Systems & Control Transactions

Original Research Article
Peer Reviewed Conference Proceeding
ESCAPE 36 - European Symposium on Computer Aided Process Engineering
Sheffield, UK. 21-24 June 2026

PSE PRESS

A Graph Reinforcement Learning Framework for Batch Process Scheduling in State-Task Networks

Syu-Ning Johnn[a], Victor-Alexandru Darvariu[b], and Vassilis M. Charitopoulos[a]*

[a] Department of Chemical Engineering & The Sargent Centre for Process Systems Engineering, University College London, London, United Kingdom
[b] Oxford Robotics Institute, Department of Engineering Science, University of Oxford, Oxford, United Kingdom
* Corresponding Author: v.charitopoulos@ucl.ac.uk.

ABSTRACT

Batch production scheduling of resources to meet fluctuating product demand is a critical topic in the process industry. Existing optimisation approaches, based on heuristic and exact methods, trade off solution optimality and scalability to large problems. In this work, we investigate deep reinforcement learning as a powerful alternative in order to *learn* heuristics for batch scheduling. We formulate the batch scheduling problem as a Markov decision process operating on a state-task network representation encoded using graph neural networks, capturing relevant structural inductive biases. We propose a centralised training with decentralised execution architecture, in which agents placed on machines individually choose which tasks to complete using a global view of the network, cooperating towards task schedules that optimise the final production quantity. Preliminary results demonstrate that the proposed end-to-end framework learns to construct task schedules comparable to the optimal solution on small instances unseen during training, exhibiting strong potential for extension to more general graph structures and better scalability.

Keywords: Batch Process Scheduling, Reinforcement Learning, Markov Decision Process, Deep-Q Networks, Graph Neural Networks

INTRODUCTION

Batch scheduling is a critical functionality in the supply chain of process industries that deals with the allocation of limited resources over time in order to meet market demand for products that follow a batch recipe (Harjunkoski et al., 2010). As demand for diverse products continues to rise in the chemical industries, developing adaptive batch process operations constitutes a fundamental problem in these sectors (Yoo et al., 2021). Particularly, the scheduling of batch processes is a critical factor in process operations and plays a vital role in optimising production system performance (Wu and Maravelias, 2021).

While batch scheduling problems can be effectively tackled by existing exact and heuristic methods, learning-based AI methods, and in particular Reinforcement Learning (RL), provide a powerful alternative. RL is capable of solving complex, large-scale problems in dynamic environments and discovering complex scheduling policies beyond human intuition (Dogru et al., 2024; Johnn and Charitopoulos, 2025). RL algorithms such as Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO) can be used to train an agent to learn the optimal decision-making policy through trial-and-error interactions with an environment that provides feedback. Given that a natural framing of batch scheduling is through State-Task Networks (STNs), neural architectures explicitly designed to operate on graphs are highly suitable. The combination of Graph Neural Network (GNN) architectures and RL algorithms can yield a powerful approach capable of solving complex graph optimisation problems that lack satisfactory traditional algorithms (Darvariu et al., 2024).

RL has demonstrated strong adaptability and substantial potential for improving industrial-level applications (del Real Torres et al., 2022). The integration of RL into process scheduling has recently garnered emerging attention in the literature. Hameed et al. (2023) developed a PPO framework with a GNN-based scheduler for production planning problems to minimise the production makespan. Zhang et al. (2023) introduced a DQN framework with multi-agent graphs for a job-shop scheduling

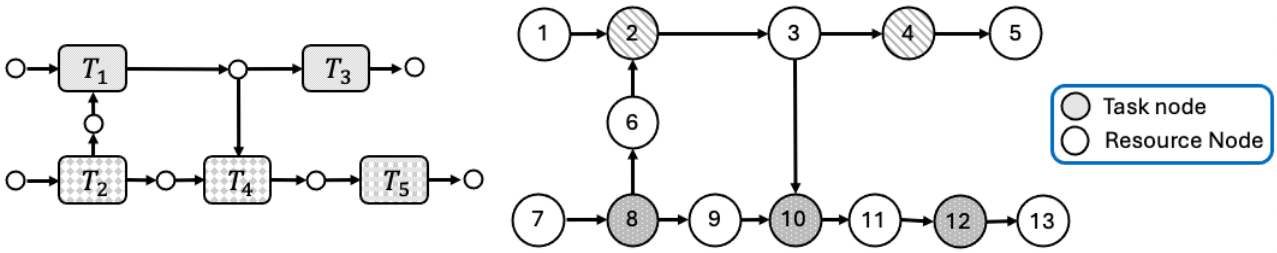

Figure 1. The classic STN model (left) can be converted into the directed graph structure (right) with nodes characterised by different features. Tasks TA1 and TA3, converted into nodes 2 and 4, are assigned to machine 1. Tasks TA2, TA4, TA5, converted to nodes 8, 10, 12 are controlled by machine 2. During the scheduling proces, the initial resources stored at node 1 and 7 are converted into intermediate nodes (3, 6, 9, 11) via different production processes, and eventually converted into final products as nodes 5 and 13.

