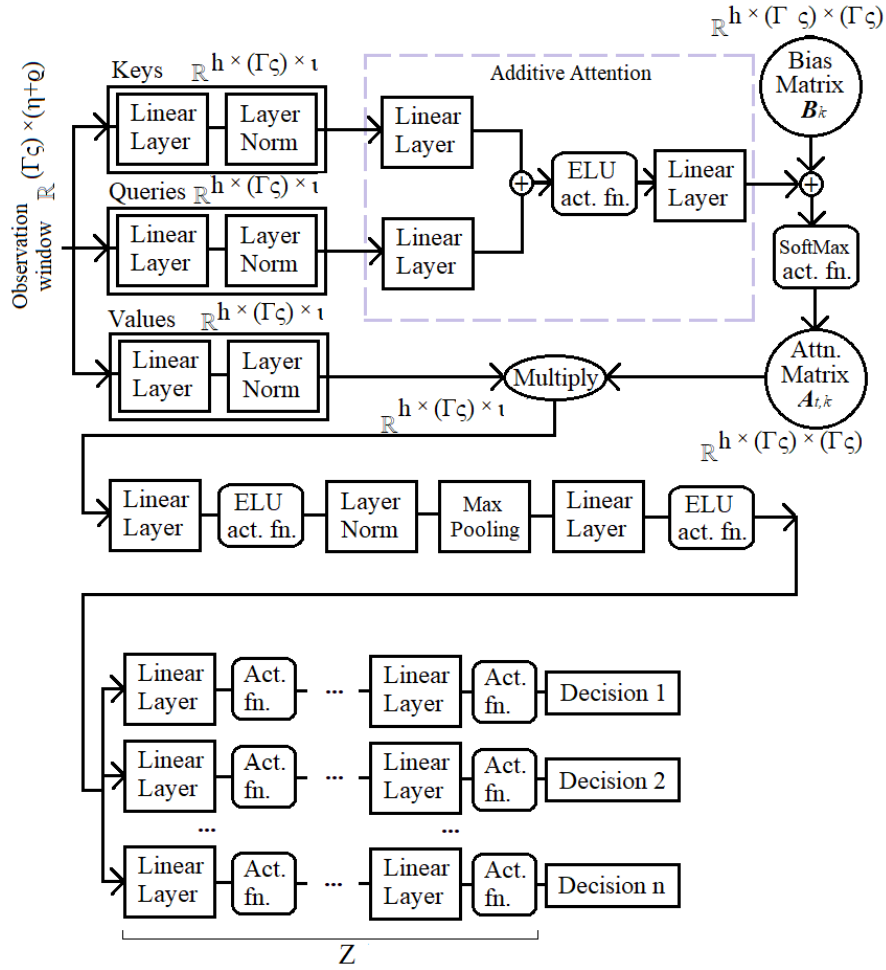


Figure 41 Diagram of the architecture from the actor

After $A_{t,k}$ is passed through the SoftMax function, it is then multiplied by the *Values* projection (third projection of the observation window in Fig. 41), and then passed through a linear layer and a normalization layer. A feature extraction operation called Max Pooling is used, in which the largest value from a region in the matrix is chosen as a representation of that region. The matrix with dimension $h \times \Gamma\zeta \times \iota$ is reduced into h vectors of features with dimension ι that hold the max values from each feature. Moreover, the vector is passed through a linear layer and an ELU activation function. The number of outputs from the SAM is equal to the number of heads; each output is a vector with a length determined by the user, a default value of 100 can be set to provide enough expressivity to the output tensor. For the last portion of the architecture, a set of independent MLP is used to process each decision n required in the environment. Each MLP uses

Z linear layers to process the outputs from the SAM. This final portion of the architecture (i.e., the set of MLPs) is common to the architecture presented in the *RNN approach*.

The architecture of the critic is essentially the same as the actor in the sense that it uses a SAM. The critic is the part of the agent that makes estimations from the current state of the environment for which the agent will make the next decision. This value is a prediction of the accumulation of the rewards into the future if the agent follows the current policy. The SAM from the critic can have multiple heads as the actor, this produces several outputs from the critic equal to the number of heads, which are then passed through a set of K layers (as in the critic from the *RNN approach*) and then outputs one value. Since the value network is also built with a SAM, then $\mathbf{A}_{t,k}$ and \mathbf{B}_k are also generated for every head.

The architecture of the attention module can vary in the number of neurons at each layer in the model. In this work, a default value of 100 neurons per layer is recommended based on preliminary tests using different scheduling instances. Also, the number of attention heads can vary depending on the complexity of the environment. A default value of $h = 3$ is recommended and a posterior analysis of their results should be carried out to increase or reduce this number as needed.

As per the descriptions provided above, the SAM is a more complex entity compared to the set of the *RNN approach*. This complexity is reflected in the evident number of trainable parameters which is larger for the SAM, but this is compensated with the features not included in the RNNs structure [30]. First, the capacity to handle larger sequences than the RNNs due to the attention feature that is calculated for each node in the input matrix. Second, $\mathbf{A}_{t,k}$ brings interpretability to the logic of the actor and the value function, which is not provided by the RNNs. Third, \mathbf{B}_k provides insights into the most relevant nodes in the environment. Lastly, the difference in training with the RNNs where a sequential update of the model is required in contrast to a parallel training that can be performed with the SAM. This last feature is related to the fact that all the information from the environment is provided in one input, while the RNN requires this input to be separated into independent time-inputs.

Overall, the use of SAMs provides advantages to the scheduling process in terms of interpretability and efficiency. The architecture allows the analysis of the spatial and temporal context of the plant through the attention which is an advantage that is not provided with RNNs. Also, the architecture allows the use of graph representations (complete or disjunctive) which can be related to the architecture of the STN.

6.2.5 Attention Matrix for Interpretability

The total number of rows and columns in every head of $\mathbf{A}_{t,k}$ and \mathbf{B}_k is dependent on the nodes ζ from the graph and the number of time-intervals Γ in the observation window. Since the size of the graph grows exponentially with the increase of these values, larger computational resources are needed for larger environments, which is a limitation of this method. For each pair of nodes in the graph representation of the environment, there is an attention value that is generated, which makes the number of values considerably large when building an interpretation. The techniques to process and interpret the values from the matrices are presented in this section. Moreover, a simple STN and the corresponding translation into a graph representation are presented in Appendix C.1. For such example, the construction of $\mathbf{A}_{t,k}$ and \mathbf{B}_k considers the number of nodes and the length Γ of the observation window \mathbf{O}_t .

The size of $\mathbf{A}_{t,k}$ and \mathbf{B}_k is considerably large as they provide an attention value for every existing connection between the nodes in \mathbf{O}_t . To reduce the complexity of the $\mathbf{A}_{t,k}$, calculating the average attention that each node is gathering from all the nodes is a convenient way to measure the relevance of such node. Eq. 53 shows the calculation of the element $Y_{t,j,k}$ from the mean-attention matrix $\mathbf{Y}_{t,k} \in \mathbb{R}^{\Gamma\zeta \times h}$; which contains the average values from each head of the agent, described as in Appendix C.1 (see Fig. C1).

$$Y_{t,j,k} = \frac{1}{\Gamma\zeta} \sum_{i=0}^i a_{t,t',i,j} \quad \forall j, t, t' \quad (53)$$

where the $\Gamma\zeta$ is the total number of attention values for the node, the subindices for the attention value (t, t', i, j) define the time intervals and the nodes that are been compared (see example in Appendix C.1 for a full explanation). Note that index k refers to a specific training and is omitted in the indexing of $a_{t,t',i,j}$ for the sake of brevity and clarity. The matrix $\mathbf{Y}_{t,k}$ lists the attention that is provided to each node in the

graph at multiple time-intervals at each head from the agent. Then, both matrices from the actor and the critic can be analyzed to see the focal points from the agent. While the actor provides the nodes for making a decision, the critic observes the nodes that are useful to predict the cumulative reward. Another way to describe these attention values is that the actor observes the nodes that influence the immediate next time-interval while the critic observes those that will have some effect in the future. This feature can be extended to ensure attention of the agent to nodes with features related to the tasks that are subject to uncertainty or that are prone to cause a constraint violation.

6.2.6 Attention for Uncertain Parameters

SAMs can be particularly useful for environments that are subject to uncertainty because the decisions of the agent can be validated through the interpretation provided by $\mathbf{A}_{t,k}$ and \mathbf{B}_k . It is expected that the agent will pay special attention to the factors that will ultimately affect the schedule and make its decisions based on them. A simple method for gathering and analyzing this information is presented in this section.

$\mathbf{A}_{t,k}$ provides the specific nodes at which the agent was focused on every decision. As mentioned before, this matrix provides a detailed node-to-node specification of the attention which needs to be reduced. The mean-attention (Eq. 53) should be calculated to know the amount of attention that each node gathered from the agent. Since the number of nodes can be considerable in large environments, it is suggested to sort the mean attention values in descending order and consider only the top nodes for the attention analysis. This analysis should be done for each head.

The analysis of the parametric uncertainty through the decisions of the agent helps to validate the tasks and nodes that the agent is considering for making the next action. Nevertheless, the influence of the stochasticity that is inherent in the training for DRL methods must be considered. Although the agent's logic can be interpreted, $\mathbf{A}_{t,k}$ and \mathbf{B}_k still behold the characteristic of a *black box*, i.e., the logic of this attention is not based in physical or conservation laws but by the reduction of the error in the prediction of the transition model from the environment. Then, describing the correct dynamics that the agent uses in its policy is still a challenge.

6.2.7 Bias Attention Matrix for Non-Stationary Environments

Non-stationary environments have a continuously changing system of rewards due to external factors; in the chemical industry, this can be exemplified with fluctuation in prices, demands or material availability. These environments require a retraining of the agent such that it can re-learn the new features and adjust its decisions such that they stay aligned with the updated reward function. Changes in the reward ultimately result in changes in the final production. The constant change in the conditions of a process is a common situation in real-world multi-product batch plants. \mathbf{B}_k is useful for non-stationary environments as it provides a general perspective of the focal points from the agent in the environment. In other words, it does not provide information from one time-interval as $\mathbf{A}_{t,k}$ but from the whole graph at any time. Overall, this method consists of retraining the agent as the environment changes and use \mathbf{B}_k as a way to observe the drift in the attention of the agent at each training k .

A bias matrix is added to every head in the SAM for both the actor and the critic; there is the possibility to produce only one matrix for the actor and one for the critic which is equally applied to every head; nevertheless, this would restrict the expressivity of the bias matrix. The same process to analyze $\mathbf{A}_{t,k}$ (i.e., through the mean values) shown in Eq. 53 is used for observing the bias values; Eq. 54 shows the calculation of the elements $\Psi_{j,k} \in \mathbb{R}^h$ from the matrix $\Psi_k \in \mathbb{R}^{\Gamma_C \times h}$ of mean values calculated from \mathbf{B}_k . Contrary to the $\mathbf{A}_{t,k}$, which is state-dependent, the information that provides \mathbf{B}_k is fixed at every time-interval t , providing a general idea of the nodes that are relevant during the training k .

$$\Psi_{j,k} = \frac{1}{\Gamma_C} \sum_{i=0}^i a_{t,tri,j} \quad \forall j, t, t' \quad (54)$$

Due to the stochasticity of the training process, tracking the changes in \mathbf{B}_k needs to be done with policies that share the same fundamental knowledge of the process. An agent that is trained for the first time in an environment will develop its own logic to achieve the highest possible reward, if a different agent is trained in the same environment, its logic will be different for achieving the same rewards. To maintain the logic from the initial policy in the environments with a new system of rewards, a transfer learning

method is proposed to handle the reward adaptation. This consists of setting a warm start for the next training by transferring the parameters from the previous policy into the next agent. Nevertheless, transferring the complete architecture might result in a rigid agent that will struggle to learn new aspects in the operation, e.g., the addition of a new product or equipment. Experiments with this approach showed that the agents would maintain the same path of decisions across environments with different rewards.

To overcome the fixation of the policy and allow the agent to learn new strategies, the transfer of parameters is performed only on the SAM, and does not include the hybrid part of the agent (represented as the Z components in Fig. 41). The parameters from Z are randomly initialized at the beginning of the re-training of every agent. Additionally, to enhance this strategy to reduce fixation, the parameters from \mathbf{B}_k from the transferred policy are reduced from their previous value by applying a user defined scaling factor g to the \mathbf{B}_k bounded between $[0, 0.9]$, i.e.,

$$\mathbf{B}_k = g(\mathbf{B}_{k-1}) \quad (55)$$

where k represents the current training of the agent. The limited transfer of parameters to the policy of the new agent allows it to learn a new strategy while holding fundamental knowledge from the previous agent. At the same time, the attention of the multiple heads from the SAM is maintained and keeps their focus on the same features. The transfer learning technique assumes that the architecture of the agent remains the same across the evolution of the environment. On the other hand, the system of rewards in the environment is expected to change. In cases where there is a change in the architecture of the agent or the plant instance, it is possible to do transfer learning from specific layers of the agent, as they can provide fundamental insights from the dynamics of the environment [125], although this translation might not be as effective as in the case where the agent remains the same. Note that a change in the structure of the agent might be needed due to changes in the inputs, the outputs, or the available computational resources.

A contribution from \mathbf{B}_k (which is also extended to the $\mathbf{A}_{t,k}$) is the insight that they provide about elements from the graph representation that are not directly connected to the rewards. For example, the storage tanks, the feeds and the production are not sources of rewards for the agent but only sources of

material for running the task. The attention values can then be used as descriptors of specific aspects of the process and their role in the decision-making process. For instance, nodes that can compromise the safety or operational constraints of the process or the quality of the product.

6.3. Results

The hybrid agent with SAM, presented in the previous section has been tested using a modified version of the plant presented in Chapter 4 [48]. This case study is used to capture the uncertainty in the attention matrix, and show the monitored transfer learning method that uses the bias matrix. This plant is used to compare the performance of the agent built with SAMs and the agent built with RNNs from the *RNN approach* [18]. Details of this instance are presented in Appendix C.2. That agent was tested in a) the deterministic version of the plant and b) the plant subject to aleatoric uncertainty. As shown in Appendix C.2.3, the results from that agent built with SAM exhibited a more stable training and a better understanding of the plant dynamics, reflected in a considerable reduction in the number of constraint violations. Moreover, the agent also showed minimal interference even when the system was subject to uncertainty, thus demonstrating a better capacity to learn the dynamics of the system compared to the agent presented in our previous work (*RNN approach*). More details about this comparison are presented in Appendix C.2.3.

The agents presented in this Chapter were trained for a total of 5,000 episodes on a CPU intel i9-14900K, 3200 MHz, 24 cores, and 64 GB of RAM, taking on average 5.04 h per training. The DRL method was developed in Python 3.11.3 and PyTorch version 2.1.0. The main libraries used to perform these experiments include Numpy 1.21.6, Gym 0.26.2, and Pandas 0.24.2.

6.3.1 Case Study

The plant configuration of this case is presented in Fig. 42. The number of tasks and states remain the same as in the original plant, but the path for producing products P1 and P2 (in states E and I, respectively) has been modified. In contrast to the original plant, the two paths have no dependency on one another: P1 follows the path: Task 3 \rightarrow Task 1 \rightarrow Task 2, while P2 follows the path: Task 1 \rightarrow Task 4 \rightarrow Task 5. Moreover, the feeds (States A, B, and C) in this case study are limited to a fixed amount. It is then expected

that the agent will dedicate part of its attention to the available material to produce P1 and P2. The same capacities from the original plant were used in this case (displayed in Table B2 in the Appendix). An observation window \mathbf{O}_t from this plant was made with four consecutive time intervals ($\Gamma = 4$). The information collected at each time interval (i.e., Ξ_t) is presented in Table C3.

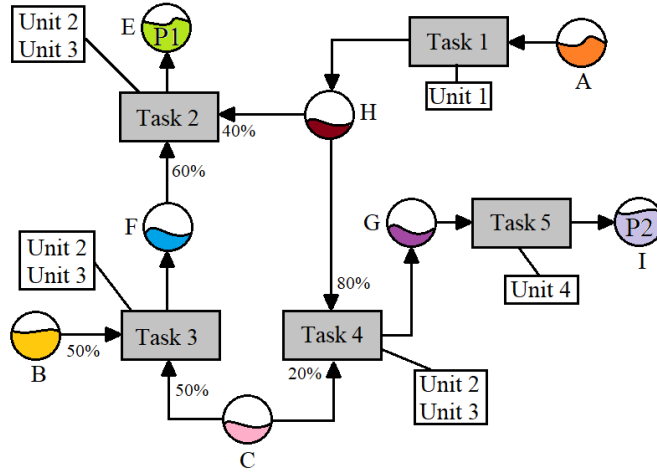


Figure 42 Plant for Case Study.

Note that the configuration of the plant instance is different from the original plant, but the optimization problem definition, the action space and the rewards remain the same as in the *RNN approach*.

The design of the next experiment aims to show how the attention of the agent drifts when the system of rewards changes in non-stationary environments. This change can be due to external changes in the process; for instance, changes in prices, priority, or demands of the products. The plant has two paths to produce P1 (state E) and P2 (state I) and they use, at some point in the process, the intermediate product from Task 1. Larger plants have more complex interconnections in which a change in rewards might result in major changes in the behavior of the agent. Then, the use of the method from Section 6.2.7 for tracking the changes in these environments becomes a useful tool for interpretable DRL.

6.3.1.1 Interpretation of the Bias Attention Matrix

To demonstrate the capacity of the bias matrix (\mathbf{B}_k) to track the changes in the policy, the rewards from the environment are subject to several changes and one training k is performed on each scenario. The

change consists in the progressive increase in the rewards to the tasks related to the production of P2. Attention to specific nodes related to the production of P2 is recorded during 5 experiments. In the first experiment ($k = 1$), both processes have an equal reward per unit of product. Then, the rewards for the tasks related to P2 are increased progressively until the last experiment ($k = 5$) where a substantial reward is given to these tasks in comparison to those related to P1. The aim is to record the shift in attention in parallel to the increase of rewards to P2.

The tasks associated with the production of P1 are Tasks 1, 2, and 3, while the related states are B, C, F, H, and E (see Fig. 42). On the other hand, P2 is related to Tasks 1, 4, and 5, and the states A, C, H, G, and I. Five experiments, presented in Table 10, were designed with incremental rewards of 20% to the tasks associated with P2 (except Task1). This increment was chosen as it allows to observe the gradual shift in the attention from the agent, which proves the sensitivity of the interpretation performed with the SAM. Each experiment corresponds to a specific training k . The first experiment sets the same amount of reward to the two paths in the process, aiming a balanced production of P1 and P2. Table 10 shows the rewards given to the tasks per unit produced for both parts of the process. For instance, if Task 1 is completed at its full capacity (which is 100 units), then the reward will be 0.3 (i.e., $3e-3*100$).

