A Reinforcement Learning Framework for Online Batch Process Scheduling

Syu-Ning Johnn, Vassilis M. Charitopoulos \*

Department **of Chemical Engineering, The Sargent Centre for Process Systems Engineering, University College London, Torrington Place, London WC1E 7JE, United Kingdom**

\* v.charitopoulos@ucl.ac.uk

Abstract

Optimisation-based batch scheduling methods serve as a practical response strategy in the process industries as a coordination of planning and execution. The efficiency and adaptability of the method are of paramount importance, especially when dealing with frequent modifications to existing decisions caused by uncertainties and unforeseen real-world disturbances, with goal of achieving substantial financial profitability.

Reinforcement Learning, compared to many classic techniques, has the advantages of learning from existed experiment and generalise to unknown scenarios, and thus automating the process with higher flexibility and adaptability. In this work, we propose a RL-based method by transferring a batch process scheduling problem into a Markov Decision Process framework and train the agent to learn to build up task sequences to optimise the production schedule. The results show that our method achieves good computational efficiency and adaptability.

**Keywords**: Autonomous Online Scheduling, Reinforcement Learning (RL), Neural Network, Optimisation, Batch Processing.

* 1. Introduction

Batch process scheduling constitutes a fundamental problem in the process industries. Optimising the coordination of planning and execution is imperative for enhancing scheduling efficiency, yet it necessitates frequent modifications owing to unforeseen real-world disturbances. Diverse sources of uncertainty encompassing market demand fluctuations, due date adjustments, equipment reliabilities, and regulatory changes in a dynamic industrial environment pose the potential to swiftly render any pre-established optimal schedule inefficacious or obsolete (Gupta et al., 2016). As a result, optimisation-based scheduling methods serve as a practical response strategy deployed to address uncertainties with the aim of achieving substantial financial profitability.

When uncertainty is considered, existing approaches can be grouped into preventive or reactive based (Castro et al., 2018). Many studies in the literature aim to address the challenges associated with tackling the batch scheduling model and minimising the disparity between the theoretical models and their industrial applicability (Basán et al., 2020). In recent years, the integration of Machine Learning (ML) into process scheduling receives emerging attention (Hubbs et al., 2020). The ML techniques, especially those in the field of Reinforcement Learning (RL), have demonstrated the potential to effectively handle uncertainties with lower evaluation expense. In this work, we introduce deep Q-learning, a model-free RL technique, to address the batch process scheduling problem. We formulate the problem as a Markov Decision process (MDP), based on which the learning takes place to train a neural network as a function approximator to estimate the optimal decision for building up the task sequence to form a schedule. From our case study, our RL-based approach achieves good adaptability for online schedule optimisation while preserving reasonably high solution quality.

As a popular training approach, RL learns optimal strategies through trial-and-error by interacting with a dynamic environment, and unlike traditional rule-based systems, RL exhibits the potential to discover optimal decision-making strategies generalised from its learned experiences. In the literature, Wang et al. (2023) formulated a flow-shop scheduling problem as a MDP for a deep RL algorithm with long short-term memory network to capture state sequences. Yang et al. (2023) applied Graph Neural Network for learning representation in RL to address a dynamic scheduling problem with the objective of minimising makespan. Additionally, Ren et al. (2021) tackled the scheduling problem by transforming it into a directed graph and applied RL technique for multistage sequential decision-making, utilising a nonlinear state-and-action neural network to approximate the reward function.

* 1. Methodology
		1. Markov Decision Process

RL is a learning method mapping states to actions to maximise the expected future rewards (Sutton and Barto, 2018). Markov Decision Process (MDP) is a fundamental mathematical framework that models a sequential decision-making process, based on which we can apply the RL techniques.

An MDP can be represented by a tuple where a state is a representation of the specific time step within the environment that the agent can be in. At each non-terminal state , the agent selects an action from the set of actions available at this state and receives a reward according to a given reward function , which is the immediate benefit for taking the state-action pair. Afterwards, the agent reaches a new state based on the transition probability that describes the likelihood of moving from one state to another given an action. The key defining feature of an MDP is the Markov property, that the future state of the system is solely dependent on the preceding state-action pair, rendering the history of events irrelevant.

The agent engages with the environment through episodes, each of which encapsulates the agent's interactions over time as a sequential chain of states, actions, and rewards. By receiving feedback in the form of rewards from the environment, the agent is trained to refines its strategy with actions in dynamic states and formulate an optimal policy that dictates the agent's behaviour to maximise the cumulative future rewards. Central to this learning is the state-action value function also denoted as Q-values, which quantify the expected rewards associated with taking any particular state-action pair by following .