process, where the job and machine-associated agents collaboratively work on job allocation, sequencing and routing. More recently, Rangel-Martinez and Ricardez-Sandoval (2025) proposed a PPO framework for multi-product batch scheduling processes, in which a hybrid agent handles multiple discrete and continuous decisions to build online schedules for STNs under uncertainty.

In this work, we propose an end-to-end DQN framework with a GNN encoder to address the batch process scheduling problem. We adopt the STN formalism by representing each production task as a node and each material flow as a link. The key differentiating characteristic of our work is the use of a multi-agent decomposition to manage the extensive decision space that arises when the problem is considered from a global perspective, enhancing scalability and computational tractability. Our approach obtains solutions close to optimality on problem instances unseen during training, significantly outperforms a single-agent baseline using joint actions, and offers strong potential for scalability and generalisability.

PROBLEM FORMULATION

Batch Process Scheduling Problem

STN graphs are a common way to visualise and model the batch scheduling problem (Méndez et al, . 2006). In this work, we reformulate classic STNs as directed graph structures, as shown in Figure 1. We preserve the graph topology and transform all task and resource components into indexed nodes distinguished by their node features (categories).

In this problem setting, we are given a set of machines that collaboratively schedule tasks to produce final products within a fixed time horizon. Each machine can be assigned more than one task over the time horizon. However, machines can process at most one task at any given time, which transforms the input resource(s) into output resource(s). When a task node is activated at a specific non-final time step, we assume it consumes exactly one unit of material from each precedent resource node at the start of this step, and produces exactly one unit of intermediate (or final) resource to each sequential resource node at the end of step. We assume only one type of material is stored at each resource node, and only one type of action is performed at a task node. A task node is available to process when all required input resource nodes have sufficient inventory levels and the time horizon has not yet been reached. Each task takes exactly one fixed time step to finish. Multiple machines collaboratively schedule task execution, with each machine processing at most one task at any given time to achieve maximum final production output.

Mixed-Integer Linear Programming Model

We adapt the STN formulation introduced by Kondili et al. (1993) and present a mixed-integer linear programming (MILP) model for the basic batch scheduling problem compatible with the directed graph representation given in Figure 1.

Let T denote the discrete set of time steps $\{0, 1, 2, \dots, T_{max}\}$. K is the set of tasks and L is the set of resource nodes, which can be partitioned into initial (L_{init}), intermediate (L_{inter}), and final product nodes (L_{final}). The set of machines is denoted M, where each machine $m \in M$ has a predefined set of tasks $K_m \subseteq K$ to execute. The binary variable $x_{k,t}$ takes the value 1 if task k is being executed at the start of time step t. The continuous variable $J_{l,t} \geq 0$ represents the inventory level of resource $l \in L$ at the start of time t. The objective is to maximise the total production of final products at the end of the time horizon. The MILP formulation is presented below:

$$\max \sum_{l \in L_{final}} J_{l,T_{max}} \quad (1)$$

$$\sum_{k \in K_m} x_{k,t} \leq 1, \forall m \in M, \forall t \in T \quad (2)$$

$$V_{l,t+1} = V_{l,t} + \sum_{k \in K_{BN}} P_{k,l} x_{k,t} - \sum_{k \in K_{FN}} C_{k,l} x_{k,t}, \forall l \in L \quad (3)$$

$$V_{l,0} = I_{l,0}, \forall k \in L_{init} \quad (4)$$

$$V_{l,t} \geq \sum_{k \in K_{FN}} C_{k,l} x_{k,t}, \forall l \in L \backslash L_{final} \quad (5)$$

$$V_{min} \leq V_{l,t} \leq V_{max}, \forall l \in L, \forall t \in T \quad (6)$$

Constraint (2) restricts each machine to choose no more than one task per step. Constraint (3) ensures flow conservation between task and resource nodes, enforcing and updating the inventory balance. Each resource node l is updated based on its precedent input task nodes producing resources and the subsequent output task nodes that consume it. Constraint (4) specifies the case in which all initial resource nodes start with fixed inventory levels prior to any task execution. Constraints (5) and (6) ensure that any task can only be selected if the required input resources are available and that storage limits are respected.