Table 10 List of rewards for the experiments (first 5 rows) and production from each training (last 2 rows). Tasks 4 and 5 are progressively incremented in intervals of 20%.

	Experiment 1 $k = 1$ Balanced	Experiment 2 $k = 2$ +20%	Experiment 3 $k = 3$ +40%	Experiment 4 $k = 4$ +60%	Experiment 5 $k = 5$ +80%
Task 1	3e-3	3e-3	3e-3	3e-3	3e-3
Task 2	0.03375	0.03375	0.03375	0.03375	0.03375
Task 3	0.10125	0.10125	0.10125	0.10125	0.10125
Task 4	0.03375	0.0405	0.04752	0.054	0.06075
Task 5	0.10125	0.1215	0.14175	0.162	0.18225
P1	309	263	236	138	0
P2	75	98	116	168	238

The reward values for Experiment 1 in Table 10 are based on the rewards from Case Study 1, which were defined through experimentation. In this experiment, the values for Task 2 and 4 are the same; the rewards for Tasks 3 and 5 are also the same but they are higher than Tasks 2 and 4. This is because Tasks

3 and 5 are the consecutive tasks to 2 and 4, respectively; hence, the increment is added to motivate the agent to continue with them and avoid getting stuck in the first tasks.

An agent was trained for each experiment presented in Table 10, the production of P1 and P2 is presented in the last two rows; the increase in production of P2 validates the increase of rewards. Naturally, this increment in production does not follow the 20% increment in the rewards. Moreover, in Experiment 1, the agent has a clear preference to produce P1 although the rewards were equal for the paths of both products. This was expected as other factors are involved in production, like the total processing time, the requirements in raw material, and the available raw material to produce each product, which is limited. In the last column with the 80% increment, the agent finds it more rewarding to produce P2 over P1.

As mentioned before, the training of the agents was performed following the method presented in Section 6.2.7. The agents were initialized with the policy from the previous experiment, i.e., $k - 1$ (except for Experiment 1), i.e., Experiment 2 used the trained policy from Experiment 1; Experiment 3 was initialized with the trained policy from Experiment 2, and so on. The \mathbf{B}_{k-1} from the previous policy was scaled down (by a factor of $g = 0.7$) to become more flexible during the training with the new system of rewards. In this way, the transfer learning from one agent to the other was performed and the logic remained from one agent to the next one. In Experiment 1 the policy was initialized from a random set of parameters. For all the cases, the hybrid part of the agent (i.e., section Z in Fig. 41) was randomly initialized so this section of the network would be developed by each agent.

Another set of experiments was run with the rewards from Table 10 but in this case all the agents were initialized with random parameters, i.e., no transfer learning was performed. These experiments showed that the agents would develop a similar set of results (i.e., with an increasing preference in P2) but with different logic. In other words, each agent developed its attention heads focusing on different aspects of the process. This situation was not convenient to track the evolution of the attention in the experiments and was used to support the use of transfer learning. When applying the transfer learning, the logic is maintained and the agents maintain focus on specific nodes, as it will be shown next.

The matrix Ψ_k , with the mean attention values of the three attention heads (obtained using Eq. 54) was generated for each experiment in Table 10. Since it is intended to analyze the attention on the production of P2, the evolution of the mean attention values over specific nodes related to this product were gathered. Fig. 43 shows the information collected from the three heads of the agent. Specifically, the evolution of the attention in Tasks 4 and Task 5 across the trainings presented in Table 10 (from $k = 1$ to $k = 5$). Since Task 4 can take place either in Reactor 1 or Reactor 2, both nodes are included in the analysis. Each plot shows four lines, corresponding to each of the time-intervals that are registered in the observation window \mathbf{O}_t : T1, T2, T3, and T4, being the latter the present time-interval (see Fig. 40). Each line in the plots shows the mean attention values from the node for each of the experiments (reward increments are indicated as percentages in the horizontal axis).

As the mean-attention values from Fig. 43a-f show, Heads 1 and 2 are strongly dedicated to Task 4 in Reactor 1, which is the largest of the two reactors. Meanwhile, the attention values from Fig. 43i in Head 3 show that this head is more focused on Task 5. The growing tendency in the attention values from most of the plots follows the increment in the rewards; this tendency is visible due to the use of the transfer of “knowledge” from policy to policy. An exception to this behavior can be seen in Fig. 43h from Head 3, where there was a drastic change in the attention of the elements. The other two plots from this Head (Fig. 43g and 43i) show a consistent increment in the attention provided to Tasks 4 in Reactor 1 and Task 5. The same order of attention in the time-intervals is kept with the increment of rewards in most of the plots, which shows a consistent evolution of the attention values. Exceptions as in Fig. 43b show a drift in the attention towards specific time-intervals. Fig. 43d on the other hand, keeps the same order of attention in the experiments of 40%, 60%, and 80%: T4, T3, T2, T1.

The reduction in the attention to a Task in a plot shows a drift in the attention from the agent to other aspects of the process; for instance, Fig 43c shows a decrease in the attention that is translated to Task 4 (Fig. 43a and 43b). This is an interesting outcome considering that Task 5 is the task with the largest reward in the process. This shows that the agent found that prioritizing Task 4 instead of Task 5 would provide a larger production in the long term.

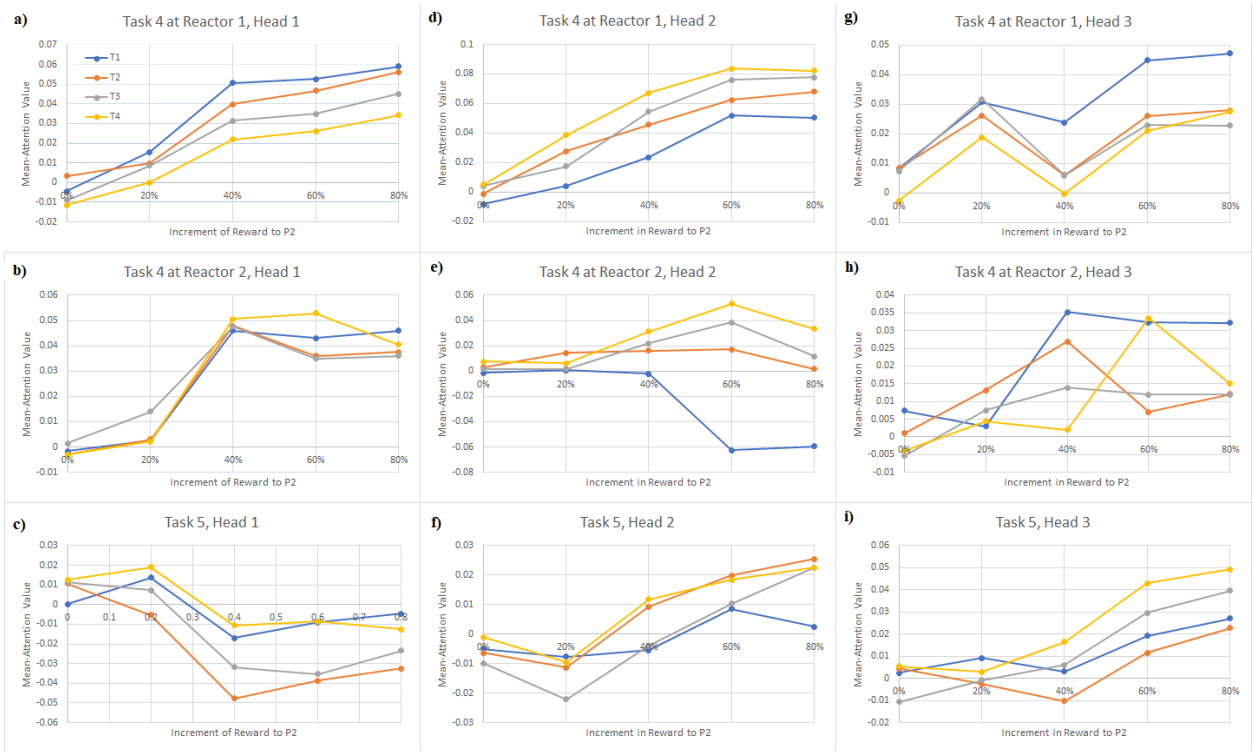


Figure 43 Evolution of the attention values to specific nodes in each of the three heads. Each of the lines in the plots show the attention to the node in one of the four time-intervals registered in the observation window, where T1 is the time-interval at $t-3$, T2 represents $t-2$, T3 represents $t-1$, and T4 is the present time-interval t .

The attention in the storage nodes is also visible through the bias matrix. Fig. 44 shows the attention of States F, G, and H in Head 1. The attention dedicated to products related to P2 (State G) increased with an increase in rewards, while state H, which holds the output from Task 1, received lower values. Although State F, which is related to the production of P1, also reported a continuous increase, it remains below the values of State G.

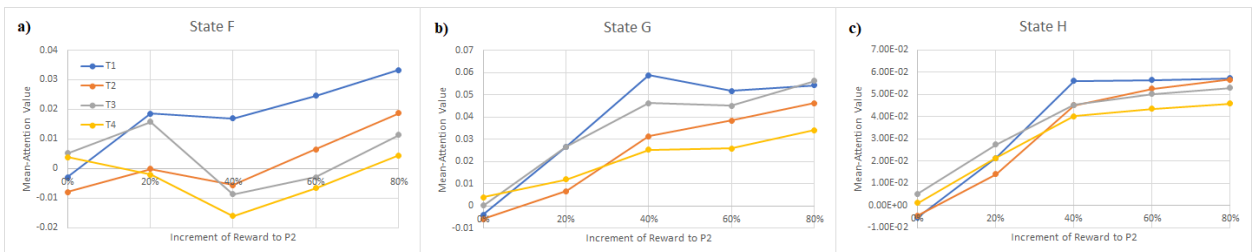


Figure 44 Attention values for states F, G, and H from Head 1.

The attention parameters from the bias matrices did not share a proportional relation with the change of rewards in the system, which was an expected outcome from this experiment. The drift in the attention does not necessarily need to be equal in magnitude to the change in rewards. This is probably due to the stochastic nature of the training processes. Nevertheless, the attention and the change in rewards from the environment can be related qualitatively, and the parameters from the bias matrix are useful to see where the general attention is in the process. This information cannot be gathered from the resulting schedule or from the rewards the agent achieves, hence the significance of the SAM in the interpretability of the agent's perception of the environment.

6.3.1.2 Attention Matrix for Case Under Uncertainty

In this section, the method presented in Section 6.2.5 is assessed. The attention matrix $\mathbf{A}_{t,k}$ generated at several time-intervals during the scheduling process are analyzed under the assumption that Task 3 is subject to epistemic uncertainty. This type of uncertainty arises from a lack of knowledge about the system and can be reduced by acquiring more information. The rewards from Experiment 3 ($k = 3$) presented in Table 10 were used since it was observed in the previous section that the mean-attention values were high for tasks related to both products. In other words, the overall attention observed in Ψ_3 with this set of rewards was balanced between the tasks that produced P1 and those that produced P2.

Two agents were trained: the first agent is trained in an environment without uncertainty and the second is trained with uncertainty. The agents are trained and the attention matrices at specific time-intervals are collected from each agent and then compared with each other. Since the horizon is composed of 31 time-intervals, this is the number of matrices trios (as there are three heads) that are generated in the scheduling process. For the sake of brevity, only the attention matrices in the time-intervals when the agent chose Task 3 were gathered to be compared; for both cases this time was $t = 2$. The length for each head in $\mathbf{Y}_{2,3}$ is of 64 values because $\Gamma = 4$ and $\zeta = 16$ (see Fig. 40); as mentioned before, not all of them are relevant for the agent because the level of attention is considerable only for 5 to 10 nodes. For comparison purposes, and for the sake of brevity, only the five largest values are gathered and compared.

The uncertainty in the processing time from Task 3 is modelled with a Gaussian distribution where the mean value is defined as a function of the amount of material available in State B, which is used for this task. It is assumed that the consumption of material from State B generates wear and tear on the operating units, which only affects the processing time of Task 3. The mean value from the distribution is described in Eq. 56: a minimum value of $\mu = 3.3$ is used as a baseline as this value allowed a well-distributed probability; otherwise, the processing times tend to concentrate around a specific value. The initial amount of material in State B is 200 units and B_t is the current amount of material in the state. The Gaussian distribution $G_t(\mu_t, \sigma)$ has a fixed standard deviation $\sigma = 1$, and every sample is rounded to the nearest integer value and clipped in between (3, 5). Consider that for the case without uncertainty, the processing time of Task 3 consists of 4 time-intervals.

$$\mu_t = 3.3 + \left(\frac{2}{200}\right)(200 - B_t) \quad (56)$$

The training of each agent was performed with the same length of training and hyperparameters listed on Table B1, being the addition of uncertainty into the environment the only difference. When the training was completed, both agents were used to generate a complete schedule in their respective environments. In both schedules, Task 3 at Reactor 1 is called by the agent to be executed in the environment in several time-intervals. The attention matrices $\mathbf{A}_{t,k}$ from one of those time-intervals were subtracted from the actor and processed to observe the focal points from the agent when making such decisions. The matrix $\mathbf{Y}_{t,k}$ was calculated and the values from each head were disposed from highest to lowest; Table 11 shows the top five nodes that received the most attention in each of the three heads. As shown in this Table, the agent did not attend the node corresponding to this task, presumably due to the deterministic nature of the environment. Instead, the attention was mainly oriented towards the states A and G at time T4. These states are related to the Feed for Task 1 (A) and to the output state for Task 4. All are observed at time 4, which refers to the present time-interval while Time 1 references the time-interval which is 4 steps into the past.

Table 11 Highest mean-attention values from the three heads from the agent in a deterministic environment.

Environment without uncertainty, Values from $\mathbf{Y}_{2,3}$

Node	Attn	Node	Attn	Node	Attn
A at Time 4	0.2731	G at T4	0.202	G at Time 4	0.1167
A at Time 1	0.0982	Task 4 at Reactor 2 at Time 1	0.0499	Task 1 at Time 1	0.0958
B at Time 4	0.0636	Task 1 at Time 4	0.0493	C at Time 4	0.0649
Task 3 at Reactor 2 at Time 3	0.0522	Task 5 at Time 2	0.0388	H at Time 4	0.0595
Task 1 at Time 3	0.0507	Task 5 at Time 1	0.0372	Task 4 at Reactor 2 at Time 4	0.0424

The attention values in the first row show the largest mean-attention value from each Head, these values decrease considerably after the first rows as can be seen in Heads 1 and 2. Considering that there are 59 remaining values (64 nodes in total), it can be confirmed that the attention from each head is focused on the first nodes from the list.

The attention values from the agent trained in the environment with uncertainty showed a drift towards the task with uncertain processing time when deciding to activate such action. Table 12 shows that the agent prioritized Task 3 at Reactor 1 at times 1 and 2 in the observation window for Heads 1 and 2 (see highlighted cells in Table 12). Meanwhile, Head 3 was focusing on Tasks 1 and 2, which are the following steps in the path for producing P1 and they utilize the output product from Task 3. A similar pattern to that shown in this Table was observed in the other time-intervals in which the agent chose to execute Task 3 at Reactor 1. This showed that the agent performs the same analysis in the environment when this action was executed.

Table 12 Highest mean-attention values from the three heads from the agent in environment with uncertainty.

Environment with uncertainty, Values from $\mathbf{Y}_{2,3}$					
Node	Attn	Node	Attn	Node	Attn
Task 1 at Time 4	0.0509	G at Time 4	0.1595	E at Time 4	0.0767
G at Time 3	0.0406	Task 3 at Reactor 1 at Time 3	0.1239	Task 5 at Time 3	0.0756
Task 5 at Time 4	0.0399	E at Time 1	0.0867	E at Time 4	0.0595
Task 3 at Reactor 1 at Time 3	0.0371	G at Time 2	0.0578	Task 4 at Reactor 2 at Time 3	0.0489
I at Time 1	0.0371	Task 3 at Reactor 1 at Time 2	0.0474	E at T2	0.0489

Although there was a consistent behavior from the agent emanated from this method, a quantitative way to correlate the uncertainty with the attention values obtained from the Heads cannot be clearly identified. This is due to the way the agent develops this logic, namely through a training process that includes stochastic decisions instead of physical or conservation principles. Nevertheless, a qualitative description of the decision process could be derived with the methods presented.

6.4. Summary

The attention and the bias parameters were used to gain insights from the decision-making process from the agent in the scheduling problem. The attention module showed to be more stable in the training process, reducing the number of episodes needed to reach a plateau in the learning curve. The violation of constraints was also reduced using this architecture although production was almost 30% less than that one reported from the RNNs framework presented in the previous chapter. A method to follow the drift in the attention when the system of rewards change is presented and evaluated, showing consistency in the reaction from the agent to the increase of rewards. The bias parameters revealed that the agent based its decisions on specific tasks that not necessarily provide the largest rewards. Also, the presented method allowed to analyze the attention in nodes from the environment that are not related with the acquisition of rewards, namely the storage tanks.

The attention matrix generated at each time-interval was evaluated for a case study under uncertainty. Two agents were trained in the same environment with and without uncertainty and their attention was compared in the Task with uncertainty in its processing times. In the case without uncertainty, the agent focused on the amount of material in the states and in some tasks, but no attention was oriented to the state of the Task that was subject to uncertainty. Presumably, because the agent had certainty about the processing time of this task. In the case with uncertainty, it was observed that the agent's attention was incremented towards such a task; two of the three heads of the actor were attending at some level during different time-intervals.

The interpretability through the attention matrices is a source of valuable information with potential to be used to track the evolution of the agent in environments with constant change of rewards. Their use for scheduling problems under epistemic and aleatoric uncertainty provides a tool to observe the actor's logics when making a decision. Although in real life implementation, the use of an agent still needs to be supervised by an expert, interpretability provides a way to confirm that the actions are considering key factors in the process, like uncertainty, safety (e.g., constraint satisfaction) and security. Future work includes the interpretation of the critic, which is in charge of the reward prediction in the long term. A method for interpretation for this part of the agent is intended to complement the actor's interpretability capacities.