* + 1. Problem Formulation

We transform the batch process scheduling problem with continuous time into an episodic task trajectory, incorporating node insertion as the action. Figure 1 provides the visualisation of a sample process schedule involving 2 machines and 2 lines of tasks, which has a problem size equals 2 for simplicity. The schedule illustrates the execution sequence as well as the corresponding processing machine for all tasks. The notation for each task follows a specific format: (flow sequence index, machine index, copy index), representing the ranking of each task in its respective line, the assigned machine's index, and the copy index of the task, respectively. For instance, signifies the first task in the sequence within an individual production line that is processed by the second machine, and this task is the third copy of its kind in the entire production process. All tasks scheduled on the same line are sequentially connected, and all tasks assigned to the same machine must be processed with no time overlapping.

Bringing in the MDP terminology, each state is a tuple , wherein the graph contains all the existing tasks (represented by nodes) and the sequence between them (represented by arcs), the feature matrix contains information for the node process sequence, the number of copies, and the starting time for each task. The starting time for each task is computed by if assigned to any machines that is not the first to run, and for all tasks assigned to the first machine. Finally, is a Boolean variable indicating whether a state is a terminal one. The agent's set of actions involves inserting nodes with increasing copy index into the feasible arc within the graph, ensuring compliance with machine availability capacity and time restrictions imposed by the given time horizon. Notably, we require that any additional copy cannot occur earlier than the existing copy .



(Left) Figure 1. A sample batch process problem of size 2; (Right) Figure 2. Batch process in a MDP framework as a directed graph from start node to end node .

The upper row in Figure 2 illustrates part of an episodic task spanning 3 sequential episodes, where the agent iteratively makes 3 node insertions according to its learnt policy. The bottom row depicts the timeline with the task schedules, categorised by the allocated machine. We start from the left-most column of graphs where there are four tasks, each of which has a single copy. We assume an inventory consumption of 1 unit for each task, and the processing times are at 1.5h for both tasks on machine 1, and for machine 2 the tasks are set to be 1h and 2h.

In the leftmost upper graph in Figure 2, the initial sequence can be written as , with the list of starting times as . Based on flow balance and machine availability, the agent at this state is given two actions: to insert another copy of either or into the graph. The agent selects , creating as a new copy of task and transitioning to the second graph with a processing sequence of and an updated processing time . At this state, the agent is presented with the options to insert another copy of , or into the graph and it chooses to create another copy of task , hence transitioning to the third graph. This process continues until reaching a terminal state where no additional node can be inserted without violating the machine availability or inventory flow balance. The reward for the episode is computed as the improvement in total production at the terminal state compared to the initial state using the reward function , and for the non-terminal states.

* + 1. Q-Learning

Q-learning (Watkins and Dayan, 1992) is a popular model-free RL approach employed to solve MDPs by iteratively updating the Q-values according to the rule:

 (1)

where is the previous Q-value, the term denotes the maximum Q-value amongst all the available actions in the subsequent state, and represents the immediate reward the agent receives by performing the pair. The discount factor balances immediate rewards against future rewards, influencing the agent's preference for short-term gains versus long-term benefits. Through the iterative refinement in the MDP, the agent refines its policy to estimate the optimal Q-values.

Deep neural networks are frequently utilised as function approximators for the function, which facilitates the generalisation of Q-values across states that share common characteristics and therefore may consequently yield similar future rewards. Function approximation is particularly advantageous for problems with large state spaces, which contrasts with the traditional Q-table approach: Unlike Q-tables that explicit record Q-values for every state-action combination, DQN provides a more scalable and efficient means of approximating the Q-values by bypassing the memory challenges associated with maintaining extensive records.

We employ the Deep Q-Network (DQN) by Mnih et al. (2015) for agent training, incorporating key techniques including the replay buffer and target network. During training, the DQN algorithm samples iterative batches of previously taken actions from the replay buffer to train the main network. In this step, the algorithm takes the current state of the environment as input and outputs a vector of Q-values corresponding to the list of available actions. The main network parameters are periodically transferred to the target network at every certain number of episodes to stabilise the training process. The target network remains consistent over this certain interval and is used to predict the future Q-values for state-action pairs.

* 1. Case Study
		1. Experiment Setup

For the neural network in the DQN algorithm, the main network parameters are updated by stochastic gradient descent using the Q-learning loss. We employ 3 hidden layers with the first layer containing 512 neurons and half the size for the subsequent layers, followed by a ReLU activation function. The feature matrix inputted into the network encompasses task-related information for each state, including the node sequence, number of copies for each task, and the starting time for each task. Prior to input for the training, the feature matrix associated with each state is flattened into a vector, undergoes one-hot encoding, and normalised using the min-max normalisation function.