RL METHODOLOGY

Multi-agent Markov Decision Process

RL is a methodology in which an agent iteratively improves a decision-making policy through trial-and-error interactions with a dynamic environment (Sutton and Barto, 2018). The agent's objective is to map states to actions such as to maximise expected long-term rewards. Markov Decision Processes (MDP) provide the underlying model of the sequential decision-making task and can be solved using a variety of RL techniques.

An MDP is defined as a tuple (S, A, P, R), where a state $s_t \in S$ represents a complete environment snapshot at a specific time step. At each non-terminal state s_t, the agent chooses an action $a_t \in A$ from the set of available actions $A(s_t)$ and receives an immediate reward r_t specified by a reward function $R(s_t, a_t)$ for the selected state-action pair at step t. The subsequent state s_{t+1} is governed by a transition function P that captures movement from one state to another given an action.

An agent interacts with the environment through episodes, each of which represents a sequential chain of states, actions, and rewards over time. Reward feedback from the environment is incorporated iteratively to refine strategic decisions and train the agent to learn an optimal policy $\pi(a_t|s_t)$ that maximises cumulative future rewards. The state-action value function $Q(s_t, a_t)$ captures the expected reward associated with a given state-action pair under the optimal policy π. This quantity is also referred to as Q-value.

Multi-agent RL (MARL) extends classic RL by modelling how multiple agents interact within a shared environment and learn strategies to maximise their expected returns. In this setting, each agent's selected actions and the associated expected rewards can be influenced by the actions of all agents. The dynamic of the shared global state evolves by the joint actions of all agents following a Markovian transition function. Through this process, agents are trained to optimise individual or collective rewards, depending on the decision-making context being competitive, collaborative, or hybrid. Concretely, we consider a fully collaborative framing of batch scheduling, which is appropriate as all machines are under the control of a single organisation.

Deep Q-Learning and Graph Neural Networks

Q-learning is a class of RL algorithms that estimate the state-action value function (Q) using samples of agent experience. Estimates are iteratively refined using the update rule below, where the discount factor γ mediates the trade-off between immediate and future rewards:

$$Q^{new}(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \bullet \max_{a' \in A(s')} Q(s',a') - Q(s,a)) \quad (7)$$

Upon convergence, a greedy policy with respect to the learned Q-function is optimal.

Deep Q-learning (DQN) is a model-free RL approach that uses deep neural networks for scaling to large state and action spaces (Mnih et al. 2015). It incorporates a replay buffer and periodically updated target network. The state-action Q-value $Q(s,a)$ is estimated by the main network, while the maximum Q-value $Q(s',a')$ at the subsequent step is estimated by the target network. The target network parameters are periodically copied from the main network.

Graph Neural Networks (GNNs) are a learning representation designed to operate on graph-structured data (Velickovic, et al 2017). A GNN computes an embedding that captures node and edge features as well as underlying structural regularities. It aggregates information from neighbouring nodes through multiple message-passing layers to derive a higher-dimensional representation for the graph that exhibits desirable properties such as permutation invariance. GNNs can learn representations that achieve strong extrapolation capabilities to feature values and topologies unseen in the training set.

End-to-end DQN Framework

Our formulation of the batch scheduling problem operates on a directed graph $G = (V, E)$ with N_n nodes and N_e edges, where $V = K \cup L$. In each state, every machine $m \in M$ is responsible for a disjoint subset of task nodes K_m. The edges E connect resource nodes to task nodes, representing direct resource supply and production relationships.

We propose a multi-agent MDP framework as an alternative to using a single agent to control all machine nodes simultaneously in order to reduce the dimensionality of the problem. Instead, agents are placed at each machine and asked to individually act in a loop. The full

production schedule is established by the agents taking turns to select tasks given those chosen by the previous agents in the loop. This inner loop is executed once per time step t until the final time step T_{max} is reached.