Chapter 7 Conclusion and Future Work

In this thesis a set of methodologies aimed to train intelligent agents for approaching the scheduling problem under uncertainty were presented. These methods were developed based on the gaps in the literature that were outlined in Section 2. The experiments and results demonstrated the advantages that these methods could bring to the chemical engineering field, and in particular to chemical process scheduling. Also, multiple challenges were observed during the development and implementation of the DRL methods, showing the early stage of this technology in academia and the industry. The insights gained from this work are presented. This section presents, first, a discussion on the key features developed in this work, second, the advantages and challenges for the development of these methods, and third, a set of recommendations for future work.

7.1. Concluding Remarks

The methods developed in this thesis were centered on the translation of the scheduling problem into the elements of DRL, i.e., environment, policy, action space, and reward function. A list of the key contributions from the methodologies presented in this work to the field of scheduling under uncertainty is presented below. The contributions provide alternatives for the implementation of DRL methods, highlighting aspects where special attention is required.

- For the design of the environment, alternatives to handle invalid actions were presented. The use of a masking technique and the development of heuristic methods in the implementation was demonstrated. Also, different forms to train the agent on infeasible actions with penalties and cancellations were presented.
- Different alternatives for translating the information from the process to the agent through the state representation of the environment with partial observability and uncertainty were shown. These methods were applied to architectures with RNNs and Self-attention modules. For the latter, a method for translating the environment into a graph representation was presented.

- Methods for designing the system of rewards for the scheduling problem were proposed; dense systems became preferred over sparse systems of rewards as they became more expressive of the objective function and more adequate for the DRL algorithms used. Tuning this parameter represented a bottleneck in the translation of the scheduling optimization problem.
- Adapting the action space to the requirements of the scheduling process through the use of hybrid agents constituted another outcome of this thesis. While the usual approach observed in the literature consists of framing the action space in a discrete set of actions. The methodologies presented in Chapters 4-6 on the use of hybrid agents demonstrated that the development of agents with multiple decisions can be adequate for the scheduling optimization problem for chemical facilities.
- In hybrid decision spaces, independent exploration methods for each decision are required, this need was identified while developing an integration method of continuous and discrete actions spaces. The temperature annealing method and a multivariate normal distribution with a continuous decrease through the training were proposed.
- A method to gather information from the agent using the attention module was proposed. This includes the preprocess of the data for its interpretation at every time-interval of the process and in a general perspective (through the bias matrix). The attention module provided an alternative way to RNNs in the approach of the scheduling process as a POMDP.

Among the advantages of the policies developed in the presented methods, the following aspects are especially relevant for their application in scheduling problems under uncertainty:

- **Fast response:** Once it is trained, the policy can be used for online scheduling which provides immediate responses. This feature represents an advantage over reactive scheduling methods that require computing a new schedule to counteract the effects of the uncertainty realizations.
- **Scalability:** The methods presented in this work demonstrated the capacity to perform with environments of different scales. The results from these experiments showed that the agents had similar performance in terms of results and constraint violations. Moreover, some works reported satisfactory

performance of policies that were tested on new environments which are similar to those from the original training [84], [85], [86].

- **Transfer Learning:** Although the initial learning process of a policy is considerably long, the training on similar scenarios can be reduced by using pre-trained policies as starting points. In other words, in the case of an evolution of the environment in which a policy is no longer effective, a re-training of this policy can be performed by using it as the initial policy in a new training process. The original parameters are, at some level, a representation of the environment that can be recycled for new training which will be shorter in the number of episodes.
- **Process Integration:** Integration of different tasks are common in the literature of process system engineering as this is a relevant challenge in enterprise-wide optimization [43], [126], [127], [128], [129]. The capacity of DRL policies to input information without an apparent structure allows to build correlations between sources of data that are not clearly related. DRL methods and ANNs in general are an option for getting insight from complex processes without the need of mathematical model. Decision-making process for task integration can be approached through these learning processes.

Following is a list of the limitations that were major challenges for the methodologies that were developed in this thesis. Although these are inherent challenges for DRL methods, they were specially considered in their implementation for scheduling problems under uncertainty.

- **Lengthy trainings:** The development of effective policies for complex environments requires long trainings to ensure the exploration of the greatest number of states. Architectures like RNNs require a long training as they are unstable, this issue can be dimmed when extending the training, allowing a slower change in the exploration and updating hyperparameters.
- **Hyperparameter tuning:** Setting the hyperparameters for training a model is a problem-dependent task in which the characteristics of the environment such as complexity and scale have to be considered. Special attention is needed for the exploration techniques, the learning rate and the number of episodes per training, which are key in the development of experience and for the learning of the agent. Methods that

can be useful for these purposes include the Grid-search [130] method and Bayesian Optimization [12] for hyperparameter tuning, which were out of the scope of this work.

- **Suboptimal policy:** There is a trade-off between accuracy of the agent and the training time; ideally an infinite training would confirm that the environment is fully explored and that the policy is optimal. For scheduling optimization problems, a comparison with an optimizer can be performed as in the work from Chapter 4, but this required the problem to be modified as a deterministic problem, i.e., without uncertainty. Moreover, constraint violations were common in the trained policies which derived in the need of heuristics for correcting the decisions when it was necessary.
- **Interpretability:** Although an approach for gaining insight into the decisions of the agent was developed using attention modules, the agent remains as an entity that is not fully clear on its decisions. ANNs are considered as “black box” models which represents a challenge in terms of interpretation and could become a problem in the context of safety and security.

The challenges listed above are inherent of DRL methods and data-driven techniques in general. Multiple techniques are used to alleviate the impact of these issues in the final model but since they are part of the tools of these methods, they cannot be avoided completely. Nevertheless, the field of DRL is on an early stage of implementation on scheduling problems and the area of opportunity is large as new methods arise for developing new approaches. It is expected that DRL methods will continue to be adapted to the decision-making processes in the industrial scenario. Challenges that might be faced for their adoption in the industry include the availability of simulation environments from real world cases that can be used for training. The incorporation of available knowledge and experts’ decisions into the policy before the training will be key for reducing the training times. Moreover, an expert observer over the implementation of the system is required to approve the decisions of the agent before execution. States of the environment that were not considered in the training as well as potentially dangerous or unsafe situations might not be well handled by the agent without a proper training. Thus, the interpretability and continuous observation of the agent’s performance must be a relevant area of research in DRL for industrial applications.

7.2. Future Work

Future work in this area should be oriented to enhance the capacities of the DRL methods to produce better and more robust policies.

- The use of state-of-the-art exploration methods like intrinsic curiosity modules will be helpful for extending the exploration methods and motivate the agent to search for other strategies [131]. Curiosity modules come with the trade-off of more parameters to be trained but there have been promising in other scheduling applications [132]. To the author's knowledge, there is no research on this area for scheduling in State Task Networks under uncertainty.
- Additionally, the efficient use of computational resources should be a priority for the scalability of the presented methodologies. The use of multi-processing methods for scheduling problems should be extended in the literature, as this will allow an efficient collection of data from the environment.
- Furthermore, the integration of data-driven methods with current optimization methodologies has been growing in the literature as they pose an option to bring the best of both parts into the field of optimization [45]. The use of surrogate models to make parameter estimations results in a promising alternative for saving time in the optimization process. These estimations include the filtration of infeasible regions or the use of predictors.

References

- [1] Z. Li and M. Ierapetritou, "Process scheduling under uncertainty: Review and challenges," *Computers & Chemical Engineering*, vol. 32, no. 4–5, Art. no. 4–5, Apr. 2008, doi: 10.1016/j.compchemeng.2007.03.001.
- [2] L. K. Church and R. Uzsoy, "Analysis of periodic and event-driven rescheduling policies in dynamic shops," *International Journal of Computer Integrated Manufacturing*, vol. 5, no. 3, Art. no. 3, May 1992, doi: 10.1080/09511929208944524.
- [3] I. Sabuncuoglu and M. Bayız, "Analysis of reactive scheduling problems in a job shop environment," *European Journal of Operational Research*, vol. 126, no. 3, Art. no. 3, Nov. 2000, doi: 10.1016/S0377-2217(99)00311-2.
- [4] P. M. Verderame, J. A. Elia, J. Li, and C. A. Floudas, "Planning and Scheduling under Uncertainty: A Review Across Multiple Sectors," *Ind. Eng. Chem. Res.*, vol. 49, no. 9, pp. 3993–4017, May 2010, doi: 10.1021/ie902009k.
- [5] J. Balasubramanian and I. E. Grossmann, "Scheduling optimization under uncertainty—an alternative approach," *Computers & Chemical Engineering*, vol. 27, no. 4, Art. no. 4, Apr. 2003, doi: 10.1016/S0098-1354(02)00221-1.
- [6] V. Venkatasubramanian, "The promise of artificial intelligence in chemical engineering: Is it here, finally?," *AIChE J*, vol. 65, no. 2, Art. no. 2, Feb. 2019, doi: 10.1002/aic.16489.
- [7] S. Reynoso-Donzelli and L. A. Ricardez-Sandoval, "A reinforcement learning approach with masked agents for chemical process flowsheet design," *AIChE Journal*, vol. n/a, no. n/a, p. e18584, doi: 10.1002/aic.18584.
- [8] T. A. Mendiola-Rodriguez and L. A. Ricardez-Sandoval, "Robust control for anaerobic digestion systems of Tequila vinasses under uncertainty: A Deep Deterministic Policy Gradient Algorithm," *Digital Chemical Engineering*, vol. 3, p. 100023, Jun. 2022, doi: 10.1016/j.dche.2022.100023.
- [9] M. Mowbray, D. Zhang, and E. A. D. R. Chanona, "Distributional Reinforcement Learning for Scheduling of Chemical Production Processes," Mar. 09, 2022, *arXiv*: arXiv:2203.00636. Accessed: Jun. 22, 2022. [Online]. Available: <http://arxiv.org/abs/2203.00636>
- [10] A. Baghban, P. M. Castro, and F. Oliveira, "Data-driven robust optimization for pipeline scheduling under flow rate uncertainty," *Computers & Chemical Engineering*, vol. 193, p. 108924, Feb. 2025, doi: 10.1016/j.compchemeng.2024.108924.
- [11] N. Triantafyllou *et al.*, "Machine learning-based decomposition for complex supply chains," in *Computer Aided Chemical Engineering*, vol. 52, A. C. Kokossis, M. C. Georgiadis, and E. Pistikopoulos, Eds., in 33 European Symposium on Computer Aided Process Engineering, vol. 52., Elsevier, 2023, pp. 1655–1660. doi: 10.1016/B978-0-443-15274-0.50263-8.
- [12] P. I. Frazier, "A Tutorial on Bayesian Optimization," Jul. 08, 2018, *arXiv*: arXiv:1807.02811. doi: 10.48550/arXiv.1807.02811.
- [13] S. Reynoso-Donzelli and L. A. Ricardez-Sandoval, "An integrated reinforcement learning framework for simultaneous generation, design, and control of chemical process flowsheets," *Computers & Chemical Engineering*, vol. 194, p. 108988, Mar. 2025, doi: 10.1016/j.compchemeng.2024.108988.
- [14] T. A. Mendiola-Rodriguez and L. A. Ricardez-Sandoval, "Integration of design and control for renewable energy systems with an application to anaerobic digestion: A deep deterministic policy gradient framework," *Energy*, vol. 274, p. 127212, Jul. 2023, doi: 10.1016/j.energy.2023.127212.
- [15] D. Rangel-Martinez and L. Ricardez-Sandoval, "Application of Reinforcement Learning with Recurrent Neural Networks for Optimal Scheduling of Flow-Shop Systems Under Uncertainty," in *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*, Rome, Italy: IEEE, Jul. 2023, pp. 1–6. doi: 10.1109/CoDIT58514.2023.10284480.
- [16] D. Rangel-Martinez and L. A. Ricardez-Sandoval, "A recurrent reinforcement learning strategy for optimal scheduling of partially observable job-shop and flow-shop batch chemical plants under

- uncertainty,” *Computers & Chemical Engineering*, vol. 188, p. 108748, Sep. 2024, doi: 10.1016/j.compchemeng.2024.108748.
- [17] D. R. Martinez and L. Ricardez-Sandoval, “Scheduling of State Task Network Under Uncertainty Via a Hybrid Reinforcement Learning Agent with Partial Observability,” presented at the 2024 AIChE Annual Meeting, AIChE, Oct. 2024. Accessed: Jan. 09, 2025. [Online]. Available: <https://aiche.confex.com/aiche/2024/meetingapp.cgi/Paper/687347>
- [18] D. Rangel-Martinez and L. A. Ricardez-Sandoval, “Recurrent Reinforcement Learning Strategy with a Parameterized Agent for Online Scheduling of a State Task Network Under Uncertainty,” *Ind. Eng. Chem. Res.*, Mar. 2025, doi: 10.1021/acs.iecr.4c04900.
- [19] D. Rangel-Martinez and L. A. Ricardez-Sandoval, “Hybrid Deep Reinforcement Learning Agent for Online Scheduling and Control for Chemical Batch Plants,” in *14th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems*, Bratislava, Slovakia, Jun. 2025.
- [20] D. Rangel-Martinez and L. Ricardez-Sandoval, “Interpretable Online Scheduling for Chemical Batch Plants with Attention Augmented Reinforcement Learning Agents”, doi: 10.2139/ssrn.5272328.
- [21] R. S. Sutton and A. G. Barto, *Reinforcement Learning, second edition: An Introduction*. MIT Press, 2018.
- [22] C. P. Andriotis and K. G. Papakonstantinou, “Managing engineering systems with large state and action spaces through deep reinforcement learning,” *Reliability Engineering & System Safety*, vol. 191, p. 106483, Nov. 2019, doi: 10.1016/j.res.2019.04.036.
- [23] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” Jan. 30, 2017, *arXiv: arXiv:1412.6980*. doi: 10.48550/arXiv.1412.6980.
- [24] D. Brezak, T. Bacek, D. Majetic, J. Kasac, and B. Novakovic, “A comparison of feed-forward and recurrent neural networks in time series forecasting,” in *2012 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*, Mar. 2012, pp. 1–6. doi: 10.1109/CIFEr.2012.6327793.
- [25] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, “Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review,” *Int. J. Autom. Comput.*, vol. 14, no. 5, pp. 503–519, Oct. 2017, doi: 10.1007/s11633-017-1054-2.
- [26] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*, First edition. Beijing ; Boston: O’Reilly Media, 2017.
- [27] S.-H. Noh, “Analysis of Gradient Vanishing of RNNs and Performance Comparison,” *Information*, vol. 12, no. 11, Art. no. 11, Nov. 2021, doi: 10.3390/info12110442.
- [28] D. Hu, “An Introductory Survey on Attention Mechanisms in NLP Problems,” in *Intelligent Systems and Applications*, Y. Bi, R. Bhatia, and S. Kapoor, Eds., Cham: Springer International Publishing, 2020, pp. 432–448. doi: 10.1007/978-3-030-29513-4_31.
- [29] V. Zambaldi *et al.*, “Deep reinforcement learning with relational inductive biases,” presented at the International Conference on Learning Representations, Sep. 2018. Accessed: Jan. 28, 2025. [Online]. Available: <https://openreview.net/forum?id=HkxaFoC9KQ>
- [30] A. Zai and B. Brown, *Deep Reinforcement Learning in Action*. Simon and Schuster, 2020.
- [31] A. Vaswani *et al.*, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. Accessed: May 14, 2022. [Online]. Available: <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>
- [32] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete, “A survey on modern trainable activation functions,” *Neural Networks*, vol. 138, pp. 14–32, Jun. 2021, doi: 10.1016/j.neunet.2021.01.026.
- [33] C. Szepesvári, *Algorithms for Reinforcement Learning*. Springer Nature, 2022.
- [34] N. Hariharan and A. G. Paavai, “A Brief Study of Deep Reinforcement Learning with Epsilon-Greedy Exploration.,” *International Journal of Computing and Digital Systems*, vol. 11, no. 1, pp. 541–552, Jan. 2022, doi: 10.12785/ijcds/110144.
- [35] M. Hessel *et al.*, “Rainbow: Combining Improvements in Deep Reinforcement Learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Art. no. 1, Apr. 2018, doi: 10.1609/aaai.v32i1.11796.

- [36] H. Zhang and T. Yu, “Taxonomy of Reinforcement Learning Algorithms,” in *Deep Reinforcement Learning: Fundamentals, Research and Applications*, H. Dong, Z. Ding, and S. Zhang, Eds., Singapore: Springer, 2020, pp. 125–133. doi: 10.1007/978-981-15-4095-0_3.
- [37] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust Region Policy Optimization,” Apr. 20, 2017, *arXiv*: arXiv:1502.05477. Accessed: Aug. 06, 2024. [Online]. Available: <http://arxiv.org/abs/1502.05477>
- [38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” Aug. 28, 2017, *arXiv*: arXiv:1707.06347. doi: 10.48550/arXiv.1707.06347.
- [39] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-Dimensional Continuous Control Using Generalized Advantage Estimation,” Oct. 20, 2018, *arXiv*: arXiv:1506.02438. Accessed: Oct. 09, 2024. [Online]. Available: <http://arxiv.org/abs/1506.02438>
- [40] J. Konečný, J. Liu, P. Richtárik, and M. Takáč, “Mini-Batch Semi-Stochastic Gradient Descent in the Proximal Setting,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 2, pp. 242–255, Mar. 2016, doi: 10.1109/JSTSP.2015.2505682.
- [41] S. M. Johnson, “Optimal two- and three-stage production schedules with setup times included,” *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 61–68, 1954, doi: 10.1002/nav.3800010110.
- [42] J. Kallrath, “Planning and scheduling in the process industry,” *OR Spectrum*, vol. 24, no. 3, pp. 219–250, Aug. 2002, doi: 10.1007/s00291-002-0101-7.
- [43] C. T. Maravelias and C. Sung, “Integration of production planning and scheduling: Overview, challenges and opportunities,” *Computers & Chemical Engineering*, vol. 33, no. 12, pp. 1919–1930, Dec. 2009, doi: 10.1016/j.compchemeng.2009.06.007.
- [44] C. T. Maravelias, “General framework and modeling approach classification for chemical production scheduling,” *AIChE Journal*, vol. 58, no. 6, pp. 1812–1828, 2012, doi: 10.1002/aic.13801.
- [45] O. Andrés-Martínez and L. A. Ricardez-Sandoval, “Integration of planning, scheduling, and control: A review and new perspectives,” *The Canadian Journal of Chemical Engineering*, vol. 100, no. 9, pp. 2057–2070, 2022, doi: 10.1002/cjce.24501.
- [46] I. E. Grossmann, “Discrete Optimization Methods and their Role in the Integration of Planning and Scheduling”.
- [47] *Production Planning by Mixed Integer Programming*. in Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. doi: 10.1007/0-387-33477-7.
- [48] E. Kondili, C. C. Pantelides, and R. W. H. Sargent, “A general algorithm for short-term scheduling of batch operations—I. MILP formulation,” *Computers & Chemical Engineering*, vol. 17, no. 2, Art. no. 2, Feb. 1993, doi: 10.1016/0098-1354(93)80015-F.
- [49] C. T. Maravelias, “Mixed-Time Representation for State-Task Network Models,” *Ind. Eng. Chem. Res.*, vol. 44, no. 24, pp. 9129–9145, Nov. 2005, doi: 10.1021/ie0500117.
- [50] Z. Li and M. G. Ierapetritou, “Robust Optimization for Process Scheduling Under Uncertainty,” *Ind. Eng. Chem. Res.*, vol. 47, no. 12, Art. no. 12, Jun. 2008, doi: 10.1021/ie071431u.
- [51] H.-J. Zimmermann, “Fuzzy set theory,” *WIREs Computational Statistics*, vol. 2, no. 3, pp. 317–332, 2010, doi: 10.1002/wics.82.
- [52] H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, “Executing production schedules in the face of uncertainties: A review and some future directions,” *European Journal of Operational Research*, vol. 161, no. 1, pp. 86–110, Feb. 2005, doi: 10.1016/j.ejor.2003.08.027.
- [53] C. A. Mendez and J. Cerdá, “An MILP framework for batch reactive scheduling with limited discrete resources,” *Computers & Chemical Engineering*, vol. 28, no. 6, pp. 1059–1068, Jun. 2004, doi: 10.1016/j.compchemeng.2003.09.008.
- [54] J. P. Vin and M. G. Ierapetritou, “A New Approach for Efficient Rescheduling of Multiproduct Batch Plants,” *Ind. Eng. Chem. Res.*, vol. 39, no. 11, pp. 4228–4238, Nov. 2000, doi: 10.1021/ie000233z.
- [55] D. Ruiz, J. Cantón, J. María Nogués, A. Espuña, and L. Puigjaner, “On-line fault diagnosis system support for reactive scheduling in multipurpose batch chemical plants,” *Computers & Chemical Engineering*, vol. 25, no. 4, pp. 829–837, May 2001, doi: 10.1016/S0098-1354(01)00657-3.