The training steps is linear to the instance size, increasing by 1000 for every 1 additional production line or machine included. The evaluation contains 500 steps for all instances. We apply a learning rate , the epsilon initial value of 0.1 that decreases by 0.00005 until either reaching a threshold of 0.01 or when the first 20% of the steps are completed. The replay buffer contains 900 steps and each time a random sample batch of 300 is generated for training the main network. For the parameter setting, each task requires one inventory unit from the preceding task, the task processing time is randomly generated uniformly from interval . The time horizon is 10 times the size of instance.

* + 1. Results
			1. Evaluation Result

From the evaluation result in Figure 3, the RL agent is able to cumulate reward for all different instance sizes by sequencing tasks to maximise final products whereas the random agent fails to secure any positive sum of reward. Both agents in Figure 4 have a confidence interval of 99%. Despite the large variation caused by randomly generated processing time, the RL agent produced a higher objective value on average for all instance sizes.

 

(Left) Figure 3. Reward evaluation for the RL-based agent and random agent; (Right) Figure 4. Objective function evaluation for the RL-based agent and random agent.

* + - 1. Computational Time

The computational time from Table 2 indicates that our RL framework is time efficient in the way that once trained, the agent is able to build up task sequences within 1.5 seconds on average for an instance with 6 machines and 6 jobs.

Table 1. Computational time (second) consumed by 500 evaluation steps for the RL-based agent and the random agent without prior-training. Equivalent to solving the batch process problem for 500 times.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Size = 2 | Size = 3 | Size = 4 | Size = 5 | Size = 6 |
| DQN | 21.72 | 44.84 | 111.89 | 311.07 | 632.83 |
| Random | 19.94 | 33.37 | 72.1 | 172.35 | 379.45 |

* 1. Conclusions

In conclusion, we have addressed a batch process scheduling problem by first formulating it into a Markov Decision Process. We subsequently employ deep Q-Learning to estimate the expected future rewards for each action and train the agent to build up the production schedule in an online manner by iteratively engaging in trial-and-error interactions with the environment. Preliminary experiment results demonstrate the ability of our learning-based approach in solving the problem with good computational efficiency and model adaptability.

Plans for future research involve the generalisation of learning framework to larger industrial instance size, the adaptation of a larger action set, and the hybridisation of RL-based learning approach with mixed-integer linear programming models to reach optimality more efficiently.

Acknowledgement

Financial support from the Engineering & Physical Sciences Research Council grant EP/V051008/1 is gratefully acknowledged by the authors.

References

N. Basán, M. Cóccola, A. del Valle, C. Méndez, 2020, Scheduling of flexible manufacturing plants with redesign options: A MILP-based decomposition algorithm and case studies, Computers & Chemical Engineering, 136, p.106777

P. Castro, I. Grossmann, Q. Zhang, 2018, Expanding scope and computational challenges in process scheduling, Computers & Chemical Engineering, 114, pp.14-42

D. Gupta, C. Maravelias, J. Wassick, 2016, From rescheduling to online scheduling, Chemical Engineering Research and Design, 116, pp.83-97

C. Hubbs, C. Li, N. Sahinidis, I. Grossmann, J. Wassick, 2020, A deep reinforcement learning approach for chemical production scheduling, Computers & Chemical Engineering, 141, p.106982

V. Jain, I. Grossmann, 2001, Algorithms for hybrid MILP/CP models for a class of optimization problems. INFORMS Journal on computing, 13(4), pp.258-276.

S. Jiang, C. Xu, L. Zhang, Y. Ma, 2023, A decomposition-based two-stage online scheduling approach and its integrated system in the hybrid flow shop of steel industry. Expert Systems with Applications, 213, p.119200.

C. Maravelias, I. Grossmann, 2004, A hybrid MILP/CP decomposition approach for the continuous time scheduling of multipurpose batch plants, Computers & chemical engineering, 28(10), pp.1921-1949

V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, 2015, Human-level control through deep reinforcement learning, Nature, 518(7540), pp.529-533

J. Ren, C. Ye, F. Yang, 2021, Solving flow-shop scheduling problem with a reinforcement learning algorithm that generalizes the value function with neural network, Alexandria Engineering Journal, 60(3), pp.2787-2800

R. Sutton, A. Barto, 2018, Reinforcement learning: An introduction. MIT press

Z. Wang, B. Cai, J. Li, D. Yang, Y. Zhao, H. Xie, 2023, Solving non-permutation flow-shop scheduling problem via a novel deep reinforcement learning approach, Computers & Operations Research, 151, p.106095

C. Watkins, P. Dayan. 1992, Q-learning, Machine Learning, 8(3-4):279–292

Z. Yang, L. Bi, X. Jiao, 2023, Combining Reinforcement Learning Algorithms with Graph Neural Networks to Solve Dynamic Job Shop Scheduling Problems, Processes, 11(5), p.1571