We obtain a fully observable, finite-horizon, turn-based Markov game (Littman, 1994) with shared rewards. Therefore, we formulate the problem as the tuple $(S, A^{(i)}, P, R)$ containing an *agent-indexed* set of valid actions. The components are defined as follows:

- States: each state at time step t of an MDP episode is denoted $s_t^{(i)} = (G, X_t^{node}, X_t^{edge}, X_t^{global}, J_t, \tau_t)$, with i corresponding to index of the agent whose turn it is to act. $X_t^{node}, X_t^{edge}, X_t^{global}$ are node, edge, and global graph features respectively. Some features are updated as the episode progresses while others remain constant (see Table 2 for details). J_t denotes the set of inventory levels for each resource node at step t, and τ_t denotes whether an agent is the final decision-maker in the agent loop (i.e., if $i = |M|$).
- Actions: at time step t, agent i selects a task $a_t^{(i)}$ from the action space $A^{(i)}(s_t^{(i)})$ containing all production tasks available to it, along with an additional no-op action that allows the corresponding machine to stay idle for this step. Action masks are applied that prevent an agent from selecting tasks outside its allocated set K_m, and to dynamically filter out infeasible actions due to insufficient input resources.
- Transitions: once an agent has chosen an action $a_t^{(i)}$ that is not no-op, the selected task node is activated. The resource inputs from preceding resource nodes are converted into output resources stored at the sequential resource nodes, yielding J_{t+1}. The node, edge, and global features are also updated accordingly. If no-op is chosen, the machine status is set to "idle" for step t. Two types of transitions occur depending on the value of τ_t. If $\tau_t = 0$, meaning that this is not the last agent to act at global time step t, the turn passes to the next agent. If $\tau_t = 1$, the global time step advances to $t + 1$, and the turn transfers to the first agent. The terminal state $s_{T_{max}}$ is reached and the episode completes when either all time steps are exhausted or the remaining time is insufficient for the minimum batch to process. The terminal state captures the complete production schedule.
- Rewards: at the terminal state, the environment generates a reward based on the production schedule $R(s_t, a_t) = F(s_{T_{max}})$. As specified in (1), this corresponds to the total number of final products at the final time step. All intermediate rewards are 0.

Figure 1 provides a visualisation of a sample process schedule involving 2 machine agents, 5 task nodes, and 7 resource nodes. We treat each machine as an individual agent that controls the execution of task k at each time step t, representing the same decision as the binary variables $x_{k,t}$ in the MILP formulation.

A visualisation of the end-to-end framework is presented in Figure 2. To begin with, the state is represented as a graph and encoded by the GNN. The resulting feature vectors are passed to multilayer perceptrons (MLPs) representing the agents' Q-networks. These networks are iteratively updated based on the observed rewards to maximise the total amount of final products produced at the end of the time horizon.

For this work, we apply a centralised training, decentralised execution (CTDE) architecture (Amato, 2024). As mentioned, this is appropriate because all machine agents collaborate on the same graph topology to determine the schedule for a single problem instance. Given each agent will correspond to a different node representation based on the assigned task nodes and global state, parameterising each agent's policy separately is unnecessary. Instead of training N MLPs based on the same graph to yield N different agent policies $\pi_{i \in N}$, we therefore only train one global MLP that estimates the Q-values for the allowed actions. This is achieved by a mask layer after the MLP that is applied such that Q-values are only generated for the allocated task nodes, whereas the infeasible actions are filtered out by the mask. The CTDE framework is advantageous as only a single deep neural network is trained, yielding good computational efficiency and training stability, while still allowing for placing the neural network on every machine in an operational deployment setting.

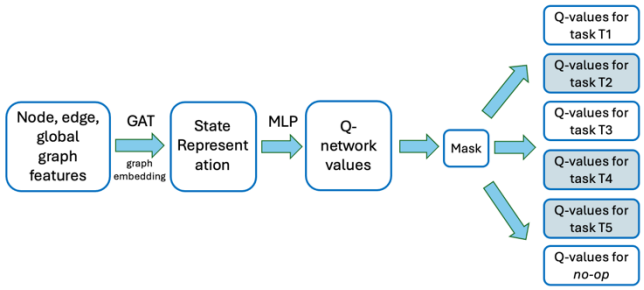

Figure 2. Visualisation of our end-to-end training framework with the CTDE architecture. All agents use the same global information and share the GNN encoder and MLP layers. A mask layer filters out disallowed tasks that are not allocated to this agent. In this example, corresponding to Figure 1, agent 1 is acting and therefore tasks T2, T4, and T5 are masked out.

Table 2: Node, edge and global features used as input for the graph representation of our end-to-end framework.