- [56] J. D. Ullman, “NP-complete scheduling problems,” *Journal of Computer and System Sciences*, vol. 10, no. 3, pp. 384–393, Jun. 1975, doi: 10.1016/S0022-0000(75)80008-0.
- [57] D. Rangel-Martinez, K. D. P. Nigam, and L. A. Ricardez-Sandoval, “Machine learning on sustainable energy: A review and outlook on renewable energy systems, catalysis, smart grid and energy storage,” *Chemical Engineering Research and Design*, vol. 174, pp. 414–441, Oct. 2021, doi: 10.1016/j.cherd.2021.08.013.
- [58] D. Rangel-Martinez and L. A. Ricardez-Sandoval, “Data-driven techniques for optimal and sustainable process integration of chemical and manufacturing systems,” in *Optimization in Chemical Engineering: Deterministic, Meta-Heuristic and Data-Driven Techniques*, Walter de Gruyter GmbH & Co KG, 2025, pp. 215–255.
- [59] I. Halperin, “Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions: by Warren B. Powell (ed.), Wiley (2022). Hardback. ISBN 9781119815051.,” *Quantitative Finance*, vol. 22, no. 12, pp. 2151–2154, Dec. 2022, doi: 10.1080/14697688.2022.2135456.
- [60] B. Waschneck *et al.*, “Deep reinforcement learning for semiconductor production scheduling,” in *2018 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, Apr. 2018, pp. 301–306. doi: 10.1109/ASMC.2018.8373191.
- [61] C. D. Hubbs, C. Li, N. V. Sahinidis, I. E. Grossmann, and J. M. Wassick, “A deep reinforcement learning approach for chemical production scheduling,” *Computers & Chemical Engineering*, vol. 141, p. 106982, Oct. 2020, doi: 10.1016/j.compchemeng.2020.106982.
- [62] C. Hubbs, A. Kelloway, J. Wassick, N. Sahinidis, and I. Grossmann, *An Industrial Application of Deep Reinforcement Learning for Chemical Production Scheduling*. 2020.
- [63] T. Altenmüller, T. Stüker, B. Waschneck, A. Kuhnle, and G. Lanza, “Reinforcement learning for an intelligent and autonomous production control of complex job-shops under time constraints,” *Prod. Eng. Res. Devel.*, vol. 14, no. 3, Art. no. 3, Jun. 2020, doi: 10.1007/s11740-020-00967-8.
- [64] C.-Y. Lee, Y.-T. Huang, and P.-J. Chen, “Robust-optimization-guiding deep reinforcement learning for chemical material production scheduling,” *Computers & Chemical Engineering*, vol. 187, p. 108745, Aug. 2024, doi: 10.1016/j.compchemeng.2024.108745.
- [65] Y. Chen, J. Ding, and Q. Chen, “A Reinforcement Learning Based Large-Scale Refinery Production Scheduling Algorithm,” *IEEE Transactions on Automation Science and Engineering*, vol. 21, no. 4, pp. 6041–6055, Oct. 2024, doi: 10.1109/TASE.2023.3321612.
- [66] Z. Shao, F. Si, D. Kudenko, P. Wang, and X. Tong, “Predictive scheduling of wet flue gas desulfurization system based on reinforcement learning,” *Computers & Chemical Engineering*, vol. 141, p. 107000, Oct. 2020, doi: 10.1016/j.compchemeng.2020.107000.
- [67] P. Pravin, Z. Luo, L. Li, and X. Wang, “Learning-based scheduling of industrial hybrid renewable energy systems,” *Computers & Chemical Engineering*, vol. 159, p. 107665, Mar. 2022, doi: 10.1016/j.compchemeng.2022.107665.
- [68] T. J. Ikonen, K. Heljanko, and I. Harjunoski, “Reinforcement learning of adaptive online rescheduling timing and computing time allocation,” *Computers & Chemical Engineering*, vol. 141, p. 106994, Oct. 2020, doi: 10.1016/j.compchemeng.2020.106994.
- [69] X. Gao, D. Peng, G. Kui, J. Pan, X. Zuo, and F. Li, “Reinforcement learning based optimization algorithm for maintenance tasks scheduling in coalbed methane gas field,” *Computers & Chemical Engineering*, vol. 170, p. 108131, Feb. 2023, doi: 10.1016/j.compchemeng.2022.108131.
- [70] X. Wu, X. Yan, D. Guan, and M. Wei, “A deep reinforcement learning model for dynamic job-shop scheduling problem with uncertain processing time,” *Engineering Applications of Artificial Intelligence*, vol. 131, p. 107790, May 2024, doi: 10.1016/j.engappai.2023.107790.
- [71] G. Bonetta, D. Zago, R. Cancelliere, and A. Grosso, “Job Shop Scheduling via Deep Reinforcement Learning: A Sequence to Sequence Approach,” in *Learning and Intelligent Optimization*, M. Sellmann and K. Tierney, Eds., Cham: Springer International Publishing, 2023, pp. 475–490. doi: 10.1007/978-3-031-44505-7_32.

- [72] Z. Pan, L. Wang, J. Wang, and J. Lu, “Deep Reinforcement Learning Based Optimization Algorithm for Permutation Flow-Shop Scheduling,” *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 7, no. 4, pp. 983–994, Aug. 2023, doi: 10.1109/TETCI.2021.3098354.
- [73] B. A. Han and J. J. Yang, “A Deep Reinforcement Learning Based Solution for Flexible Job Shop Scheduling Problem,” *Int. j. simul. model.*, vol. 20, no. 2, Art. no. 2, Jun. 2021, doi: 10.2507/IJSIMM20-2-CO7.
- [74] Z. Wang, B. Cai, J. Li, D. Yang, Y. Zhao, and H. Xie, “Solving non-permutation flow-shop scheduling problem via a novel deep reinforcement learning approach,” *Computers & Operations Research*, vol. 151, p. 106095, Mar. 2023, doi: 10.1016/j.cor.2022.106095.
- [75] R. Pan, X. Dong, and S. Han, “Solving Permutation Flowshop Problem with Deep Reinforcement Learning,” in *2020 Prognostics and Health Management Conference (PHM-Besançon)*, May 2020, pp. 349–353. doi: 10.1109/PHM-Besancon49106.2020.00068.
- [76] M. Monaci, V. Agasucci, and G. Grani, “An actor-critic algorithm with policy gradients to solve the job shop scheduling problem using deep double recurrent agents,” *European Journal of Operational Research*, vol. 312, no. 3, pp. 910–926, Feb. 2024, doi: 10.1016/j.ejor.2023.07.037.
- [77] G. Wu, M. A. de Carvalho Servia, and M. Mowbray, “Distributional reinforcement learning for inventory management in multi-echelon supply chains,” *Digital Chemical Engineering*, vol. 6, p. 100073, Mar. 2023, doi: 10.1016/j.dche.2022.100073.
- [78] S. Lang, F. Behrendt, N. Lanzerath, T. Reggelin, and M. Müller, “Integration of Deep Reinforcement Learning and Discrete-Event Simulation for Real-Time Scheduling of a Flexible Job Shop Production,” in *2020 Winter Simulation Conference (WSC)*, Dec. 2020, pp. 3057–3068. doi: 10.1109/WSC48552.2020.9383997.
- [79] R. Chen, W. Li, and H. Yang, “A Deep Reinforcement Learning Framework Based on an Attention Mechanism and Disjunctive Graph Embedding for the Job-Shop Scheduling Problem,” *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1322–1331, Feb. 2023, doi: 10.1109/TII.2022.3167380.
- [80] W. Zhang, F. Zhao, Y. Li, C. Du, X. Feng, and X. Mei, “A novel collaborative agent reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for flexible job shop scheduling problem,” *Journal of Manufacturing Systems*, vol. 74, pp. 329–345, Jun. 2024, doi: 10.1016/j.jmsy.2024.03.012.
- [81] R. Wang, G. Wang, J. Sun, F. Deng, and J. Chen, “Flexible Job Shop Scheduling via Dual Attention Network-Based Reinforcement Learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 3, pp. 3091–3102, Mar. 2024, doi: 10.1109/TNNLS.2023.3306421.
- [82] J. Lee, S. Kee, M. Janakiram, and G. Runger, “Attention-based Reinforcement Learning for Combinatorial Optimization: Application to Job Shop Scheduling Problem,” Mar. 18, 2024, *arXiv: arXiv:2401.16580*. doi: 10.48550/arXiv.2401.16580.
- [83] R. Magalhães, M. Martins, S. Vieira, F. Santos, and J. Sousa, “Encoder-Decoder Neural Network Architecture for solving Job Shop Scheduling Problems using Reinforcement Learning,” in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec. 2021, pp. 01–08. doi: 10.1109/SSCI50451.2021.9659849.
- [84] G. Gebreyesus, G. Fellek, A. Farid, S. Fujimura, and O. Yoshie, “Gated-Attention Model with Reinforcement Learning for Solving Dynamic Job Shop Scheduling Problem,” *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 18, no. 6, pp. 932–944, 2023, doi: 10.1002/tee.23788.
- [85] Z. Liao, J. Chen, and Z. Zhang, “Solving Job-Shop Scheduling Problem via Deep Reinforcement Learning with Attention Model,” in *Advances and Trends in Artificial Intelligence. Theory and Applications*, H. Fujita, Y. Wang, Y. Xiao, and A. Moonis, Eds., Cham: Springer Nature Switzerland, 2023, pp. 201–212. doi: 10.1007/978-3-031-36822-6_18.
- [86] S. Yang, “Using Attention Mechanism to Solve Job Shop Scheduling Problem,” in *2022 2nd International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, Jan. 2022, pp. 59–62. doi: 10.1109/ICCECE54139.2022.9712705.

- [87] D. J. Garcia, J. Gong, and F. You, “Multi-Stage Adaptive Robust Optimization over Bioconversion Product and Process Networks with Uncertain Feedstock Price and Biofuel Demand,” in *Computer Aided Chemical Engineering*, vol. 38, Z. Kravanja and M. Bogataj, Eds., in 26 European Symposium on Computer Aided Process Engineering, vol. 38., Elsevier, 2016, pp. 217–222. doi: 10.1016/B978-0-444-63428-3.50041-2.
- [88] N. V. Sahinidis, “Optimization under uncertainty: state-of-the-art and opportunities,” *Computers & Chemical Engineering*, vol. 28, no. 6, pp. 971–983, Jun. 2004, doi: 10.1016/j.compchemeng.2003.09.017.
- [89] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson, “Deep Variational Reinforcement Learning for POMDPs,” Jun. 06, 2018, *arXiv*: arXiv:1806.02426. Accessed: Sep. 09, 2023. [Online]. Available: <http://arxiv.org/abs/1806.02426>
- [90] “Masking in Deep Reinforcement Learning - Boring Guy.” Accessed: Sep. 02, 2023. [Online]. Available: <https://boring-guy.sh/posts/masking-rl/>
- [91] E. Wiewiora, “Reward Shaping,” in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds., Boston, MA: Springer US, 2010, pp. 863–865. doi: 10.1007/978-0-387-30164-8_731.
- [92] W. Jung, M. Cho, J. Park, and Y. Sung, “Quantile Constrained Reinforcement Learning: A Reinforcement Learning Framework Constraining Outage Probability,” Nov. 27, 2022, *arXiv*: arXiv:2211.15034. Accessed: Mar. 07, 2024. [Online]. Available: <http://arxiv.org/abs/2211.15034>
- [93] S. Paternain, L. Chamon, M. Calvo-Fullana, and A. Ribeiro, “Constrained Reinforcement Learning Has Zero Duality Gap,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2019. Accessed: Mar. 17, 2024. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/hash/c1aeb6517a1c7f33514f7ff69047e74e-Abstract.html
- [94] “Adam - Cornell University Computational Optimization Open Textbook - Optimization Wiki.” Accessed: Oct. 03, 2023. [Online]. Available: <https://optimization.cbe.cornell.edu/index.php?title=Adam>
- [95] M. Hausknecht and P. Stone, “Deep Recurrent Q-Learning for Partially Observable MDPs,” Jan. 11, 2017, *arXiv*: arXiv:1507.06527. Accessed: Feb. 19, 2023. [Online]. Available: <http://arxiv.org/abs/1507.06527>
- [96] V. Goel and I. E. Grossmann, “A Class of stochastic programs with decision dependent uncertainty,” *Math. Program.*, vol. 108, no. 2, pp. 355–394, Sep. 2006, doi: 10.1007/s10107-006-0715-7.
- [97] M. Ierapetritou and Z. Jia, “Scheduling under uncertainty using MILP sensitivity analysis,” in *Computer Aided Chemical Engineering*, vol. 18, A. Barbosa-Póvoa and H. Matos, Eds., in European Symposium on Computer-Aided Process Engineering-14, vol. 18., Elsevier, 2004, pp. 931–936. doi: 10.1016/S1570-7946(04)80221-9.
- [98] Z. Jia and M. Ierapetritou, “Uncertainty Analysis of Milp Problems”.
- [99] I. Pappas *et al.*, “Multiparametric Programming in Process Systems Engineering: Recent Developments and Path Forward,” *Frontiers in Chemical Engineering*, vol. 2, Jan. 2021, doi: 10.3389/fceng.2020.620168.
- [100] “What are the advantages and disadvantages of robust optimization in machine learning? | 5 Answers from Research papers,” SciSpace - Question. Accessed: Mar. 21, 2024. [Online]. Available: <https://typeset.io/questions/what-are-the-advantages-and-disadvantages-of-robust-28pcigt79>
- [101] P. Osinenko, D. Dobriborsci, and W. Aumer, “Reinforcement learning with guarantees: a review,” *IFAC-PapersOnLine*, vol. 55, no. 15, pp. 123–128, Jan. 2022, doi: 10.1016/j.ifacol.2022.07.619.
- [102] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Mach Learn*, vol. 8, no. 3, pp. 279–292, May 1992, doi: 10.1007/BF00992698.
- [103] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, and A. Kyek, “Optimization of global production scheduling with deep reinforcement learning,” *Procedia CIRP*, vol. 72, pp. 1264–1269, Jan. 2018, doi: 10.1016/j.procir.2018.03.212.
- [104] K. Lei, P. Guo, Y. Wang, J. Zhang, X. Meng, and L. Qian, “Large-Scale Dynamic Scheduling for Flexible Job-Shop With Random Arrivals of New Jobs by Hierarchical Reinforcement Learning,”

- IEEE Transactions on Industrial Informatics*, vol. 20, no. 1, pp. 1007–1018, Jan. 2024, doi: 10.1109/TII.2023.3272661.
- [105] J. Acevedo and E. N. Pistikopoulos, “Stochastic optimization based algorithms for process synthesis under uncertainty,” *Computers & Chemical Engineering*, vol. 22, no. 4, pp. 647–671, Jan. 1998, doi: 10.1016/S0098-1354(97)00234-2.
- [106] K. G. Menon, R. Fukasawa, and L. A. Ricardez-Sandoval, “A novel stochastic programming approach for scheduling of batch processes with decision dependent time of uncertainty realization,” *Ann Oper Res*, vol. 305, no. 1, pp. 163–190, Oct. 2021, doi: 10.1007/s10479-021-04141-w.
- [107] Y. Wang, H. He, and X. Tan, “Truly Proximal Policy Optimization,” in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, PMLR, Aug. 2020, pp. 113–122. Accessed: Oct. 07, 2024. [Online]. Available: <https://proceedings.mlr.press/v115/wang20b.html>
- [108] Z. Fan, R. Su, W. Zhang, and Y. Yu, “Hybrid Actor-Critic Reinforcement Learning in Parameterized Action Space,” May 30, 2019, *arXiv*: arXiv:1903.01344. Accessed: Jan. 18, 2024. [Online]. Available: <http://arxiv.org/abs/1903.01344>
- [109] L. Orseau and M. Armstrong, “Safely interruptible agents,” *Conference on Uncertainty in Artificial Intelligence*, 2016, Accessed: Dec. 04, 2024. [Online]. Available: <https://ora.ox.ac.uk/objects/uuid:17c0e095-4e13-47fc-bace-64ec46134a3f>
- [110] B. Liu, Q. Cai, Z. Yang, and Z. Wang, “Neural Trust Region/Proximal Policy Optimization Attains Globally Optimal Policy,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2019. Accessed: Nov. 30, 2024. [Online]. Available: <https://proceedings.neurips.cc/paper/2019/hash/227e072d131ba77451d8f27ab9afdfb7-Abstract.html>
- [111] B. Charpentier, R. Senanayake, M. Kochenderfer, and S. Günnemann, “Disentangling Epistemic and Aleatoric Uncertainty in Reinforcement Learning,” Jun. 03, 2022, *arXiv*: arXiv:2206.01558. Accessed: Nov. 11, 2024. [Online]. Available: <http://arxiv.org/abs/2206.01558>
- [112] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” Mar. 09, 2015, *arXiv*: arXiv:1503.02531. doi: 10.48550/arXiv.1503.02531.
- [113] S. Huang and S. Ontañón, “A Closer Look at Invalid Action Masking in Policy Gradient Algorithms,” May 31, 2022, *arXiv*: arXiv:2006.14171. doi: 10.48550/arXiv.2006.14171.
- [114] M. Morimoto, K. Fukami, R. Maulik, R. Vinuesa, and K. Fukagata, “Assessments of epistemic uncertainty using Gaussian stochastic weight averaging for fluid-flow regression,” *Physica D: Nonlinear Phenomena*, vol. 440, p. 133454, Nov. 2022, doi: 10.1016/j.physd.2022.133454.
- [115] C. Qian, T. Ganter, J. Michalenko, F. Liang, and J. Adams, “Quantifying Epistemic Uncertainty in Binary Classification via Accuracy Gain,” *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 17, no. 5, p. e11709, 2024, doi: 10.1002/sam.11709.
- [116] V. Singh, K. Dey, and S. Padmanabhan, “Signal Processing Techniques for Noise Reduction in Industrial Machinery,” in *2024 International Conference on Advances in Computing Research on Science Engineering and Technology (ACROSET)*, Sep. 2024, pp. 1–6. doi: 10.1109/ACROSET62108.2024.10743847.
- [117] L. G. Papageorgiou and C. C. Pantelides, “Optimal Campaign Planning/Scheduling of Multipurpose Batch/Semicontinuous Plants. 2. A Mathematical Decomposition Approach,” *Ind. Eng. Chem. Res.*, vol. 35, no. 2, pp. 510–529, Jan. 1996, doi: 10.1021/ie950082d.
- [118] R. W. Koller, L. A. Ricardez-Sandoval, and L. T. Biegler, “Stochastic back-off algorithm for simultaneous design, control, and scheduling of multiproduct systems under uncertainty,” *AIChE Journal*, vol. 64, no. 7, Art. no. 7, 2018, doi: 10.1002/aic.16092.
- [119] J. Zhuge and M. G. Ierapetritou, “An integrated framework for scheduling and control using fast model predictive control,” *AIChE Journal*, vol. 61, no. 10, pp. 3304–3319, 2015, doi: 10.1002/aic.14914.
- [120] Y. I. Valdez-Navarro and L. A. Ricardez-Sandoval, “A Novel Back-off Algorithm for Integration of Scheduling and Control of Batch Processes under Uncertainty,” *Ind. Eng. Chem. Res.*, vol. 58, no. 48, pp. 22064–22083, Dec. 2019, doi: 10.1021/acs.iecr.9b04963.

- [121] K. G. Menon, R. Fukasawa, and L. A. Ricardez-Sandoval, "Integration of Planning and Scheduling for Large-Scale Multijob Multitasking Batch Plants," *Ind. Eng. Chem. Res.*, vol. 63, no. 2, pp. 1039–1054, Jan. 2024, doi: 10.1021/acs.iecr.3c02408.
- [122] H. U. Rodríguez Vera and L. A. Ricardez-Sandoval, "Integration of Scheduling and Control for Chemical Batch Plants under Stochastic Uncertainty: A Back-Off Approach," *Ind. Eng. Chem. Res.*, vol. 61, no. 12, pp. 4363–4378, Mar. 2022, doi: 10.1021/acs.iecr.1c04386.
- [123] P. Agathoklis and L. T. Bruton, "Practical-BIBO stability of n-dimensional discrete systems," *IEE Proceedings G (Electronic Circuits and Systems)*, vol. 130, no. 6, pp. 236–242, Dec. 1983, doi: 10.1049/ip-g-1.1983.0045.
- [124] C. Pukdeboon, "A Review of Fundamentals of Lyapunov Theory," vol. 10, no. 2.
- [125] N. Abou Baker, N. Zengeler, and U. Handmann, "A Transfer Learning Evaluation of Deep Neural Networks for Image Classification," *Machine Learning and Knowledge Extraction*, vol. 4, no. 1, Art. no. 1, Mar. 2022, doi: 10.3390/make4010002.
- [126] M. Rafiei and L. A. Ricardez-Sandoval, "New frontiers, challenges, and opportunities in integration of design and control for enterprise-wide sustainability," *Computers & Chemical Engineering*, vol. 132, p. 106610, Jan. 2020, doi: 10.1016/j.compchemeng.2019.106610.
- [127] D. A. Liñán and L. A. Ricardez-Sandoval, "Trends and perspectives in deterministic MINLP optimization for integrated planning, scheduling, control, and design of chemical processes," *Reviews in Chemical Engineering*, Mar. 2025, doi: 10.1515/revce-2024-0064.
- [128] F. I. Gómez-Castro and V. Rico-Ramírez, *Optimization in Chemical Engineering: Deterministic, Meta-Heuristic and Data-Driven Techniques*. Walter de Gruyter GmbH & Co KG, 2025.
- [129] D. A. Liñán, J. A. McCormick-Mantilla, and L. A. Ricardez-Sandoval, "Economic Nonlinear Model Predictive Control and Scheduling of Multiple Fed-Batch Fermenters in a Lignocellulosic Biorefinery," *Computers & Chemical Engineering*, p. 109223, Jun. 2025, doi: 10.1016/j.compchemeng.2025.109223.
- [130] H. Alibrahim and S. A. Ludwig, "Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization," in *2021 IEEE Congress on Evolutionary Computation (CEC)*, Jun. 2021, pp. 1551–1559. doi: 10.1109/CEC45853.2021.9504761.
- [131] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction," in *Proceedings of the 34th International Conference on Machine Learning*, PMLR, Jul. 2017, pp. 2778–2787. Accessed: Feb. 05, 2025. [Online]. Available: <https://proceedings.mlr.press/v70/pathak17a.html>
- [132] M. S. Abdul Hameed, M. M. Khan, and A. Schwung, "Curiosity Based RL on Robot Manufacturing Cell," in *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, Mar. 2021, pp. 1048–1053. doi: 10.1109/ICIT46573.2021.9453577.

Appendix A.

A.1. Collection of Sequential Information

A.1.1. Observation Vector

An example on how an observation vector is filled along six time-intervals is described in this section. A two-product (Pr1 and Pr2) plant with four stages, each one with one machine is used for this example. Assume the initialization of jobs, i.e., job_1 and job_2 , where the processing times are $\mathbf{X}_1 = \begin{bmatrix} 1,2,0,1 \\ 1,0,1,1 \end{bmatrix}$, recall that the first row correspond to job_1 and the second to job_2 which will be identified as 1 and 2 respectively in the observation vector. Note that the zero values means that the job does not use that particular stage. The idle state action is identified with a 0. The production value q_j for job_1 is $q_{j=1} = 50 \text{ m}^3$ and for job_2 is $q_{j=2} = 35 \text{ m}^3$. A demand of 100 m^3 is assumed for product Pr1 and a demand of 70 m^3 is assumed for Pr2. The value of $c = 4$ and a horizon of $T = 20$ is considered. The observation at each time interval until the end of the horizon is described as follows where product Pr1 is initialized at $t = 15$ and $t = 18$, and Pr2 is initialized at $t = 16$ and $t = 19$. As shown in Table A.1, the information corresponding to the $action_t$ is updated as the operation proceeds in the plant. For instance, the information related to $action_{15}$ in $\mathbf{O}_{t=15}$ is updated and relocated in the subsequent observations (i.e., $\mathbf{O}_{t=16}$, $\mathbf{O}_{t=17}$, and $\mathbf{O}_{t=18}$). The idle state action, represented with a 0, and the information related to it remains equal in the observations as time progresses, for instance $action_{t=14}$ in $\mathbf{O}_{t=15}$, $\mathbf{O}_{t=16}$, and $\mathbf{O}_{t=17}$.

Table A. 1 Example of observations from $t=15$ to $t=20$

$\mathbf{O}_{t=15} = [0, 0, 0, 0, action_{12} = 0,$ $0, 0, 0, 0, action_{13} = 0,$ $0, 0, 0, 0, action_{14} = 0,$ $1, 0, 0, 0, action_{15} = 1,$ $z_{pr1} = 100, z_{pr2} = 70, t = 15]$	$\mathbf{O}_{t=16} = [0, 0, 0, 0, action_{13} = 0,$ $0, 0, 0, 0, action_{14} = 0,$ $-1, 1, 0, 0, action_{15} = 1,$ $1, 0, 0, 0, action_{16} = 2,$ $z_{pr1} = 100, z_{pr2} = 70, t = 16]$
$\mathbf{O}_{t=17} = [0, 0, 0, 0, action_{14} = 0,$ $-1, 2, 0, 0, action_{15} = 1,$ $-1, 0, 1, 0, action_{16} = 2,$ $0, 0, 0, 0, action_{17} = 0,$ $z_{pr1} = 100, z_{pr2} = 70, t = 17]$	$\mathbf{O}_{t=18} = [-1, -1, 0, 1, action_{15} = 1,$ $-1, 0, -1, 1, action_{16} = 2,$ $0, 0, 0, 0, action_{17} = 0,$ $1, 0, 0, 0, action_{18} = 1,$ $z_{pr1} = 100, z_{pr2} = 70, t = 18]$
$\mathbf{O}_{t=19} = [-1, -1, -1, -1, action_{16} = 2,$ $0, 0, 0, 0, action_{17} = 0,$ $-1, 1, 0, 0, action_{18} = 1,$ $1, 0, 0, 0, action_{19} = 2,$ $z_{pr1} = 50, z_{pr2} = 35, t = 19]$	$\mathbf{O}_{t=20} = [0, 0, 0, 0, action_{17} = 0,$ $-1, 2, 0, 0, action_{18} = 0,$ $-1, 0, 1, 0, action_{19} = 2,$ $0, 0, 0, 0, action_{20} = 0,$ $z_{pr1} = 50, z_{pr2} = 35, t = 20]$

A.1.2. Input Sequence for the RNN

Assume that $\Gamma = 3$ and that three consecutive inputs for the RNN ($\mathbf{input}_{t=18}$, $\mathbf{input}_{t=19}$, and $\mathbf{input}_{t=20}$) are meant to be defined with the observations described in Table A.1. Thus, the sequences of observations for each input are:

$$\begin{aligned}
 \mathbf{input}_{t=18} &= [\mathbf{O}_{t=16}, \mathbf{O}_{t=17}, \mathbf{O}_{t=18}] \\
 \mathbf{input}_{t=19} &= [\mathbf{O}_{t=17}, \mathbf{O}_{t=18}, \mathbf{O}_{t=19}] \\
 \mathbf{input}_{t=20} &= [\mathbf{O}_{t=18}, \mathbf{O}_{t=19}, \mathbf{O}_{t=20}]
 \end{aligned} \tag{A.1}$$

The agent considers the information generated in the previous time units; hence, the *recurrence* of the network is observable. Note that although the agent is partially using the same information, the \mathbf{input}_t sequence is introduced into the RNN as a concatenated vector. Thus, this \mathbf{input}_t vector is continuously changing at each time unit.

A.2. Equations and Tables

Table A. 2 Hyperparameters for RL methodology

Hyperparameter	Value
Memory of the buffer	200
Learning Rate α	5e-5
Gamma γ	0.7
Epsilon value ϵ	0.8 to 0
Window of parameters c	30
Sequence length Γ	4
Hidden layer size λ	20
Activation function	tanh(x)
Time-intervals per episode T	150
Target network update	Every 10 episodes

Table A. 3 Paths for case study 2

Product	Jobs in I	Processing route K_j ($p_1^j \rightarrow p_2^j \rightarrow \dots \rightarrow p_{10}^j$)	Output (q_j)
Pr1	job ₁	$p_{1,1}^1 \rightarrow p_{2,2}^1 \rightarrow p_{1,3}^1 \rightarrow p_{1,4}^1 \rightarrow p_{1,5}^1 \rightarrow p_{2,6}^1 \rightarrow p_{2,7}^1 \rightarrow p_{1,8}^1 \rightarrow p_{1,9}^1 \rightarrow p_{1,10}^1$	70
Pr1	job ₂	$p_{1,1}^2 \rightarrow p_{2,2}^2 \rightarrow p_{1,3}^2 \rightarrow p_{1,4}^2 \rightarrow p_{1,5}^2 \rightarrow p_{2,6}^2 \rightarrow p_{2,7}^2 \rightarrow p_{1,8}^2 \rightarrow p_{1,9}^2 \rightarrow p_{1,10}^2$	50
Pr2	job ₃	$p_{2,1}^3 \rightarrow p_{1,2}^3 \rightarrow p_{1,3}^3 \rightarrow p_{1,4}^3 \rightarrow p_{1,5}^3 \rightarrow p_{1,6}^3 \rightarrow p_{1,7}^3 \rightarrow p_{2,8}^3 \rightarrow p_{2,9}^3 \rightarrow p_{2,10}^3$	80
Pr2	job ₄	$p_{1,1}^4 \rightarrow p_{2,2}^4 \rightarrow p_{1,3}^4 \rightarrow p_{1,4}^4 \rightarrow p_{1,5}^4 \rightarrow p_{2,6}^4 \rightarrow p_{2,7}^4 \rightarrow p_{1,8}^4 \rightarrow p_{1,9}^4 \rightarrow p_{1,10}^4$	60
Pr3	job ₅	$p_{2,1}^5 \rightarrow p_{1,2}^5 \rightarrow p_{1,3}^5 \rightarrow p_{1,4}^5 \rightarrow p_{1,5}^5 \rightarrow p_{1,6}^5 \rightarrow p_{1,7}^5 \rightarrow p_{2,8}^5 \rightarrow p_{2,9}^5 \rightarrow p_{2,10}^5$	80
Pr3	job ₆	$p_{1,1}^6 \rightarrow p_{1,2}^6 \rightarrow p_{2,3}^6 \rightarrow p_{2,4}^6 \rightarrow p_{2,5}^6 \rightarrow p_{2,6}^6 \rightarrow p_{1,7}^6 \rightarrow p_{1,8}^6 \rightarrow p_{2,9}^6 \rightarrow p_{1,10}^6$	50
Pr4	job ₇	$p_{1,1}^7 \rightarrow p_{1,2}^7 \rightarrow p_{2,3}^7 \rightarrow p_{2,4}^7 \rightarrow p_{2,5}^7 \rightarrow p_{2,6}^7 \rightarrow p_{1,7}^7 \rightarrow p_{1,8}^7 \rightarrow p_{2,9}^7 \rightarrow p_{1,10}^7$	70
Pr4	job ₈	$p_{2,1}^8 \rightarrow p_{1,2}^8 \rightarrow p_{1,3}^8 \rightarrow p_{2,4}^8 \rightarrow p_{2,5}^8 \rightarrow p_{1,6}^8 \rightarrow p_{2,7}^8 \rightarrow p_{1,8}^8 \rightarrow p_{2,9}^8 \rightarrow p_{2,10}^8$	50
Pr5	job ₉	$p_{2,1}^9 \rightarrow p_{1,2}^9 \rightarrow p_{1,3}^9 \rightarrow p_{2,4}^9 \rightarrow p_{2,5}^9 \rightarrow p_{1,6}^9 \rightarrow p_{2,7}^9 \rightarrow p_{1,8}^9 \rightarrow p_{2,9}^9 \rightarrow p_{2,10}^9$	70
Pr5	job ₁₀	$p_{1,1}^{10} \rightarrow p_{2,2}^{10} \rightarrow p_{2,3}^{10} \rightarrow p_{2,4}^{10} \rightarrow p_{1,5}^{10} \rightarrow p_{1,6}^{10} \rightarrow p_{1,7}^{10} \rightarrow p_{2,8}^{10} \rightarrow p_{1,9}^{10} \rightarrow p_{1,10}^{10}$	60
Pr6	job ₁₁	$p_{2,1}^{11} \rightarrow p_{2,2}^{11} \rightarrow p_{2,3}^{11} \rightarrow p_{1,4}^{11} \rightarrow p_{2,5}^{11} \rightarrow p_{2,6}^{11} \rightarrow p_{1,7}^{11} \rightarrow p_{1,8}^{11} \rightarrow p_{2,9}^{11} \rightarrow p_{2,10}^{11}$	80
Pr7	job ₁₂	$p_{1,1}^{12} \rightarrow p_{1,2}^{12} \rightarrow p_{2,3}^{12} \rightarrow p_{1,4}^{12} \rightarrow p_{1,5}^{12} \rightarrow p_{1,6}^{12} \rightarrow p_{2,7}^{12} \rightarrow p_{2,8}^{12} \rightarrow p_{1,9}^{12} \rightarrow p_{2,10}^{12}$	70
Pr8	job ₁₃	$p_{2,1}^{13} \rightarrow p_{2,2}^{13} \rightarrow p_{1,3}^{13} \rightarrow p_{1,4}^{13} \rightarrow p_{2,5}^{13} \rightarrow p_{2,6}^{13} \rightarrow p_{2,7}^{13} \rightarrow p_{2,8}^{13} \rightarrow p_{1,9}^{13} \rightarrow p_{1,10}^{13}$	80
Pr9	job ₁₄	$p_{1,1}^{14} \rightarrow p_{1,2}^{14} \rightarrow p_{1,3}^{14} \rightarrow p_{1,4}^{14} \rightarrow p_{1,5}^{14} \rightarrow p_{1,6}^{14} \rightarrow p_{2,7}^{14} \rightarrow p_{2,8}^{14} \rightarrow p_{1,9}^{14} \rightarrow p_{2,10}^{14}$	70
Pr10	job ₁₅	$p_{2,1}^{15} \rightarrow p_{2,2}^{15} \rightarrow p_{2,3}^{15} \rightarrow p_{2,4}^{15} \rightarrow p_{2,5}^{15} \rightarrow p_{2,6}^{15} \rightarrow p_{1,7}^{15} \rightarrow p_{1,8}^{15} \rightarrow p_{1,9}^{15} \rightarrow p_{1,10}^{15}$	80

The following equation shows the possible realizations for X_a with the values of four specific jobs: 1, 6, 9, and 14 in stages 8, 2, 5, and 3 respectively, changed at each possible realization. Recall that the jobs correspond to the rows and the stages to the columns in the matrix. The elements are displayed in the font color red so the reader can localize them straightforwardly.