Name	Type	Re-scaled	Static over training	Feature description
agent_pinpoint	node	F	F	Indicator of the active agent at this node. Uses one-hot encoding: type 1 - agent is currently acting; type 2 - not acting; type 3 - resource node.
decision_status	node	F	F	Indicates the decision status of the agent associated with this node. Uses one-hot encoding: type 1 - currently deciding; type 2 - task nodes already decided; type 3 - task nodes not yet decided; type 4 - resource nodes.
node_type	node	F	T	Node category. Uses one-hot encoding: type 1 - task node, type 2 - resource node.
agent_allocation	node	T	T	Index of the machine agent ID assigned to a task node (1, 2, ...); 0 for resource nodes.
task_node_status	node	F	F	Task occupancy status. Uses one-hot encoding: type 1 - task occupied, type 2 - task idle, type 3 - resource nodes.
product_rate	node	F	T	Production rate associated with the node.
consume_rate	node	F	T	Consumption rate associated with the node.
inventory_level	node	T	F	Normalised inventory level for resource nodes; −1 for task nodes.
final_sales_price	node	T	T	Unit profit for each final resource node, 0 otherwise.
job_start_time	node	T	F	Job starting time for task node at this step; -1 for resource nodes.
job_end_time	node	T	F	Job completion time for task node at step; -1 for resource nodes.
current_time	global	T	F	Tracks the current state's time step for an episode.
remaining_time	global	T	F	Tracks the remaining time in this episode: $(T_{max} - T_{current}) / T_{max}$.
current_agent_id	global	T	F	Tracks the current agent id in one-hot encoding within the agent loop.
remaining_agent	global	T	F	Tracks the agent decision status sequence to count how many more agents should act within the agent loop.
task_size	global	T	T	Tracks the number of task nodes for this instance.
resource_size	global	T	T	Tracks the number of resource nodes for this instance.
physical_edges	edge	F	T	Stores node linkage showing consumption (resource to task) or production (task to resource).

To highlight the benefits of the proposed multi-agent decomposition, a single-agent baseline is also considered in which a DQN agent handles the scheduling of all production tasks. Instead of applying the proposed decomposition, the baseline uses a centralised architecture with a single agent selecting from a combinatorial joint action space as opposed to iterating M times over individual agents. Consequently, the replay buffer records the global state s_t only, together with the joint action tuple $\left(a_t^{(1)}, a_t^{(2)}, \ldots, a_t^{(m)}\right)$ generated by the single DQN agent. For the MLP output layer, rather than assign outputs to individual tasks (layer size $|K|+1$), the baseline model uses an output layer over every task combination of size $\prod_{i \in M}\left(|A^{(i)}|+1\right)$. In the illustrative example in Figure 1, this corresponds to an MLP output dimension of 6 for the multi-agent decomposition framework, and $3 \times 4 = 12$ for the single-agent baseline. A comparative performance analysis is presented in the next section.

CASE STUDY

Experiment Setting

We evaluate the proposed end-to-end framework on a range of process scheduling problems modelled as STN graphs, thereby demonstrating its effectiveness across varying problem parameters.

We adapt the batch scheduling problem instance given in Figure 1, with a fixed time horizon consisting of

10 steps for each episode. The graph topology remains fixed while the initial inventory level at each intermediate resource node is uniformly randomly generated from the interval $[0, 5]$. The initial resource nodes are unlimited and the final product quantities are set to zero.

Three distinct sets of training, validation and evaluation instances are generated. The training set is used to update the policy parameters during learning. The validation set is periodically used to quantify the performance of the policy specified by the learned parameters at a particular learning step. Finally, the evaluation set is used for reporting the quality of the best-performing learned model. The training, validation, and evaluation sets contain distinct problem instances.

As a GNN, we opt for the graph attention network (GAT) architecture (Velickovic et al., 2017). Parameters are updated by stochastic gradient descent using the Q-learning loss and the Adam optimiser. Hyperparameter values obtained as a result of hyperparameter tuning and other settings are specified in Table 1 below. The detailed list of GNN input features is given in Table 2.

We train our end-to-end DQN framework using an epsilon-greedy policy to ensure both exploration and exploitation. At evaluation time, a greedy policy is adopted, which always selects the action with the highest estimated Q-value.