$$\begin{aligned}
& \{X_1, X_2, X_3, X_4\} \\
& = \\
& \left(\begin{array}{cccc}
\left[\begin{array}{l} 3, 2, 2, 2, 2, 2, 2, 2, 3, 3 \\ 2, 2, 2, 1, 2, 1, 2, 1, 3, 2 \\ 3, 3, 2, 2, 3, 1, 2, 3, 1, 2 \\ 2, 2, 1, 2, 3, 1, 2, 2, 1, 2 \\ 2, 3, 1, 2, 2, 1, 2, 3, 3, 2 \\ 2, 3, 1, 2, 2, 1, 2, 2, 3, 1 \\ 3, 2, 2, 3, 1, 3, 1, 2, 3, 1 \\ 2, 2, 2, 3, 1, 2, 1, 1, 3, 1 \\ 2, 3, 1, 1, 2, 3, 2, 1, 3, 2 \\ 2, 2, 1, 1, 2, 2, 2, 1, 2, 1 \\ 3, 2, 1, 2, 2, 1, 2, 2, 3, 2 \\ 1, 2, 1, 2, 2, 2, 2, 2, 3, 2 \\ 2, 2, 2, 2, 1, 1, 2, 3, 2, 3 \\ 1, 2, 1, 2, 2, 2, 1, 2, 2, 2 \\ 1, 2, 2, 2, 3, 2, 2, 3, 2, 2 \end{array} \right] & , & \left[\begin{array}{l} 3, 2, 2, 2, 2, 2, 2, 3, 3, 3 \\ 2, 2, 2, 1, 2, 1, 2, 1, 3, 2 \\ 3, 3, 2, 2, 3, 1, 2, 3, 1, 2 \\ 2, 2, 1, 2, 3, 1, 2, 2, 1, 2 \\ 2, 3, 1, 2, 2, 1, 2, 3, 3, 2 \\ 2, 1, 1, 2, 2, 1, 2, 2, 3, 1 \\ 3, 2, 2, 3, 1, 3, 1, 2, 3, 1 \\ 2, 2, 2, 3, 1, 2, 1, 1, 3, 1 \\ 2, 3, 1, 1, 3, 3, 2, 1, 3, 2 \\ 2, 2, 1, 1, 2, 2, 2, 1, 2, 1 \\ 3, 2, 1, 2, 2, 1, 2, 2, 3, 2 \\ 1, 2, 1, 2, 2, 2, 2, 2, 3, 2 \\ 2, 2, 2, 2, 1, 1, 2, 3, 2, 3 \\ 1, 2, 2, 2, 2, 2, 1, 2, 2, 2 \\ 1, 2, 2, 2, 3, 2, 2, 3, 2, 2 \end{array} \right] & , & \left[\begin{array}{l} 3, 2, 2, 2, 2, 2, 2, 4, 3, 3 \\ 2, 2, 2, 1, 2, 1, 2, 1, 3, 2 \\ 3, 3, 2, 2, 3, 1, 2, 3, 1, 2 \\ 2, 2, 1, 2, 3, 1, 2, 2, 1, 2 \\ 2, 3, 1, 2, 2, 1, 2, 3, 3, 2 \\ 2, 4, 1, 2, 2, 1, 2, 2, 3, 1 \\ 3, 2, 2, 3, 1, 3, 1, 2, 3, 1 \\ 2, 2, 2, 3, 1, 2, 1, 1, 3, 1 \\ 2, 3, 1, 1, 4, 3, 2, 1, 3, 2 \\ 2, 2, 1, 1, 2, 2, 2, 1, 2, 1 \\ 3, 2, 1, 2, 2, 1, 2, 2, 3, 2 \\ 1, 2, 1, 2, 2, 2, 2, 2, 3, 2 \\ 2, 2, 2, 2, 1, 1, 2, 3, 2, 3 \\ 1, 2, 4, 2, 2, 2, 1, 2, 2, 2 \\ 1, 2, 2, 2, 3, 2, 2, 3, 2, 2 \end{array} \right] & , & \left[\begin{array}{l} 3, 2, 2, 2, 2, 2, 2, 1, 3, 3 \\ 2, 2, 2, 1, 2, 1, 2, 1, 3, 2 \\ 3, 3, 2, 2, 3, 1, 2, 3, 1, 2 \\ 2, 2, 1, 2, 3, 1, 2, 2, 1, 2 \\ 2, 3, 1, 2, 2, 1, 2, 3, 3, 2 \\ 2, 1, 1, 2, 2, 1, 2, 2, 3, 1 \\ 3, 2, 2, 3, 1, 3, 1, 2, 3, 1 \\ 2, 2, 2, 3, 1, 2, 1, 1, 3, 1 \\ 2, 3, 1, 1, 1, 3, 2, 1, 3, 2 \\ 2, 2, 1, 1, 2, 2, 2, 1, 2, 1 \\ 3, 2, 1, 2, 2, 1, 2, 2, 3, 2 \\ 1, 2, 1, 2, 2, 2, 2, 2, 3, 2 \\ 2, 2, 2, 2, 1, 1, 2, 3, 2, 3 \\ 1, 2, 3, 2, 2, 2, 1, 2, 2, 2 \\ 1, 2, 2, 2, 3, 2, 2, 3, 2, 2 \end{array} \right]
\end{array} \right) \quad (A.2)
\end{aligned}$$

$$D_{30} = D_{60} = D_{90} = \begin{pmatrix} [90, 80, 90, 100, 80, 80, 90, 90, 80, 90], \\ [80, 80, 90, 90, 90, 90, 80, 100, 100, 80], \\ [100, 90, 90, 80, 100, 100, 90, 80, 80, 80], \\ [80, 100, 100, 80, 90, 90, 90, 80, 90, 100] \end{pmatrix} \quad (A.3)$$

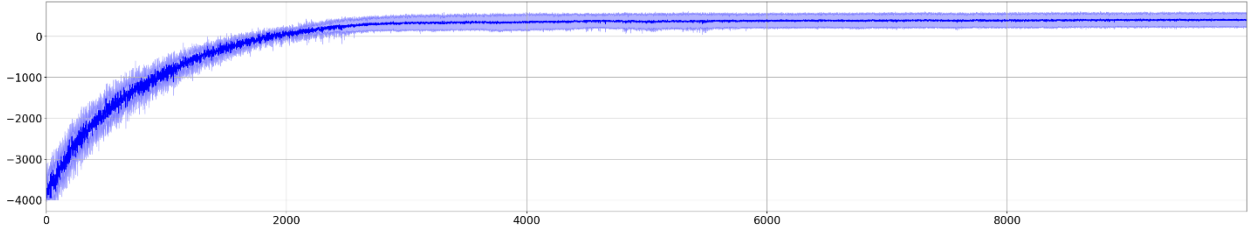


Figure A 1 Cumulative reward during 10,000 episodes of training

A.3. Flow-Shop Case Study

In this section, a case study for a flow-shop under uncertainty is solved using the framework presented in this work. The plant has 20 stages with one machine at each stage, the number of products is 10 and the number of jobs available to produce them is 15. The initial demand is $50 m^3$ for each of the ten products. The set of processing times is shown in Eq. A.4 and the sets $D_{t \in \Omega}$ of demands is that one presented in Eq. A.5. The selection process of the uncertain parameters remains the same and the hyperparameters are also the same as used in the previous cases (see Table A.2). The training was set for 10,000 episodes but

convergence was achieved after 2,750 episodes. The system of warehouses to store the excess product remains the same as in case study 2, i.e., ten warehouses with a maximum capacity of $100 m^3$ each.

$$\begin{aligned}
 X_1 = \begin{bmatrix} 3,1,1,2,2,1,2,1,2,1,1,2,1,2,2,1,3,1,3,1 \\ 2,1,1,2,2,1,1,1,2,1,1,1,2,1,1,3,1,2,1 \\ 1,3,3,1,2,1,2,1,3,1,1,1,2,1,1,3,1,1,1,2 \\ 1,2,2,1,1,1,2,1,3,1,1,1,2,1,1,2,1,1,1,2 \\ 2,1,3,1,1,1,1,2,1,2,1,1,2,1,3,1,1,3,2,1 \\ 2,1,3,1,1,1,1,2,1,2,1,1,2,1,2,1,1,3,1,1 \\ 1,3,2,1,2,1,1,3,1,1,3,1,1,1,2,1,1,3,1,1 \\ 1,2,2,1,2,1,1,3,1,1,2,1,1,1,1,1,1,3,1,1 \\ 2,1,1,3,1,1,1,1,2,1,3,1,2,1,1,1,3,1,2,1 \\ 2,1,1,2,1,1,1,1,2,1,2,1,2,1,1,1,2,1,1,1 \\ 1,3,1,2,1,1,2,1,1,2,1,1,2,1,2,1,1,3,1,2 \\ 1,1,2,1,1,1,2,1,2,1,2,1,1,2,1,2,3,1,1,2 \\ 1,2,1,2,2,1,2,1,1,1,1,1,2,1,3,2,1,3,1 \\ 1,1,2,1,1,1,2,1,2,1,2,1,1,1,2,1,2,1,2 \\ 1,1,1,2,1,2,1,2,1,3,1,2,2,1,3,1,2,1,2,1 \end{bmatrix}, X_2 = \begin{bmatrix} 3,1,1,2,2,1,2,1,2,1,1,2,1,2,3,1,3,1,3,1 \\ 2,1,1,2,2,1,1,1,2,1,1,1,2,1,1,3,1,2,1 \\ 1,3,3,1,2,1,2,1,3,1,1,1,2,1,1,3,1,1,1,2 \\ 1,2,2,1,1,1,2,1,3,1,1,1,2,1,1,2,1,1,1,2 \\ 2,1,3,1,1,1,1,2,1,2,1,1,2,1,3,1,1,3,2,1 \\ 2,1,2,1,1,1,1,2,1,2,1,1,2,1,2,1,1,3,1,1 \\ 1,3,2,1,2,1,1,3,1,1,3,1,1,1,2,1,1,3,1,1 \\ 1,2,2,1,2,1,1,3,1,1,2,1,1,1,1,1,1,3,1,1 \\ 2,1,1,3,1,1,1,1,2,1,3,1,2,1,1,1,3,1,2,1 \\ 2,1,1,2,1,1,1,1,2,1,2,1,2,1,1,1,2,1,1,1 \\ 1,3,1,2,1,1,2,1,1,2,1,1,2,1,2,1,1,3,1,2 \\ 1,1,2,1,1,1,2,1,2,1,2,1,1,2,1,2,3,1,1,2 \\ 1,2,1,2,2,1,2,1,1,1,1,1,2,1,3,2,1,3,1 \\ 1,1,2,1,2,1,2,1,2,1,2,1,1,1,2,1,2,1,2 \\ 1,1,1,2,1,2,1,2,1,3,1,2,2,1,3,1,2,1,2,1 \end{bmatrix}
 \end{aligned}
 \tag{A.4}$$

$$\begin{aligned}
 X_3 = \begin{bmatrix} 3,1,1,2,2,1,2,1,2,1,1,2,1,2,4,1,3,1,3,1 \\ 2,1,1,2,2,1,1,1,2,1,1,1,2,1,1,3,1,2,1 \\ 1,3,3,1,2,1,2,1,3,1,1,1,2,1,1,3,1,1,1,2 \\ 1,2,2,1,1,1,2,1,3,1,1,1,2,1,1,2,1,1,1,2 \\ 2,1,3,1,1,1,1,2,1,2,1,1,2,1,3,1,1,3,2,1 \\ 2,1,4,1,1,1,1,2,1,2,1,1,2,1,2,1,1,3,1,1 \\ 1,3,2,1,2,1,1,3,1,1,3,1,1,1,2,1,1,3,1,1 \\ 1,2,2,1,2,1,1,3,1,1,2,1,1,1,1,1,1,3,1,1 \\ 2,1,1,3,1,1,1,1,2,1,3,1,2,1,1,1,3,1,2,1 \\ 2,1,1,2,1,1,1,1,2,1,2,1,2,1,1,1,2,1,1,1 \\ 1,3,1,2,1,1,2,1,1,2,1,1,2,1,2,1,1,3,1,2 \\ 1,1,2,1,1,1,2,1,2,1,2,1,1,2,1,2,3,1,1,2 \\ 1,2,1,2,2,1,2,1,1,1,1,1,2,1,3,2,1,3,1 \\ 1,1,2,1,4,1,2,1,2,1,2,1,1,1,2,1,2,1,2 \\ 1,1,1,2,1,2,1,2,1,3,1,2,2,1,3,1,2,1,2,1 \end{bmatrix}, X_4 = \begin{bmatrix} 3,1,1,2,2,1,2,1,2,1,1,2,1,2,1,1,3,1,3,1 \\ 2,1,1,2,2,1,1,1,2,1,1,1,2,1,1,3,1,2,1 \\ 1,3,3,1,2,1,2,1,3,1,1,1,2,1,1,3,1,1,1,2 \\ 1,2,2,1,1,1,2,1,3,1,1,1,2,1,1,2,1,1,1,2 \\ 2,1,3,1,1,1,1,2,1,2,1,1,2,1,3,1,1,3,2,1 \\ 2,1,1,1,1,1,1,2,1,2,1,1,2,1,2,1,1,3,1,1 \\ 1,3,2,1,2,1,1,3,1,1,3,1,1,1,2,1,1,3,1,1 \\ 1,2,2,1,2,1,1,3,1,1,2,1,1,1,1,1,1,3,1,1 \\ 2,1,1,3,1,1,1,1,2,1,3,1,2,1,1,1,3,1,2,1 \\ 2,1,1,2,1,1,1,1,2,1,2,1,2,1,1,1,2,1,1,1 \\ 1,3,1,2,1,1,2,1,1,2,1,1,2,1,2,1,1,3,1,2 \\ 1,1,2,1,1,1,2,1,2,1,2,1,1,2,1,2,3,1,1,2 \\ 1,2,1,2,2,1,2,1,1,1,1,1,2,1,3,2,1,3,1 \\ 1,1,2,1,3,1,2,1,2,1,2,1,1,1,2,1,2,1,2 \\ 1,1,1,2,1,2,1,2,1,3,1,2,2,1,3,1,2,1,2,1 \end{bmatrix}
 \end{aligned}$$

$$D_{30} = D_{60} = D_{90} = \begin{cases} [60,50,60,70,50,50,60,60,50,60], \\ [50,50,60,60,60,60,50,70,70,50], \\ [70,60,60,50,70,70,60,50,50,50], \\ [50,70,70,50,60,60,60,50,60,70] \end{cases}
 \tag{A.5}$$

The description of the 15 jobs is shown in Table A.4, since there is only one machine at each stage in the process, the column with the processing routes was shortened for the sake of brevity as all the stages in the job use machine I . The amount of product (q_j) produced from each job is the same as in the case study 2. The main difference in the jobs is the extension in the number of machines and stages in the plant.

Table A. 4: Paths for case study of flow-shop.

Product	Jobs in J	Processing route K_j	Output (q_j)
Pr1	job_1	$p_{1,1}^1 \rightarrow p_{1,2}^1 \rightarrow p_{1,3}^1 \rightarrow \dots \rightarrow p_{1,19}^1 \rightarrow p_{1,20}^1$	70
Pr1	job_2	$p_{1,1}^2 \rightarrow p_{1,2}^2 \rightarrow p_{1,3}^2 \rightarrow \dots \rightarrow p_{1,19}^2 \rightarrow p_{1,20}^2$	50
Pr2	job_3	$p_{1,1}^3 \rightarrow p_{1,2}^3 \rightarrow p_{1,3}^3 \rightarrow \dots \rightarrow p_{1,19}^3 \rightarrow p_{1,20}^3$	80
Pr2	job_4	$p_{1,1}^4 \rightarrow p_{1,2}^4 \rightarrow p_{1,3}^4 \rightarrow \dots \rightarrow p_{1,19}^4 \rightarrow p_{1,20}^4$	60
Pr3	job_5	$p_{1,1}^5 \rightarrow p_{1,2}^5 \rightarrow p_{1,3}^5 \rightarrow \dots \rightarrow p_{1,19}^5 \rightarrow p_{1,20}^5$	80
Pr3	job_6	$p_{1,1}^6 \rightarrow p_{1,2}^6 \rightarrow p_{1,3}^6 \rightarrow \dots \rightarrow p_{1,19}^6 \rightarrow p_{1,20}^6$	50
Pr4	job_7	$p_{1,1}^7 \rightarrow p_{1,2}^7 \rightarrow p_{1,3}^7 \rightarrow \dots \rightarrow p_{1,19}^7 \rightarrow p_{1,20}^7$	70
Pr4	job_8	$p_{1,1}^8 \rightarrow p_{1,2}^8 \rightarrow p_{1,3}^8 \rightarrow \dots \rightarrow p_{1,19}^8 \rightarrow p_{1,20}^8$	50
Pr5	job_9	$p_{1,1}^9 \rightarrow p_{1,2}^9 \rightarrow p_{1,3}^9 \rightarrow \dots \rightarrow p_{1,19}^9 \rightarrow p_{1,20}^9$	70
Pr5	job_{10}	$p_{1,1}^{10} \rightarrow p_{1,2}^{10} \rightarrow p_{1,3}^{10} \rightarrow \dots \rightarrow p_{1,19}^{10} \rightarrow p_{1,20}^{10}$	60
Pr6	job_{11}	$p_{1,1}^{11} \rightarrow p_{1,2}^{11} \rightarrow p_{1,3}^{11} \rightarrow \dots \rightarrow p_{1,19}^{11} \rightarrow p_{1,20}^{11}$	80
Pr7	job_{12}	$p_{1,1}^{12} \rightarrow p_{1,2}^{12} \rightarrow p_{1,3}^{12} \rightarrow \dots \rightarrow p_{1,19}^{12} \rightarrow p_{1,20}^{12}$	70
Pr8	job_{13}	$p_{1,1}^{13} \rightarrow p_{1,2}^{13} \rightarrow p_{1,3}^{13} \rightarrow \dots \rightarrow p_{1,19}^{13} \rightarrow p_{1,20}^{13}$	80
Pr9	job_{14}	$p_{1,1}^{14} \rightarrow p_{1,2}^{14} \rightarrow p_{1,3}^{14} \rightarrow \dots \rightarrow p_{1,19}^{14} \rightarrow p_{1,20}^{14}$	70
Pr10	job_{15}	$p_{1,1}^{15} \rightarrow p_{1,2}^{15} \rightarrow p_{1,3}^{15} \rightarrow \dots \rightarrow p_{1,19}^{15} \rightarrow p_{1,20}^{15}$	80

The training showed that the agent managed to fulfill the demands of the process by introducing products considering the uncertainty in the processing times. Nevertheless, it was found that in 88.07% of the tests, there were violations of the allocation constraint. From an evaluation of the final policy π^* with different instances of the process, it was found that on average the agent produced 2 decisions per episode that produced, on average 4 allocation violations. This number of decisions represents almost 1.33% of the total number of decisions taken by the agent during the horizon. Fig. A.2 shows the reduction of violations to the allocation constraint from the beginning of the training until episode 3,000. As shown in this figure, the RL strategy was able to reduce the number of violations but after 2,750 episodes the number remained consistent. As in Fig. 18 and Fig. 21, the training shows that the agent has already converged into a policy that might not change significantly with additional training.

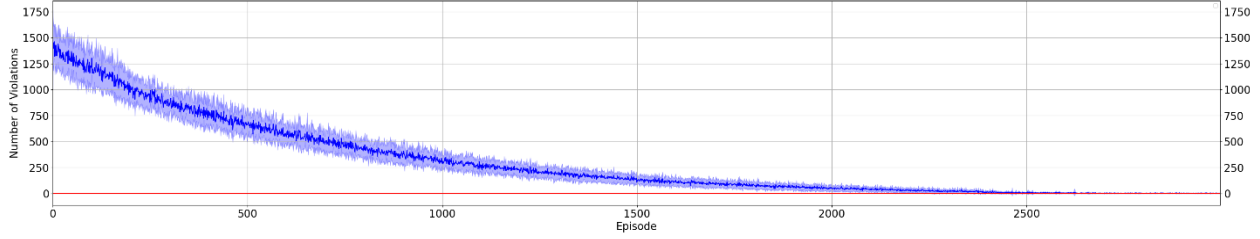


Figure A 2 Average reduction of allocation constraint violation during multiple trainings.

A schedule built with the final policy is presented in Fig. A.3 where the agent managed to complete the demand without any constraint violation. This instance was extracted from a test for evaluating the policy where the agent did not commit any violation during the process. For this instance, the vectors for updating the demand were $\mathbf{d}_{t=30} = [60,50,60,70,50,50,60,60,50,60]$, $\mathbf{d}_{t=60} = [50,50,60,60,60,60,50,70,70,50]$, and $\mathbf{d}_{t=90} = [50,50,60,60,60,60,50,70,70,50]$. There was a surplus of production equivalent to 15% of the total demand, which was the average overproduction found when the policy was evaluated.

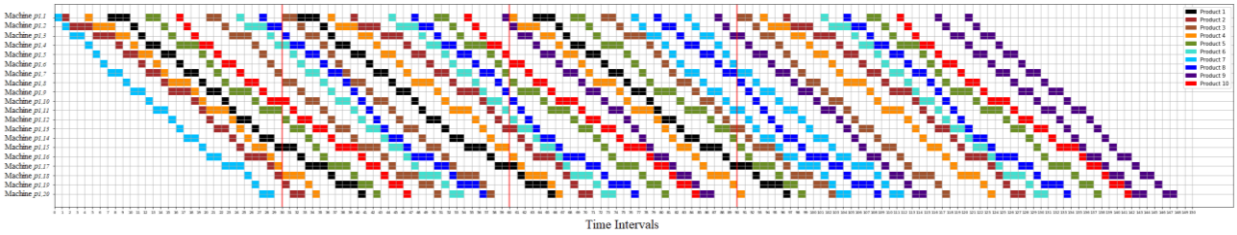


Figure A 3 Flow-shop schedule under uncertainty.

The results are consistent with the behavior of the agents that were developed for cases 1 and 2 in the instances that were subject to uncertainty. From these results, it can be concluded that the agent developed a policy that learned the processing times from each job and considered the uncertainty in certain machines for the decision-making process. The training of 10,000 episodes of the agent took 18,157 seconds and convergence was achieved after 5,447.1 seconds. After the agent was fully trained, the time it took to provide an action for a given $input_t$ was of $1.086e-3$ seconds on average.

A.4. MDP Approach for Solving Case 1 with Uncertainty

The DQN algorithm is similar to the DRQN in the methods to collect data, on the use of a target network, and in the updates for the neural network. As a comparison to the POMDP method shown to solve

the case studies, an MDP is set to solve the problem presented in Section 3.3.2 which consists of the job-shop with four products. The problem is the same and the changes are done in the method which now is a DQN. The construction of the agent consists of an FNN with two hidden layers of 100 nodes each; the network uses $\text{Tanh}(x)$ as an activation function. A new configuration of the state that was similar to the \mathbf{O}_t was defined to provide information to the agent. The state vector is described in Eq. A.6.

$$\mathbf{O}_t = [o_{1,1}, o_{2,2}, o_{3,3} \dots, o_{s,1}, \dots, o_{1,j-1}, o_{2,j-1}, o_{3,j-1} \dots, o_{s,j-1}, o_{1,j}, o_{2,j}, o_{3,j} \dots, o_{s,j}, z_1, z_2, \dots, z_i, t] \quad (\text{A.6})$$

The input integrates some of the elements that correspond to the original input vector from the DRQN method which include the demand left for each product (z_i) and the time t . The element $o_{s,j}$ (which differs from the element $o_{s,t}$) provides the time that a machine has been in use from the stage s in the job j . The number of rows is equal to the total number of jobs, each one corresponding to the machine usage from a particular job. For instance, if job 1 is initialized, the machine at stage 1, i.e., $o_{1,1}$ will be set to 1, and after 2 more time-intervals in use, this value will change to 3. When the machine has finished and the process for the job continues in another machine from another stage, the value for stage 1 changes to -1. If job 1 is not in process, then the first row is filled with zeros.

The output of the agent remains the same as in the DRQN, i.e., the Q-values for the available actions. The reward and the masking technique also remain the same for the DQN method, only the state and the architecture are the primary changes (apart from the DRL algorithm). Also, the first four hyperparameters for training presented in Table A.2 were applied equally to the learning process. The activation function hyperbolic tangent was applied in all the layers except the output layer.

The same number of episodes were used for the training of this agent. Fig. A.4 shows the average number of allocation constraint violations over multiple trainings. In the figure, it is evident that the agent reduces the number of constraint violations which demonstrates continuous learning of the process. Nevertheless, it

fails to reduce the number to values lower than 20 which compared to the DRQN represents an increase of more than four times the number of violations per episode.

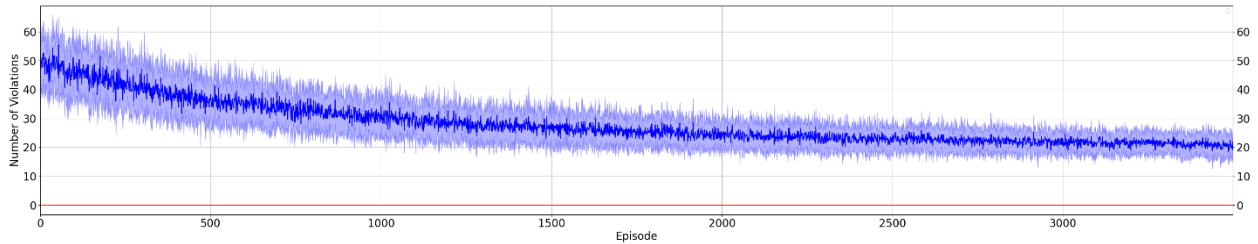


Figure A 4 Average violation of allocation constraints at each episode over multiple trainings.

The DQN method showed a worse performance than the DRQN because the information provided in the state is not enough for the agent to perceive the conditions of the plant. The information missing in the state that is present in the observation of the DRQN is not added because of the temporal limitations of the MDP, i.e., it does not allow more than the present information on the process.

A.5. MDP Approach for Solving Case 1 with a Reduced State

In this section an evaluation of the proposed framework for case study 1 under uncertainty is presented. For this assessment, a part of the information in some of the observations in the observation window are deleted. With this experiment it was aimed to prove the capacity of our method for evaluating the environment when the observations are deficient or inaccurate. Although there was a reduction in the maximum reward, results showed that the agent developed skills for reducing the demand while avoiding violations of constraints. All the hyperparameters reported in Table A.2 were set in this experiment. The training was done with 3,500 episodes.

The reduction of observations consisted in substituting 60% of the information of the observation with a zero value; this was applied only on observation vectors $(o_{s,t})$ collected in even time intervals (i.e., 2, 4, 6, ...). The substituted values correspond to the information given for a specific action registered in the observation (see description in section 3.2). In other words, the registered information related to the occupation of the machines for a particular job initialized in the past is *converted to zero* in the observation.

The jobs which are reduced are selected randomly. Note that since the observations registered in odd time intervals are not altered with this mechanism, the agent will still have access to the evolution in time of these actions.

Fig. A.5 shows the average number of constraint violations done in an episode. It was found that the agent was still capable of reducing the number of overlapping machines in the system to less than 10 during the horizon. In the case of the demands, it was found that the trained agent would not meet between 0 and 10% of the total demand in the processes where it was tested. This means that for all scenarios, production will cover at least 90% of the demands made to the agent.

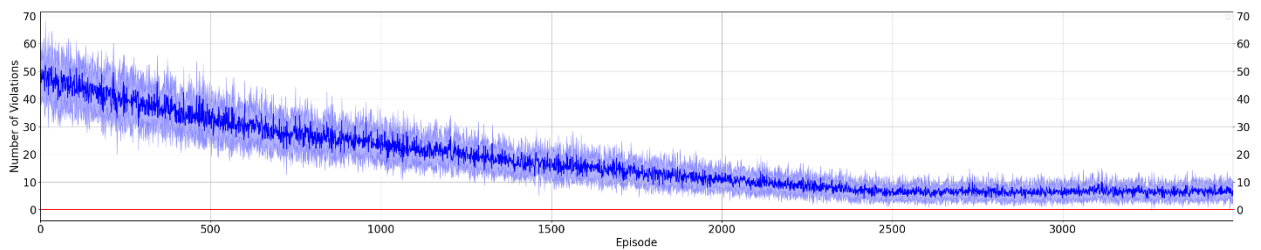


Figure A 5 Average violation of allocation constraints at each episode over multiple trainings.

Although the method showed a lower performance for this experiment, it should be pointed out that the agent was still capable of extracting information from the observation window. With this experiment the capacity of the proposed method to maintain adaptability was demonstrated despite the reduction of information. For real case scenarios, this interruption of information can be related to inaccuracies in the data or problems in communication between the planner and the systems.

Appendix B.

B.1. Specifications of Case Study 1

Table B 1 Hyperparameters

Hyperparameter	Value
Learning Rate Annealing	1e-4→1e-6
Temperature Annealing	1→0.1
Variance Annealing	1e-2→1e-5
Entropy Annealing	1e-2→1e-3
Gamma	0.99
Updates per epoch	5 to 10
Episodes per epoch	10
Episodes per training	6e3 to 10e3
Number of Nodes in hidden layers	250
Hidden State size (LSTMs)	25
Lambda (GAE)	0.98
Parameter clipping	0.2
Minibatches for updates	5

Table B 2 Relations between tasks, units, and processing times.

Task	Type	Available units	Processing time-intervals
Task 1	Heating	Unit 1: Heater, cap.: 100 units	2 intervals
Task 2	Reaction 1	Unit 2: Reactor 1, cap.: 80 units Unit 3: Reactor 2, cap.: 50 units	4 intervals
Task 3	Reaction 2	Unit 2: Reactor 1, cap.: 80 units Unit 3: Reactor 2, cap.: 50 units	4 intervals
Task 4	Reaction 3	Unit 2: Reactor 1, cap.: 80 units Unit 3: Reactor 2, cap.: 50 units	2 intervals
Task 5	Separation	Unit 4: Distillation column, cap.: 200 units	2 intervals

Table B 3 Relations between tasks and states.

Task	Input states	Output states
Task 1	Feed State A, 100%	State D, 100%
Task 2	State D, 40% State F, 60%	State E, 40% State H, 60%
Task 3	Feed State B, 50% Feed State C, 50%	State F, 100%
Task 4	State C, 20% State H, 80%	State G, 100%
Task 5	State G, 100%	State I, 90% State H, 10%

Table B 4 Notation for problem statement P2.

I_j	Set of tasks which can be performed by unit j .
p_i	Completion time for task i .
V_{ij}^{min}	Minimum capacity of unit j when used for performing task i .
V_{ij}^{max}	Maximum capacity of unit j when used for performing task i .
S_{st}	Amount of material stored in state s , at the beginning of t .
C_s	Maximum storage capacity dedicated to state s .
T_s	Set of tasks receiving material from state s .
\bar{T}_s	Set of tasks producing material in state s .
ρ_{is}	Proportion of input of task i from state $s \in S_i$.
$\bar{\rho}_{is}$	Proportion of output of task i to state $s \in \bar{S}_i$.
K_i	Set of units capable of performing task i .

Table B 5 Values for r_q for each task.

Task	Weight
Task 1	50
Task 2	125
Task 3	100
Task 4	300
Task 5	1600

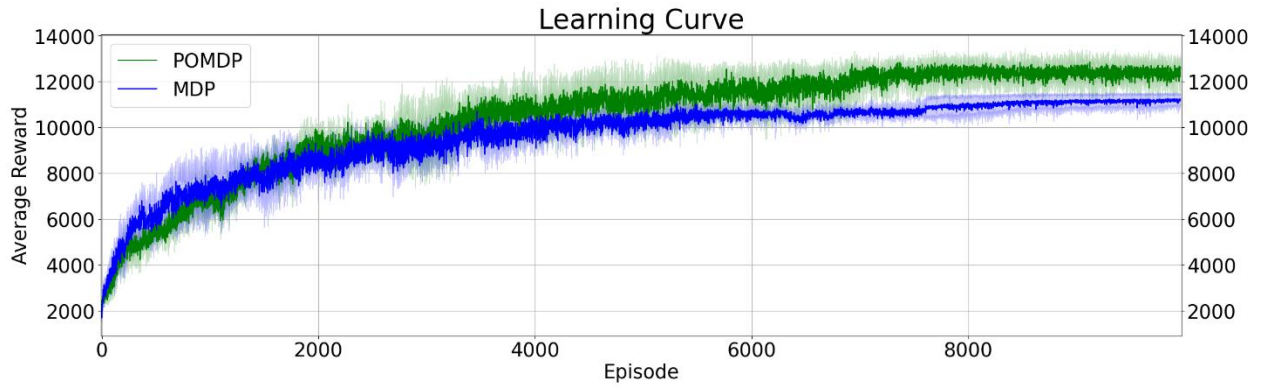


Figure B 1 Learning curve of the deterministic problem

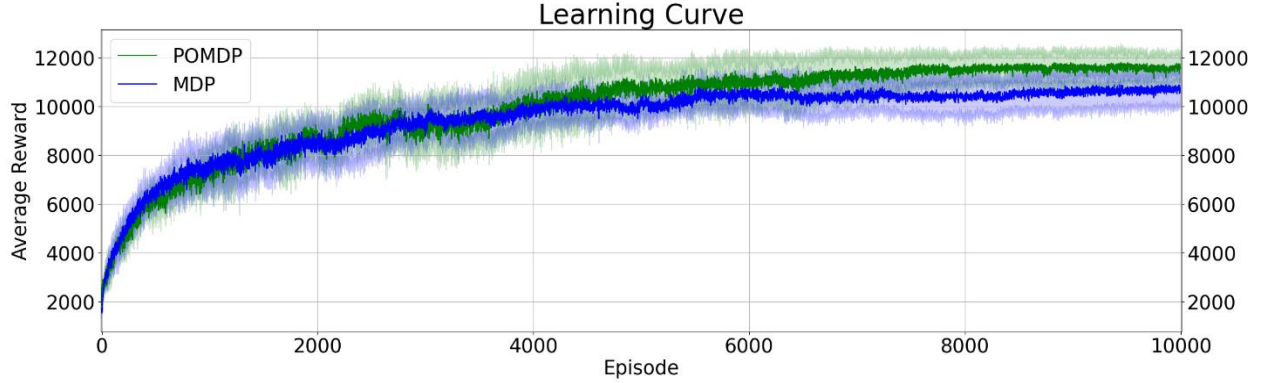


Figure B 2 Learning curve for case with Gaussian noise in the observations.

B.2. Specifications of Case Study 2

There are 6 units available that are shared by some tasks as in the previous example, Table B6 describes these relations and provides the processing times by each task. The fourth column from this table describes the auxiliary service required by each task; a total of three types of auxiliary services are available, namely Cold Water (CW), Low-Pressure Service (LPS), and High-Pressure Service (HPS). The mass flow from the required service is dependent on the capacity determined by continuous action. Each service has a maximum capacity that must not be overpassed by the decisions of the agent, these limits are: $CW = 25 \text{ kg/min} \cdot \text{kg}$, $LPS = 40 \text{ kg/min} \cdot \text{kg}$, and $HPS = 20 \text{ kg/min} \cdot \text{kg}$.

Table B 6 Relations between units and tasks in Case Study 2.

Task	Available units and capacities	Processing time-intervals	Service required (per kg of batch)
Task 1	Unit 1: cap.: 5 units	2 intervals	2 kg/min LPS
Task 2	Unit 2: cap.: 8 units	1 interval	2 kg/min CW
Task 3	Unit 3: cap.: 6 units	1 interval	3 kg/min LPS
Task 4	Unit 1: cap.: 5 units	2 intervals	2 kg/min HPS
Task 5	Unit 4: cap.: 8 units	2 intervals	4 kg/min LPS
Task 6	Unit 4: cap.: 8 units	2 intervals	3 kg/min HPS
Task 7	Unit 5: cap.: 3 units	4 intervals	4 kg/min CW
Task 8	Unit 6: cap.: 4 units	2 intervals	3 kg/min LPS
Task 9	Unit 5: cap.: 3 units	2 intervals	3 kg/min CW
Task 10	Unit 6: cap.: 4 units	3 intervals	3 kg/min CW

Table B 7 Weights of rewards for case study 2.

Task	Weight
Task 1	10
Task 2	80
Task 3	80
Task 4	40
Task 5	80
Task 6	80
Task 7	50
Task 8	60
Task 9	60
Task 10	70

Appendix C.