We consider the following baselines. To derive an upper bound on performance, we use the MILP formulation in (1)-(6) to find the optimal solution for each problem configuration. The MILP model is implemented in Pyomo and solved with the Gurobi optimizer. As a lower bound, we use a uniform random policy that allows a machine to select any task with equal chance. This uses the same time horizon and also benefits from the action masking to filter out disallowed tasks. Lastly, we compare against the centralised single-agent method described in the previous section. This agent uses identical hyperparameters to the proposed method.

Table 1: end-to-end framework hyperparameter values.

Hyperparameter	Value
Learning rate	0.0005
Discount factor γ	1.0
Initial exploration epsilon	1.0
Final exploration epsilon	0.1
GAT attention heads	4
GAT number of layers	5
GAT hidden dimension	64
MLP 1st hidden layer dimension	256
MLP layer number	2
Sample batch size	4
Replay buffer size	20% of training
Validation frequency (episodes)	25

Experiment Results

We train the policy for 50, 000 episodes. Performance on the validation set containing 16 instances is checked every 25 episodes and the best-performing model is stored. Figure 3 shows the performance on the validation set during training. The best-performing policy is stored and evaluated on the separate evaluation set containing 64 instances.

We conduct a Monte Carlo simulation for the trained end-to-end DQN framework and compare the results with the three baseline methods. The lines shown in Figure 4 represent the average value accumulated over the evaluation instances. Notably, the single-agent baseline yields fewer production outputs and therefore less profit given the same number of training steps. Due to the sharp increase in action space size, we expect this effect to be even more pronounced on larger problem instances.

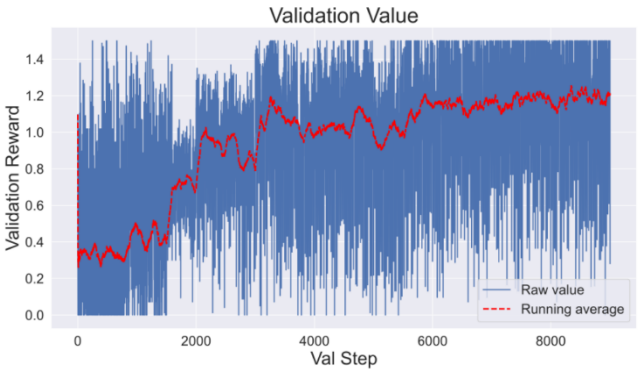

Figure 3. Validation performance during training based on the average reward computed from 32 predefined problem instances.

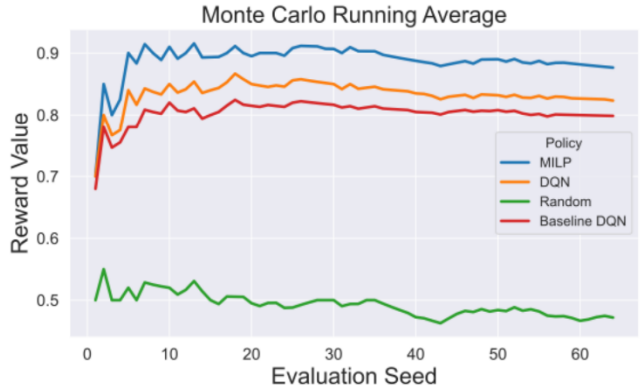

Figure 4. Average reward obtained on the evaluation set by the 3 considered methods. Averaged from 10 rounds.

From the box plot in Figure 5, we observe that our method constructs optimal solutions in 60.9% of cases and schedules with at most one task missing in 98.4% of cases. The random policy constructs solutions that, on average, only meet half of the production quantity compared to the optimal solution. Our framework leads to

solutions whose quality is clearly an improvement over the random baseline.

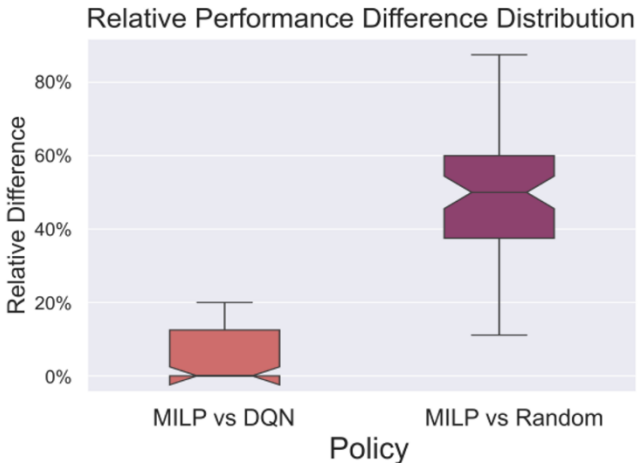

Figure 5. Evaluation performance comparing the MILP against the trained policy and a uniform random policy.

The Gantt charts in Figures 6 to 8 showcase the batch schedules constructed by the three methods using the problem instance on which the MILP scheduling method obtains the median performance among all problem instances.

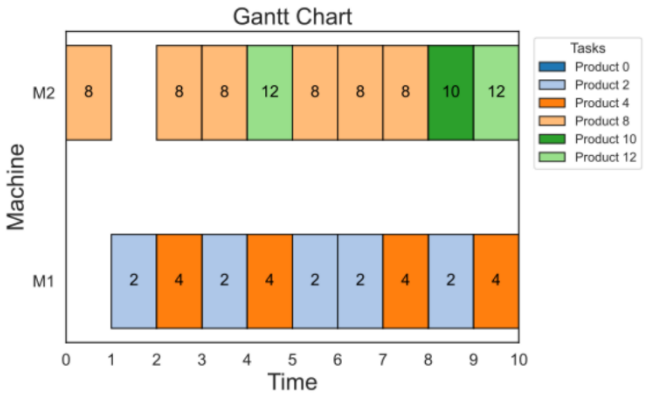

Figure 6. MILP scheduling plan on the median problem instance as a Gantt chart. Optimal solution.

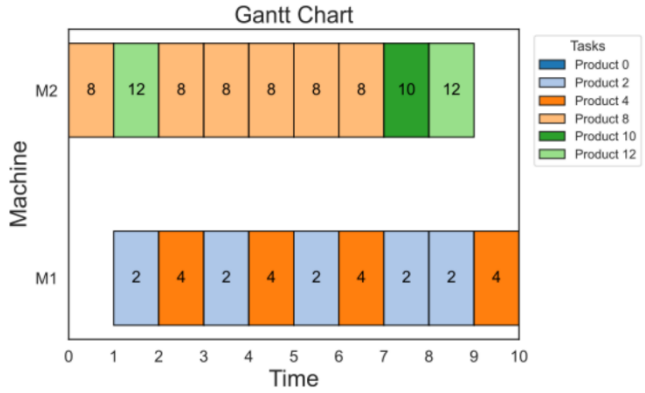

Figure 7. Scheduling plan output by our method on the median problem instance.

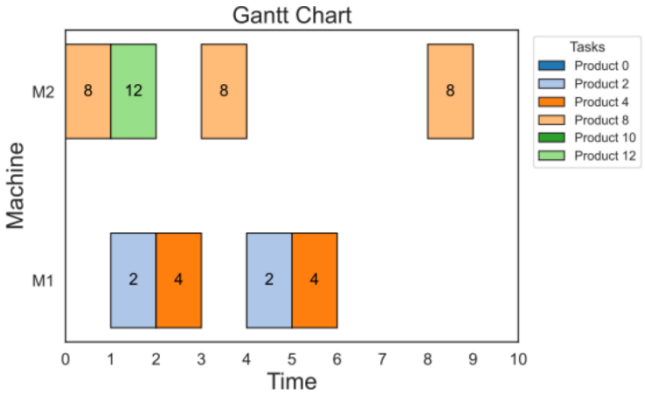

Figure 8. Scheduling plan output by uniform random action choices on the median problem instance.

CONCLUSION

In this work, we addressed the batch process scheduling problem using a deep reinforcement learning method. We formulated it as a Markov decision process and proposed an end-to-end DQN framework underpinned by a GNN encoder, in which multiple agents collaboratively control manufacturing machines to process tasks and construct work sequences to maximise final production quantity. Preliminary results on small-scale instances with up to 13 nodes demonstrate strong performance, with the trained agents achieving high-quality solutions in 98.4% of the evaluation scenarios and reaching optimality in 60.9% of scenarios.

In future research, we aim to leverage the generalisation ability of GNNs to train and evaluate our model on graph topologies with different sizes and characteristics. Our goal is to demonstrate the scalability of our framework to problem instances that are too large to be solved optimally using a MILP formulation but are amenable to a learning-based approach that can produce outputs quickly.

ACKNOWLEDGEMENTS

Financial support from the UK EPSRC grant EP/V051008/1 is gratefully acknowledged. V.-A. Darvariu acknowledges support from the Natural Environment Research Council (NERC) Twinning Capability for the Natural Environment (TWINE) Programme NE/Z503381/1 and the Innovate UK AutoInspect Grant 1004416.