C.1. Example for defining an attention matrix

To show a simple representation of the attention matrix $\mathbf{A}_{t,k}$, assume a batch plant with two tasks which correspond to a batch reactor and a filter. The plant considers three states: one that feeds the reactor and another two which store the product and the waste from the filtering process, respectively. Fig. C1 shows the STN representation of such a plant model. Assume $\Gamma = 2$, i.e., two consecutive matrices $\mathbf{\Xi}_t$ are sent to the agent at every time-interval. A node is assigned for each state (i.e., A, P1, and Waste) and task (i.e., Reaction and Filtering) in the network, then there are five nodes in the plant model; nevertheless, the observation window \mathbf{O}_t includes the nodes from the two time-intervals considered, then the total number of nodes in the \mathbf{O}_t is ten (i.e., $\Gamma\zeta = 2 \times 5 = 10$). Each node has η features that are used to describe the state of that node. For this simple case, three features are defined: 1) the amount of material in the node, 2) the time it has passed without change, and 3) the concentration of impurities in the product. A time code composed of two values is added into the features of the observation window.

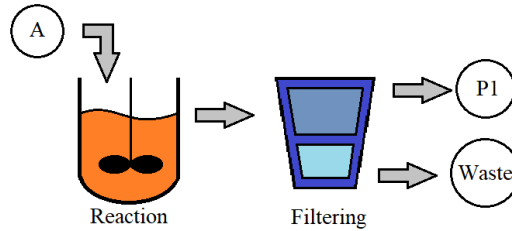


Figure C 1 STN representation of reaction-filtering process

The expected $\mathbf{A}_{t,k}$ and \mathbf{B}_k for this instance will have a dimension of 10×10 . Fig. C2 provides a representation of $\mathbf{A}_{t,k}$ for an agent with one head. The rows show the nodes at the two time-intervals when the information is registered, in this case time t and time $t - 1$. Then, for each node there is a connection with the other nodes and itself which is reflected in the columns. The columns are also grouped in time-intervals. Each value $a_{t,t',i,j}$ represents the relevance of the interaction between the node i at a time-interval t with the node j at a time-interval t' . The values $a_{t,t',i,j}$ in each row from Fig. C2 add up to one due to the

SoftMax activation function. Reading the matrix is an easy task, for instance, to see the level of attention that is paid by the Filter at time t to the State P1 at time $t - 1$, one would have to see the value of $a_{t,t-1,3,4}$.

Note that the attention that the nodes pay to themselves at different times happens when $i = j$.

		Time t-1					Time t				
		Feed (1)	Reactor (2)	Filter (3)	P1 (4)	Waste (5)	Feed (1)	Reactor (2)	Filter (3)	P1 (4)	Waste (5)
Time t-1	Feed (1)	$a_{t-1,t-1,1,1}$	$a_{t-1,t-1,1,2}$	$a_{t-1,t-1,1,3}$	$a_{t-1,t-1,1,4}$	$a_{t-1,t-1,1,5}$	$a_{t-1,t,1,1}$	$a_{t-1,t,1,2}$	$a_{t-1,t,1,3}$	$a_{t-1,t,1,4}$	$a_{t-1,t,1,5}$
	Reactor (2)	$a_{t-1,t-1,2,1}$	$a_{t-1,t-1,2,2}$	$a_{t-1,t-1,2,3}$	$a_{t-1,t-1,2,4}$	$a_{t-1,t-1,2,5}$	$a_{t-1,t,2,1}$	$a_{t-1,t,2,2}$	$a_{t-1,t,2,3}$	$a_{t-1,t,2,4}$	$a_{t-1,t,2,5}$
	Filter (3)	$a_{t-1,t-1,3,1}$	$a_{t-1,t-1,3,2}$	$a_{t-1,t-1,3,3}$	$a_{t-1,t-1,3,4}$	$a_{t-1,t-1,3,5}$	$a_{t-1,t,3,1}$	$a_{t-1,t,3,2}$	$a_{t-1,t,3,3}$	$a_{t-1,t,3,4}$	$a_{t-1,t,3,5}$
	P1 (4)	$a_{t-1,t-1,4,1}$	$a_{t-1,t-1,4,2}$	$a_{t-1,t-1,4,3}$	$a_{t-1,t-1,4,4}$	$a_{t-1,t-1,4,5}$	$a_{t-1,t,4,1}$	$a_{t-1,t,4,2}$	$a_{t-1,t,4,3}$	$a_{t-1,t,4,4}$	$a_{t-1,t,4,5}$
	Waste (5)	$a_{t-1,t-1,5,1}$	$a_{t-1,t-1,5,2}$	$a_{t-1,t-1,5,3}$	$a_{t-1,t-1,5,4}$	$a_{t-1,t-1,5,5}$	$a_{t-1,t,5,1}$	$a_{t-1,t,5,2}$	$a_{t-1,t,5,3}$	$a_{t-1,t,5,4}$	$a_{t-1,t,5,5}$
Time t	Feed (1)	$a_{t,t-1,1,1}$	$a_{t,t-1,1,2}$	$a_{t,t-1,1,3}$	$a_{t,t-1,1,4}$	$a_{t,t-1,1,5}$	$a_{t,t,1,1}$	$a_{t,t,1,2}$	$a_{t,t,1,3}$	$a_{t,t,1,4}$	$a_{t,t,1,5}$
	Reactor (2)	$a_{t,t-1,2,1}$	$a_{t,t-1,2,2}$	$a_{t,t-1,2,3}$	$a_{t,t-1,2,4}$	$a_{t,t-1,2,5}$	$a_{t,t,2,1}$	$a_{t,t,2,2}$	$a_{t,t,2,3}$	$a_{t,t,2,4}$	$a_{t,t,2,5}$
	Filter (3)	$a_{t,t-1,3,1}$	$a_{t,t-1,3,2}$	$a_{t,t-1,3,3}$	$a_{t,t-1,3,4}$	$a_{t,t-1,3,5}$	$a_{t,t,3,1}$	$a_{t,t,3,2}$	$a_{t,t,3,3}$	$a_{t,t,3,4}$	$a_{t,t,3,5}$
	P1 (4)	$a_{t,t-1,4,1}$	$a_{t,t-1,4,2}$	$a_{t,t-1,4,3}$	$a_{t,t-1,4,4}$	$a_{t,t-1,4,5}$	$a_{t,t,4,1}$	$a_{t,t,4,2}$	$a_{t,t,4,3}$	$a_{t,t,4,4}$	$a_{t,t,4,5}$
	Waste (5)	$a_{t,t-1,5,1}$	$a_{t,t-1,5,2}$	$a_{t,t-1,5,3}$	$a_{t,t-1,5,4}$	$a_{t,t-1,5,5}$	$a_{t,t,5,1}$	$a_{t,t,5,2}$	$a_{t,t,5,3}$	$a_{t,t,5,4}$	$a_{t,t,5,5}$

Figure C 2 Structure of the attention matrix

The bias matrix \mathbf{B}_k has the same dimension as $\mathbf{A}_{t,k}$ with the difference in the values of the former which are fixed during the training k . That is, the values from $\mathbf{A}_{t,k}$ change at every time-interval because it is dependent on the input values; however, \mathbf{B}_k works as an amplifier/attenuator of the attention values, and it affects in the same way all the attention matrices that are generated. Then, this matrix is forced to learn during the training the most relevant and irrelevant nodes in the environment such that it does not alter relevant information of the attention values.

C.2. Deterministic Case and Under Aleatoric Uncertainty

In this section the case instance from the literature is used to train a DRL agent that can build online schedules. The instance is solved in two scenarios: a) as a deterministic problem and b) as a scheduling problem subject to aleatoric uncertainty. First, the elements from the RL implementation are defined

following the methodology presented in Section 6.2, then the performance of the agent is compared for scenarios a) and b). The objective function of the scheduling problem aims to maximize the final products at States E and I at the end of the horizon H , see Eq. 37.

C.2.1. Environment

The plant instance is the same as that one presented on Fig. 27 with 5 tasks and 8 states. To make a comparison with the results from the *RNN approach* [18], most of the parameters from that problem were maintained in this case. The objective function remains as production maximization, and the action space is defined as the activation of the tasks and the capacity at which the unit is loaded. Two agents are trained in this environment: the first agent is trained in a deterministic version of the environment while the second is trained in an environment that is subject to aleatoric uncertainty which causes partial observability to the agent. The agent needs to build a policy that can generate a schedule in an online fashion given the conditions of the environment. The results from both implementations are benchmarked with the results from the *RNN approach* to compare the performance of the RNNs against the SAM.

C.2.2. Graph Representation

The environment is translated into nodes; the tasks are expanded into specific task-unit elements in the environment. The states that include intermediate storage, feeds, and products are also identified as nodes. The observation window for this case assumes a value of $\Gamma = 4$, then the values from all the nodes from previous three time-intervals and the current time-interval are reported to create the observation window \mathbf{O}_t , which concatenates the following observation matrices: $\Xi_{t-3}, \Xi_{t-2}, \Xi_{t-1}, \Xi_t$. Table C1 presents the complete list of nodes from the environment at one time-interval. Recall that a set of time-intervals is sent to the agent in the observation window, not presented for the sake of brevity.

Table C 1 Matrix representation of the nodes and features.

	U1	U2	U3	U4	Ref Cap	Cap	time
Task 1	1/0	0	0	0	Ref	0-1	0-1
Task 2 at Unit 1	0	1/0	0	0	Ref	0-1	0-1

Task 3 at Unit 1	0	1/0	0	0	Ref	0-1	0-1
Task 4 at Unit 1	0	1/0	0	0	Ref	0-1	0-1
Task 2 at Unit 2	0	0	1/0	0	Ref	0-1	0-1
Task 3 at Unit 2	0	0	1/0	0	Ref	0-1	0-1
Task 4 at Unit 2	0	0	1/0	0	Ref	0-1	0-1
Task 5	0	0	0	1/0	Ref	0-1	0-1
State D	0	0	0	0	Ref	0-1	0
State E	0	0	0	0	Ref	0-1	0
State F	0	0	0	0	Ref	0-1	0
State G	0	0	0	0	Ref	0-1	0
State H	0	0	0	0	Ref	0-1	0
State I	0	0	0	0	Ref	0-1	0

The features are listed as columns and the values that they could take are shown on the table. The first four columns (U1, U2, U3, and U4) refer to the four units used to process the task: Unit 1, Unit 2, Unit 3, and Unit 4. Tasks 1 and 5 are processed in Units 1 and 4, respectively, while Tasks 2, 3, and 4 can be processed either in Unit 2 or Unit 3. Since the states do not occupy the units, their value is set to zero at all times. The fifth feature (Ref Cap) is a reference value that also remains fixed (Ref) and aims to indicate the capacity of the task or state. This value is bound between 0 and 1 and is obtained by dividing the capacity of a node by the capacity of the largest unit (maximum capacity in the environment). In this way the matrix includes reference values from the capacities of the nodes. The column denoted as “Cap” describes the current capacity of the task or state. The value is represented as a fraction, i.e., this value is between 0 and 1, indicating the current amount of material that is in the task or state. The column “time” indicates the time that a task has been activated, the time is expressed as a fraction and is calculated by dividing the number of time-intervals that a task has been in progress by the total processing-time of such task. In this way, this column also reports fractional values as the previous one. The values of this column that correspond to the states are set to zero.

C.2.3. Results and Comparison

The learning curve from the training process of the deterministic agent is presented in Fig. C3, which is the average of three training courses of the same agent. The training for the current agent was done over

5000 episodes, while the RNNs were highly unstable in the training, the presented agent showed an improvement in stability during the training. The *RNN approach* required between 8,000 and 10,000 episodes of training to guarantee smooth training [18]. Shorter training for the RNNs derived in abrupt changes for the agent’s hyperparameters (recall the annealing methods incorporated for exploration and learning rate) which destabilize the training. The current method showed to be more stable even in trainings as short as 2,000 episodes but in these cases, there was not enough exploration to achieve convergence in the learning curve.

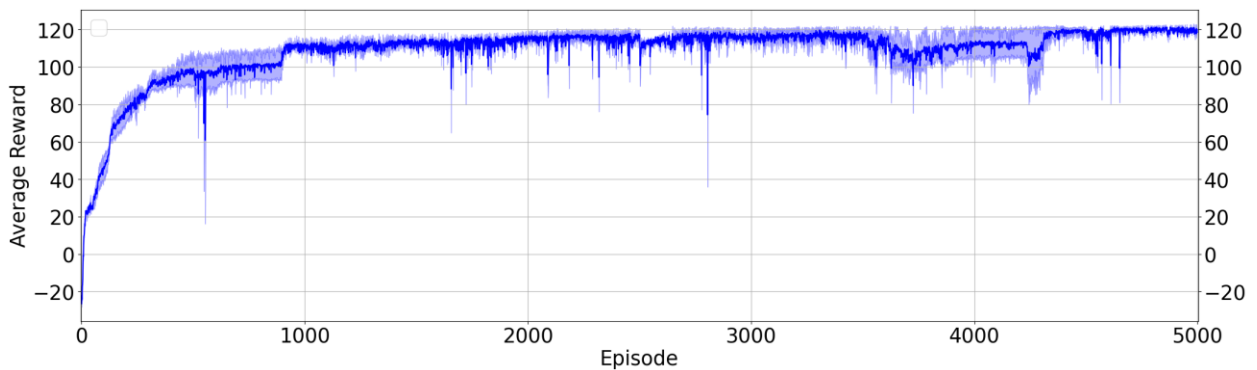


Figure C 3 Average learning curve of the agent in deterministic case

In comparison with the agent from the *RNN approach*, it was observed that although the two systems of rewards were equivalent, production was different for both agents. The production of the schedule from the RNN agent was 168 units of product P1 and 284 units of P2, while the agent from this experiment produced 159 units of P1 and 163 units of P2. This production represents 71% of the total production achieved by the RNN agent. On the other hand, no constraint violations were observed with the current agent whereas the RNN agent violated a constraint in 16.54% of all the decisions it made. Then, the agent with attention performed better at understanding which action-state might end in a penalty in the environment, which is a desirable feature to be considered in an online scheduling policy.

The agent was also tested in a case with aleatoric uncertainty affecting the communication channel between the agent and the environment. In even time-intervals ($t = 2, 4, 6, \dots$) the environment will send an observation in which the column-feature that shows the current amount of material at every node (see Table C1, Cap column) is modified. The modification consists of the aleatoric selection of 50% of the

elements in that column and setting their values to 0. Then the agent receives partial observations from the environment in a recurrent fashion, forcing the agent to focus on using the available information in the observation window. Fig. C4 shows the average of three training courses from the agent in the environment under partial observability. As shown in this figure, there are significant oscillations during the training; also, the variance is larger compared to Fig. C3. This variability in the training is due to the aleatoric uncertainty introduced in the observations, which was completely random for the agent. Increasing the time for exploration in the environment (i.e., increasing the number of episodes) allowed the agent to handle the partial observability in the input. Nevertheless, the agent managed to converge into a stable solution at the end of the training. Shorter training proved to be inefficient for learning, resulting in policy collapse or high instability in the resulting learning curves. In other words, the variance could not be handled resulting in a divergence of learning.

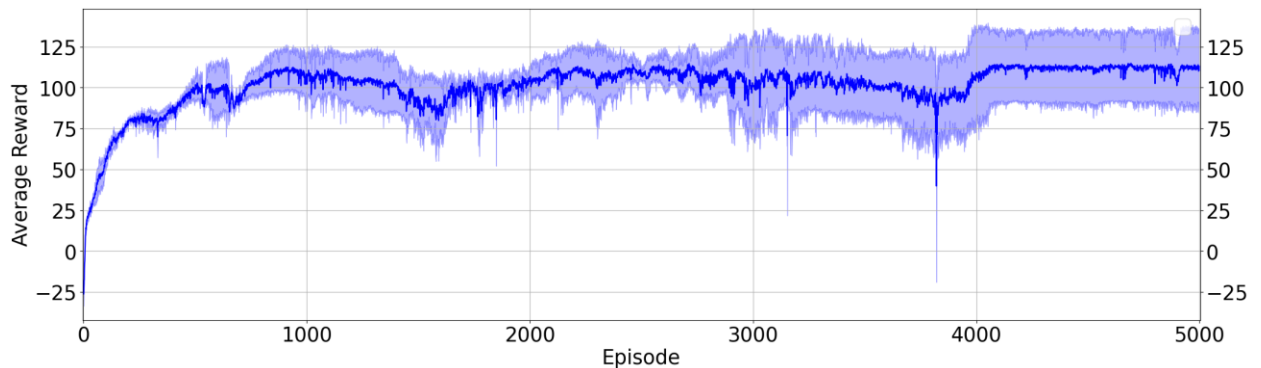


Figure C 4 Average learning curve for the agent with partial observation

The production of P1 and P2 for this case remained almost the same as in the deterministic case (i.e., without partial observation of the environment). Production of P1 was of 160 units and production of P2 was of 166 units, showing no apparent impacts from the partial observability. In the *RNN approach*, the RNNs presented a decrease in rewards of 19.35% from the deterministic experiment. To further validate the present DRL framework, an evaluation of 1,000 environments with different realizations of the partial observability was performed, for this case an average of 3 constraint violations per iteration was observed. Specifically, the allocation constraint was violated in Task 1. Out of the 62 decisions made by the agent at

every episode, this number of violations represent 4.84% of all the decisions made, which is a lower value compared to that from the RNN agent in the same conditions (15.19%).

C.3. Specification of Case Studies

Table C 2 Weights of the reward shaping method used in Case Study 1

Task	Weight
Task 1	0.5
Task 2	2.0
Task 3	1.0
Task 4	4.8
Task 5	16.0

Table C 3 Matrix representation of the nodes and features for Case Study 2 with the addition of feed states A, B, and C

	U1	U2	U3	U4	Ref Cap	Cap	time
Task 1	1/0	0	0	0	Ref	0-1	0-1
Task 2 at Unit 1	0	1/0	0	0	Ref	0-1	0-1
Task 3 at Unit 1	0	1/0	0	0	Ref	0-1	0-1
Task 4 at Unit 1	0	1/0	0	0	Ref	0-1	0-1
Task 2 at Unit 2	0	0	1/0	0	Ref	0-1	0-1
Task 3 at Unit 2	0	0	1/0	0	Ref	0-1	0-1
Task 4 at Unit 2	0	0	1/0	0	Ref	0-1	0-1
Task 5	0	0	0	1/0	Ref	0-1	0-1
State D	0	0	0	0	Ref	0-1	0
State E	0	0	0	0	Ref	0-1	0
State F	0	0	0	0	Ref	0-1	0
State G	0	0	0	0	Ref	0-1	0
State H	0	0	0	0	Ref	0-1	0
State I	0	0	0	0	Ref	0-1	0
State A	0	0	0	0	Ref	0-1	0
State B	0	0	0	0	Ref	0-1	0
State C	0	0	0	0	Ref	0-1	0