AUTHOR IDENTIFIERS

Author ORCIDs:
Johnn S.-N.: 0000-0001-7958-2434
Darvariu V.-A.: 0000-0001-9250-8175
Charitopoulos V. M.: 0000-0001-9051-917X

REFERENCES

1. Amato, C., 2024. An introduction to centralized training for decentralized execution in cooperative multi-agent reinforcement learning. arXiv preprint arXiv:2409.03052.
2. Darvariu, V.-A., Hailes, S., & Musolesi, M. (2024). Graph reinforcement learning for combinatorial optimization: A survey and unifying perspective. *Transactions on Machine Learning Research (TMLR)*.
3. del Real Torres A, Andreiana DS, Ojeda Roldán Á, Hernández Bustos A, Acevedo Galicia LE. A review of deep reinforcement learning approaches for smart manufacturing in industry 4.0 and 5.0 framework. Applied Sciences 12:12377 (2022). https://doi.org/10.3390/app122312377
4. Dogru O, Xie J, Prakash O, Chiplunkar R, Soesanto J, Chen H, Velswamy K, Ibrahim F, Huang B. Reinforcement learning in process industries: review and perspective. IEEE/CAA J. Autom. Sinica 11:283-300 (2024). https://doi.org/10.1109/jas.2024.124227
5. Hameed MSA, Schwung A. Graph neural networks-based scheduler for production planning problems using reinforcement learning. Journal of Manufacturing Systems 69:91-102 (2023). https://doi.org/10.1016/j.jmsy.2023.06.005
6. Harjunkoski I, Maravelias CT, Bongers P, Castro PM, Engell S, Grossmann IE, Hooker J, Méndez C, Sand G, Wassick J. Scope for industrial applications of production scheduling models and solution methods. Computers & Chemical Engineering 62:161-193 (2014). https://doi.org/10.1016/j.compchemeng.2013.12.001
7. Johnn SN, Charitopoulos VM. A hybrid deep q-learning approach to online planning and rescheduling of single-stage multi-product continuous processes. Computers & Chemical Engineering 204:109415 (2026). https://doi.org/10.1016/j.compchemeng.2025.109415
8. Kondili E, Pantelides CC, Sargent RWH. A general algorithm for short-term scheduling of batch operations—i. MILP formulation. Computers & Chemical Engineering 17:211-227 (1993). https://doi.org/10.1016/0098-1354(93)80015-f
9. Méndez CA, Cerdá J, Grossmann IE, Harjunkoski I, Fahl M. State-of-the-art review of optimization methods for short-term scheduling of batch processes. Computers & Chemical Engineering 30:913-946 (2006). https://doi.org/10.1016/j.compchemeng.2006.02.008
10. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D. Human-level control through deep reinforcement learning. Nature 518:529-533 (2015). https://doi.org/10.1038/nature14236
11. Rangel-Martinez D, Ricardez-Sandoval LA. Recurrent reinforcement learning strategy with a parameterized agent for online scheduling of a state task network under uncertainty. Ind. Eng. Chem. Res. 64:7126-7140 (2025). https://doi.org/10.1021/acs.iecr.4c04900
12. Sutton, R. and Barto, A., 2018. Reinforcement learning: An introduction. MIT Press.
13. Casanova A, Cucurull G, Drozdzal M, Romero A, Bengio Y. On the iterative refinement of densely connected representation levels for semantic segmentation. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) :1091-109109 (2018). https://doi.org/10.1109/cvprw.2018.00144
14. Littman ML. Markov games as a framework for multi-agent reinforcement learning. Machine Learning Proceedings 1994 :157-163 (1994). https://doi.org/10.1016/b978-1-55860-335-6.50027-1
15. Wu Y, Maravelias CT. A general framework and optimization models for the scheduling of continuous chemical processes. AIChE Journal 67: (2021). https://doi.org/10.1002/aic.17344
16. Yoo H, Byun HE, Han D, Lee JH. Reinforcement learning for batch process control: review and perspectives. Annual Reviews in Control 52:108-119 (2021). https://doi.org/10.1016/j.arcontrol.2021.10.006
17. Zhang JD, He Z, Chan WH, Chow CY. Deepmag: deep reinforcement learning with multi-agent graphs for flexible job shop scheduling. Knowledge-Based Systems 259:110083 (2023). https://doi.org/10.1016/j.knosys.2022.110083